



**FACULTAD
DE INGENIERIA**

Universidad de Buenos Aires

TP2

-Crypto Monitor-

75.10 Técnicas de Diseño
Primer cuatrimestre de 2024

Abraham Osco	102256
Ricardo Luizaga	87528
Agustín Rodríguez	101570
Luca Lazcano	107044

Trabajo Práctico N°: 2 Crypto Monitor

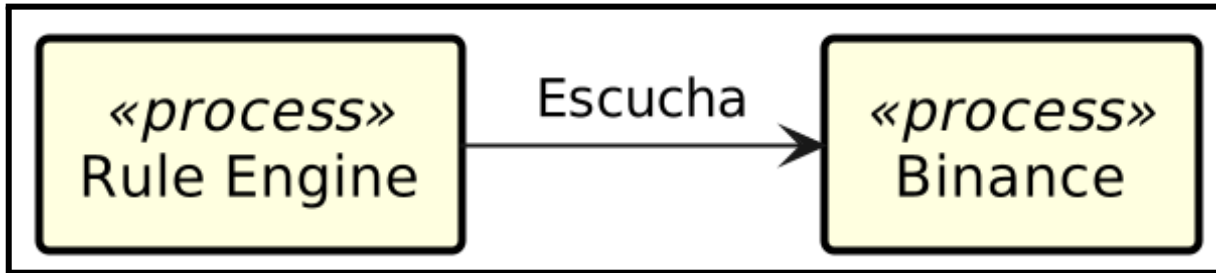
1) Arquitectura:

Para el trabajo práctico se utilizó una **arquitectura de layers**. Aunque no hay capa de interfaz gráfica ni capa de acceso a datos, hay una capa lógica (RuleEngine) y una capa de servicio (Binance).

Vista lógica: La vista está más explicada en la **sección de diseño**.

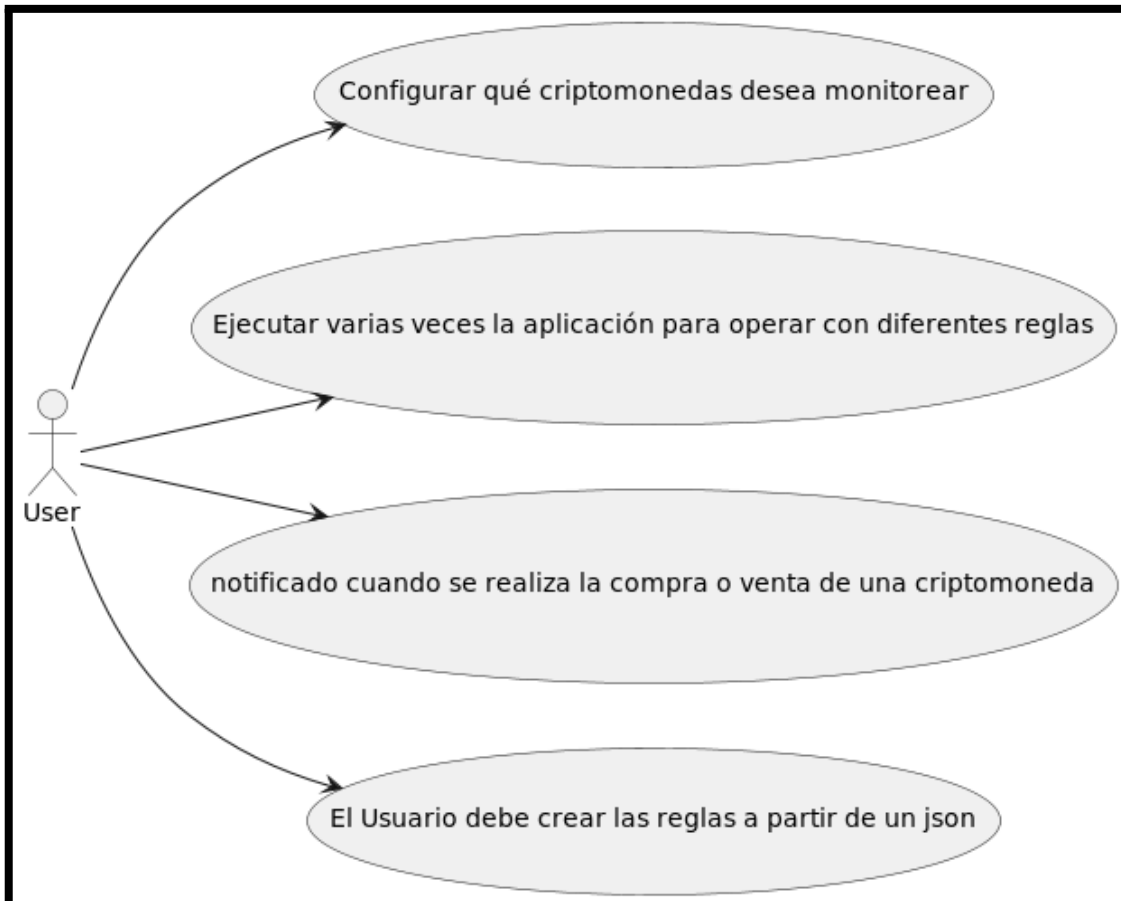
Vista de procesos:

- 1) Proceso de **RuleEngine**.
- 2) Proceso de Binance (Es un thread lanzado por la clase **Binance Service**).



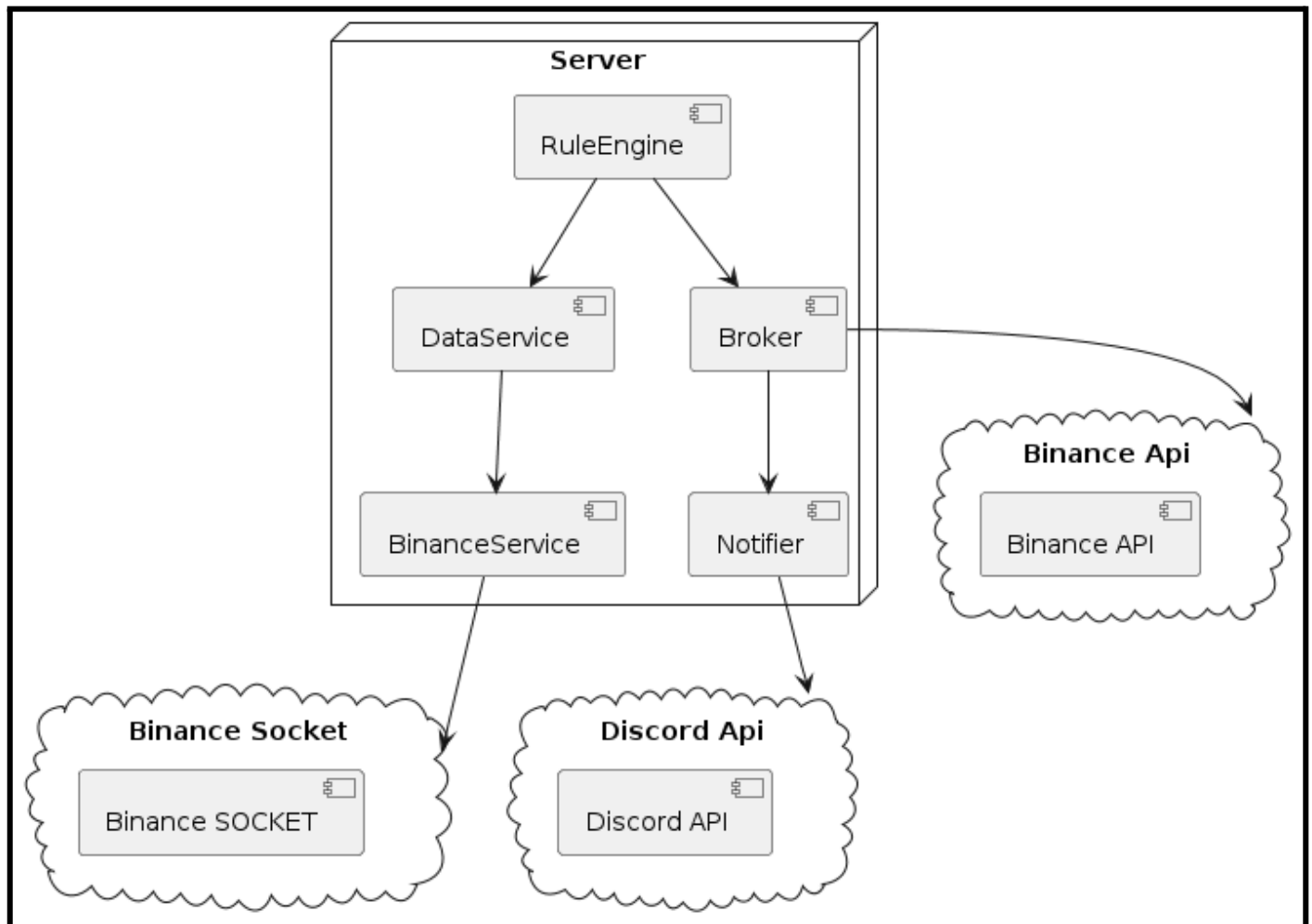
Vista de casos de uso:

- 1) El usuario **puede configurar qué criptomonedas desea monitorear**.
- 2) El usuario puede ejecutar varias veces la aplicación para operar con diferentes reglas.
- 3) El Usuario debe crear las reglas a partir de un json.
- 4) El usuario debe ser notificado cuando se realiza la compra o venta de una criptomoneda a través de un mensaje de discord.



Vista Física

Se muestra una vista física con algunas entidades de la capa lógica, donde además se muestran las distintas API que interactúan con la aplicación.



2) Diseño

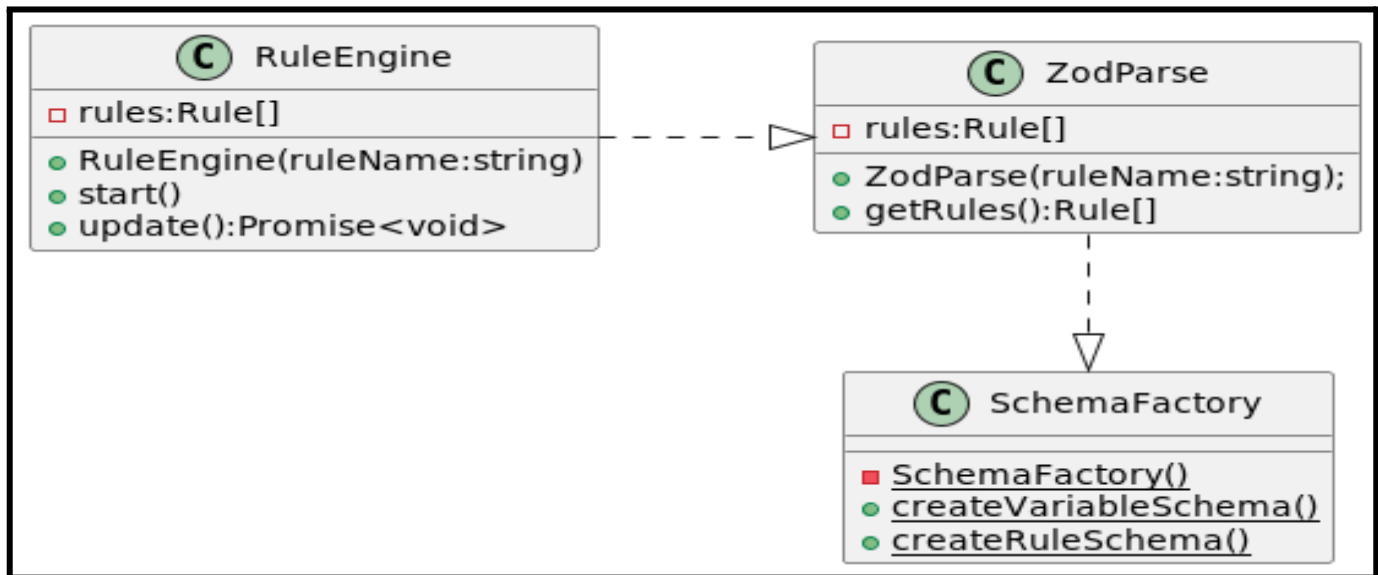
Parseo de reglas

Para el parseo de reglas se crearon las siguientes entidades:

- 1) **SchemaFactory**: Encapsula todos los schemas necesarios para parsear el json de las reglas y tiene los lazy necesarios para los casos recursivos. (call, value, etc).
- 2) **ZodParseo**: Lee el json y se encarga de partir los campos para aplicarle su respectivo schema, además de cargar las variables del json en el singleton de **Variables** y las reglas en un array de **Rule**.

Rule Engine

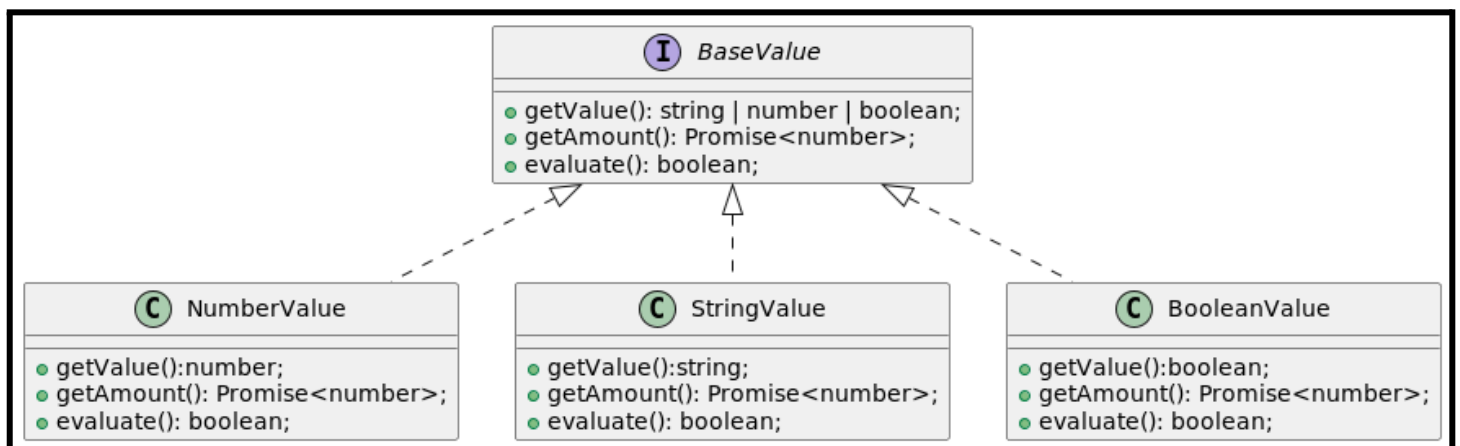
Es el motor de reglas, recibe el ruleName en su constructor y delega su parseo, cada vez que haya un cambio significativo de moneda intenta aplicar las reglas, aplicándolas solo en el caso de que la condición sea verdadera.



Value & Data:

BaseValue

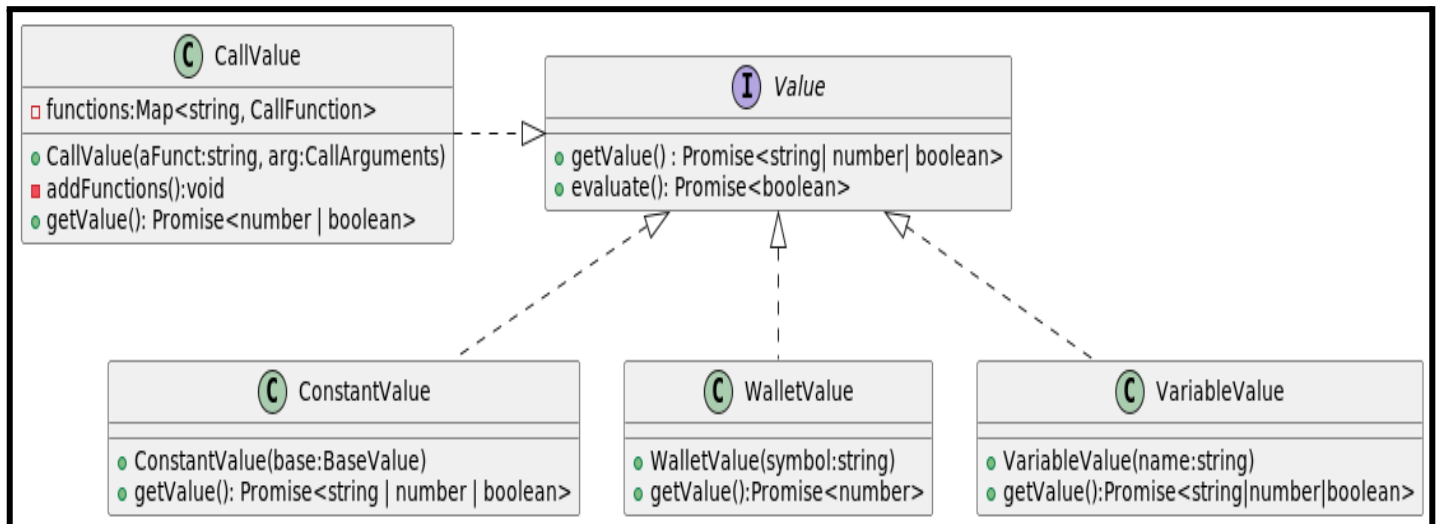
Para el modelo de valores inicialmente se empezó desde lo más básico creando una abstracción **BaseValue** que encapsula los tipos de datos más básicos, es decir los casos de Number/String/Boolean que se pueden asignar a **value** como CONSTANT/VARIABLE. De esta manera se le da comportamiento.



Value

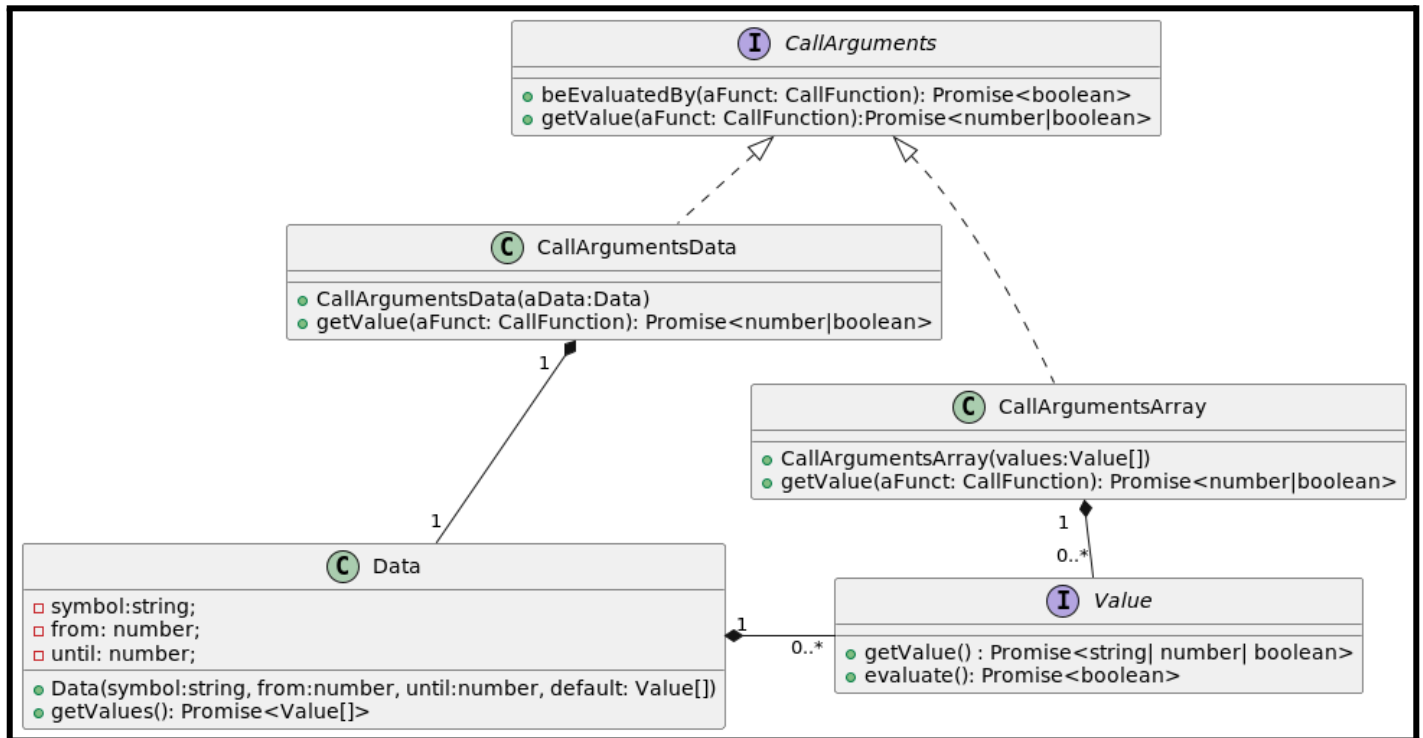
Value es una abstracción y tiene sus implementaciones concretas **CallValue**, **ConstantValue**, **VariableValue**, **WalletValue**. Cada implementación concreta representa un tipo de value respectivo.

- 1) **ConstantValue**: tiene una abstracción de tipo **BaseValue**, delegando así su comportamiento.
- 2) **VariableValue**: recibe el nombre de la variable a hacer referencia y delega en el singleton de **Variables** el comportamiento.
- 3) **WalletValue**: se delega el comportamiento al singleton de **Broker**.
- 4) **CallValue**: se aplica el patrón Command, cada función matemática está encapsulada en una clase a ejecutar.
 - a) Además **CallValue** recibe por inyección de dependencia una abstracción del tipo **CallArguments**.
 - b) Esto para el caso de recibir un array de Value o un objeto de tipo Data (Que se mapea luego a un array de Value).



CallArguments: es una **abstracción** y tiene dos implementaciones concretas

- 1) **CallArgumentsArray:** Encapsula el array de Value (Value[]) que recibimos en el json.
- 2) **CallArgumentsData:** Encapsula el objeto de tipo Data junto al default Array de values, para luego aplicarle comportamiento.

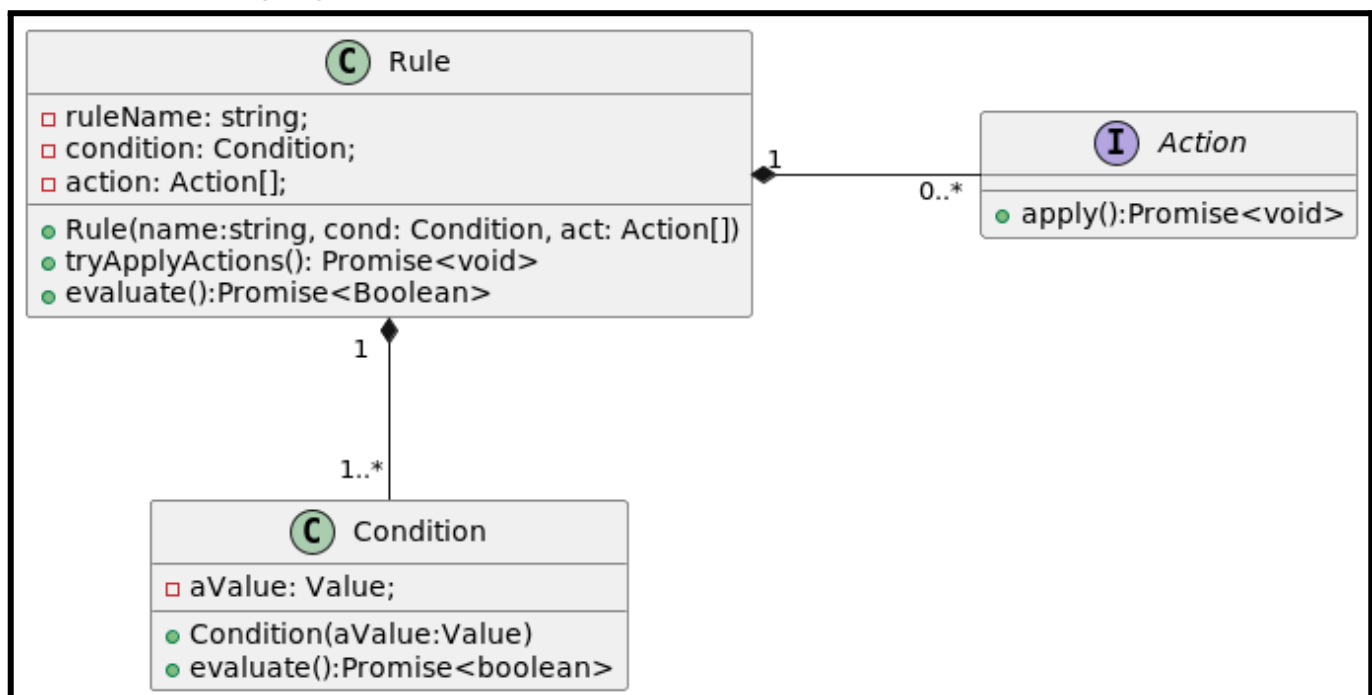


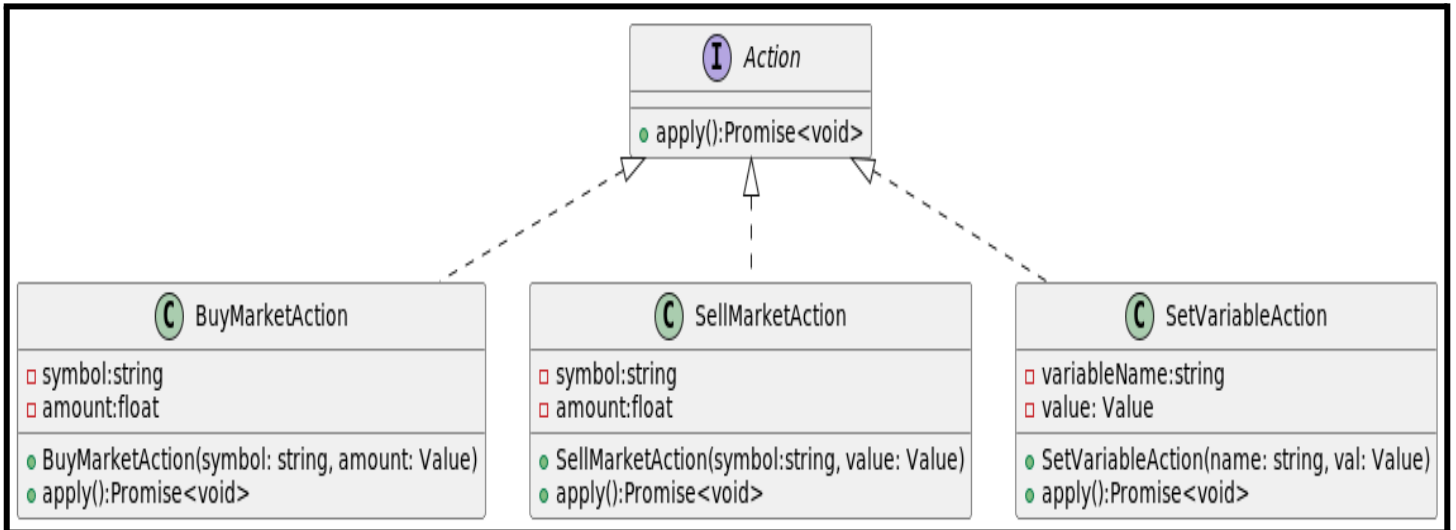
Rule

Cada regla del archivo json es encapsulada en un objeto de tipo Rule, al crearse recibe la condición a ejecutar y un array de Action (Abstracción) asociadas a esa regla.

Condition: aplica un **evaluate** al objeto value que almacena.

Action: Es una abstracción con 3 implementaciones concretas, cumpliendo open-close en el caso de agregar otro tipo de action más.





CallFunctions: Es la abstracción más genérica para poder representar las funciones que se reciben en call.

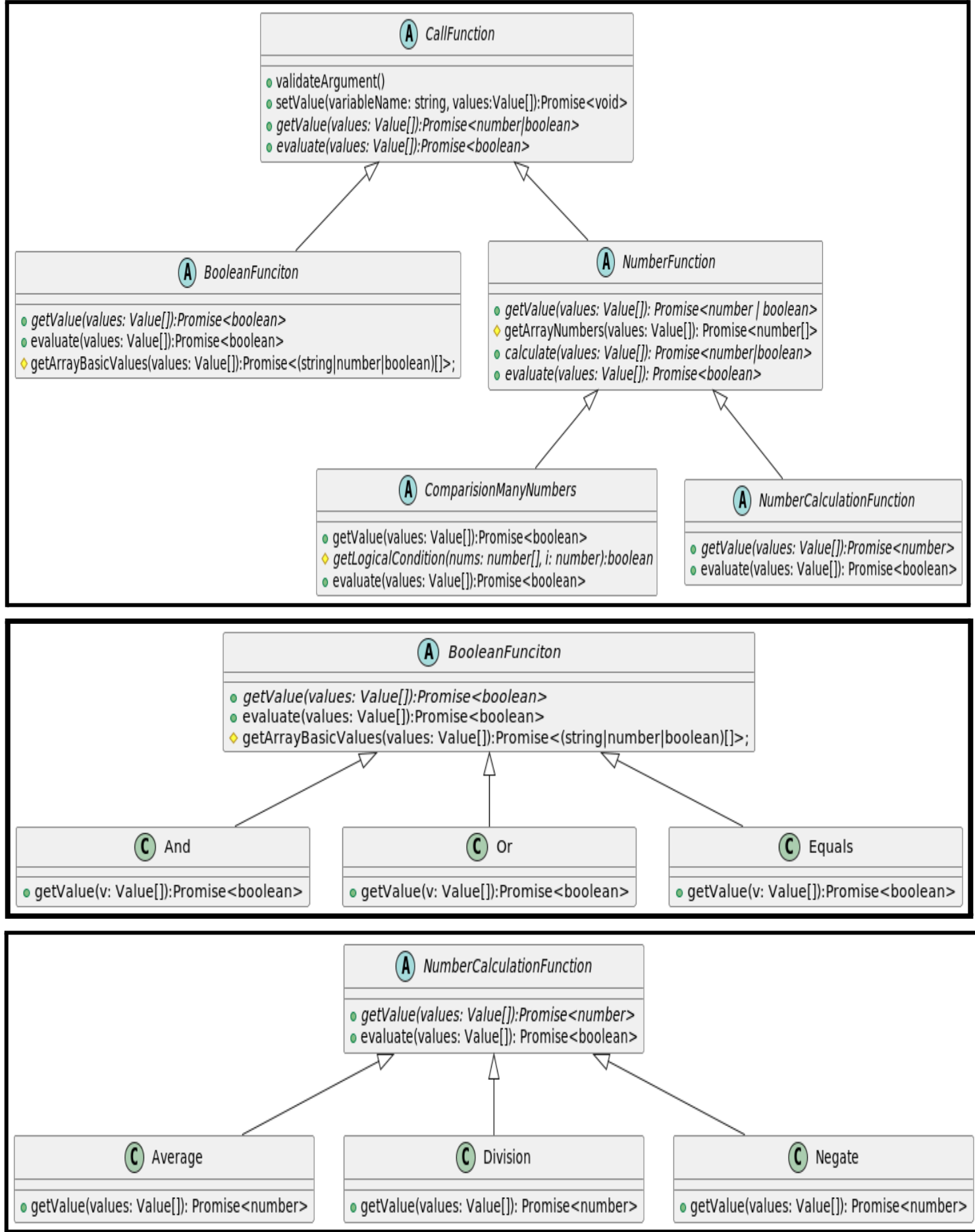
Tiene dos clases hijas que son a la vez abstracciones **BooleanFunction** y **NumberFunction** en el cual se encapsulan un comportamiento más particular en cada hija.

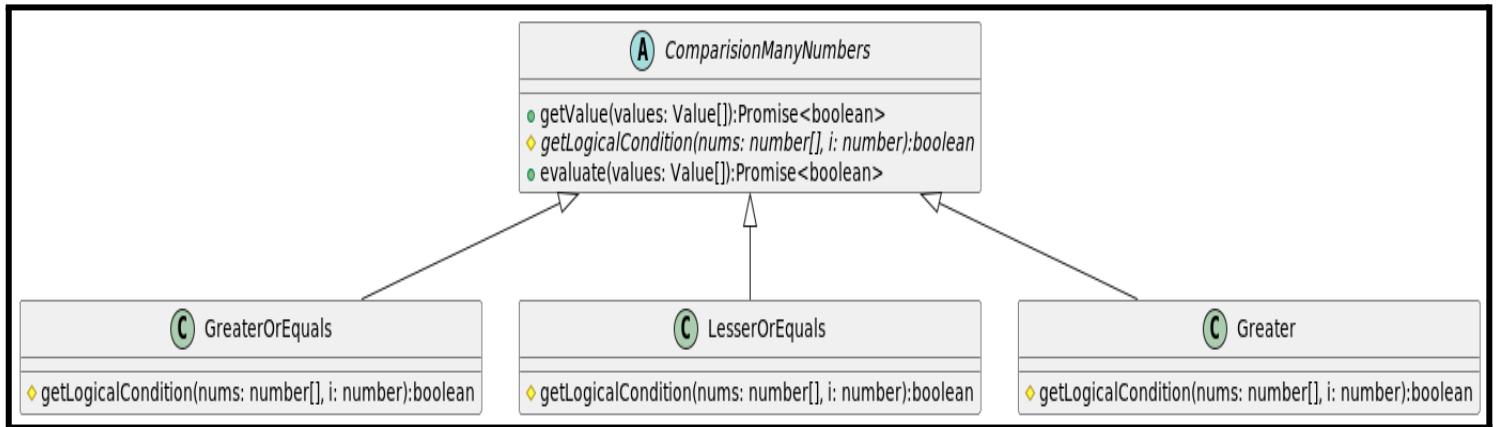
Luego para el caso de las funciones que retornan booleanos (And,Distinct, Equals, Or,Not) heredan de **BooleanFunction**.

En la abstracción **NumberFunction** tiene como clases hijas otras abstracciones:

- 1) **ComparisionManyNumbers:** cuyos objetos que encapsulan funciones como $>$, $>=$, $<$, $<=$ heredan de esta clase madre y retornan booleanos
- 2) **NumberCalculationFunction:** Esto se implementó para evitar el código duplicado (además en **ComparisionManyNumbers** se aplicó el **templateMethod** otra razón más para crear la abstracción. Así que el resto de las funciones matemáticas (MIN,MAX,LAST,..., etc) heredan de **NumberCalculationFunction** y retornan un número.

Diagrama de clases para la abstracción de CallFunctions y las clases hijas respectivas (otras abstracciones).





Market

El sistema está diseñado como un singleton para garantizar que solo haya una instancia activa en todo momento. Utiliza la API de **Binance** para obtener datos de precios en tiempo real y permite a los observadores (por ejemplo, servicios de trading automatizado) reaccionar a los cambios de precio.

El sistema consta de los siguientes componentes principales:

- 1) **DataService**: Gestiona los datos de precios históricos y notifica a los observadores cuando se detectan cambios significativos en los precios.
- 2) **BinanceService**: Gestiona la conexión a la API de Binance para recibir datos de precios en tiempo real.
- 3) **Observer**: Interfaz que define el método **update**, que es implementado por los observadores que desean ser notificados de los cambios de precio.
- 4) **RuleEngine**: Implementa la interfaz **Observer** que recorre las reglas cuando se detecta un cambio de precio significativo.

Diagrama de Clases del Market

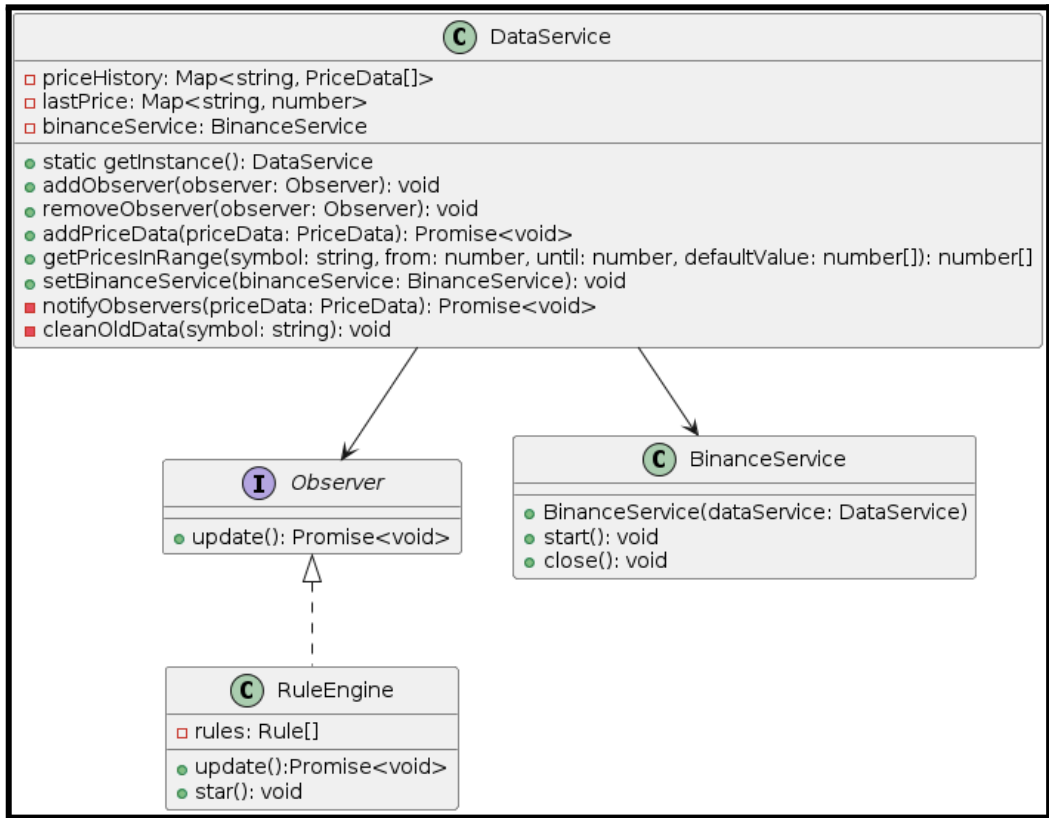
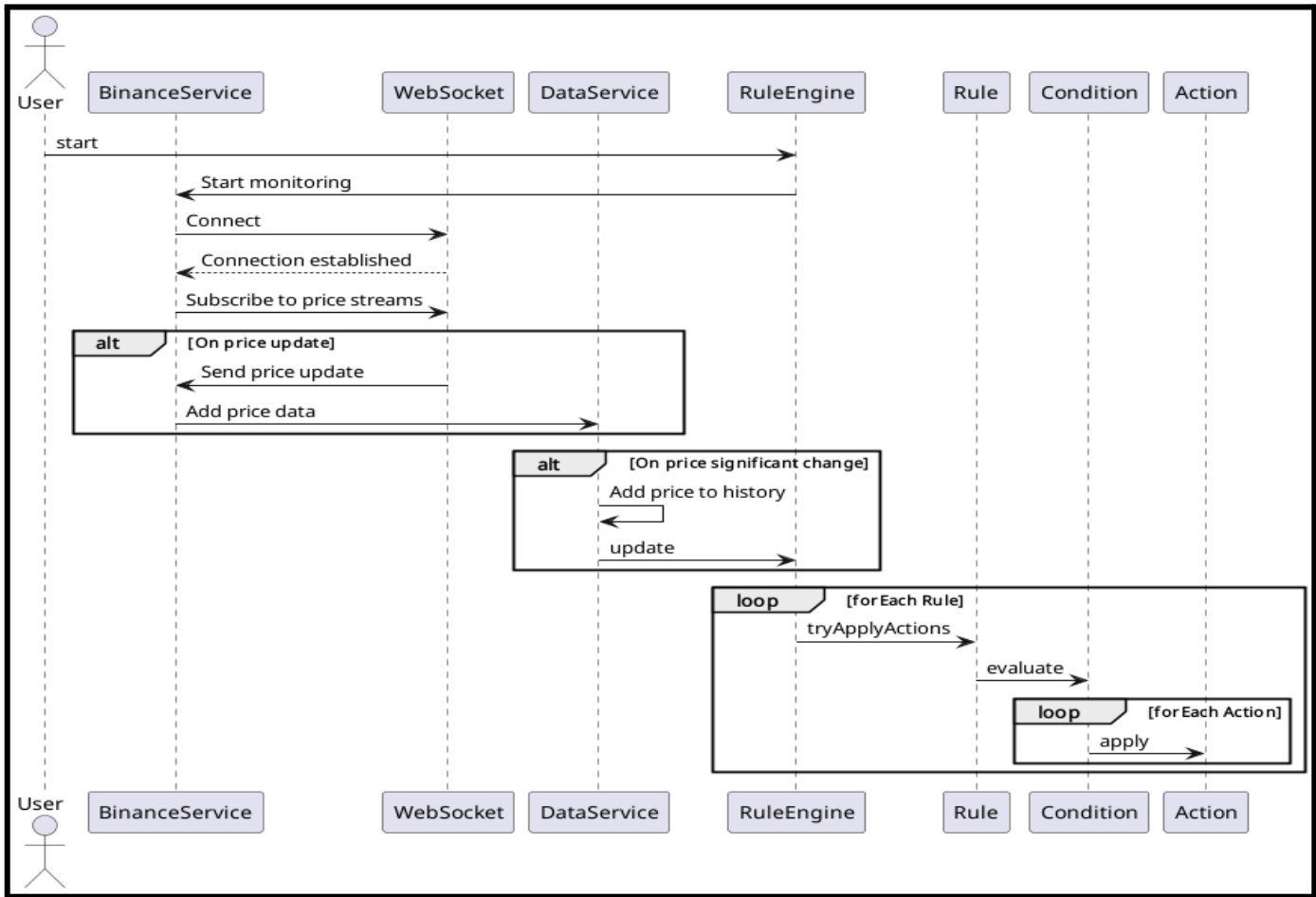
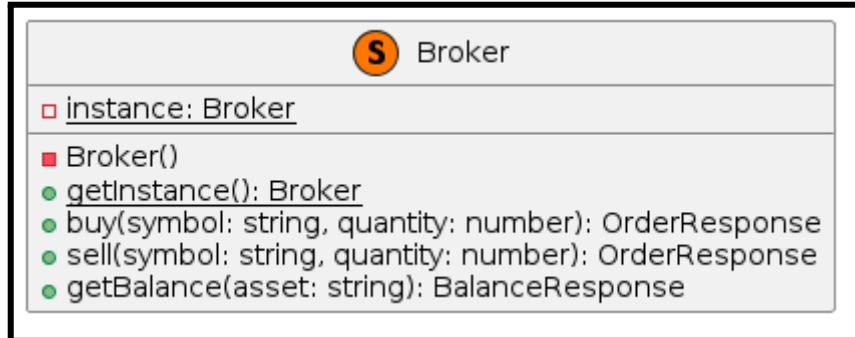


Diagrama de secuencias Market: donde se muestra la interacción con las entidades como WebSocket hasta la aplicación de las acciones.



Broker

El módulo Broker es un intermediario entre el bot y la API de Binance. Se encarga de realizar las peticiones necesarias para poder comprar, vender y obtener la cantidad disponible de las distintas criptomonedas. Para implementarlo se utiliza el patrón de diseño *Singleton*, el cual garantiza que solo habrá una única instancia de la clase durante la ejecución del programa.



buy

Recibe como parámetro un string **symbol** que es un par de criptomonedas. La primera indica la que deseo comprar, mientras que la segunda indica cómo pago dicha compra. El parámetro **quantity** indica la cantidad que deseo comprar.

```
const broker = Broker.getInstance();
broker.buy("bnbusdt", 0.1);
```

Indica que quiero comprar 0.1 Binance Coin, y a cambio voy a pagar USDT. ¿Cuánto USDT voy a pagar? Depende de la cotización del BNB en dólares al momento de realizar la compra. Asumiendo que en dicho momento 1 BNB vale 602 dólares, como estoy comprando 0.1 BNB voy a pagar 60.2 USDT.

sell

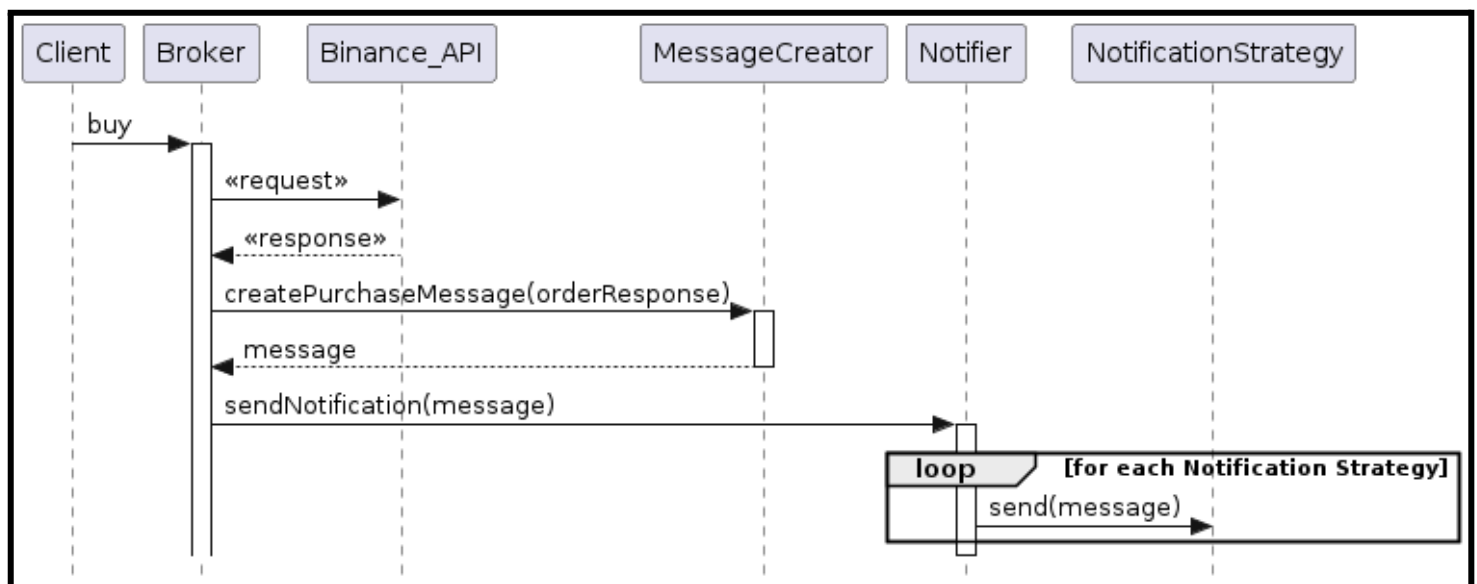
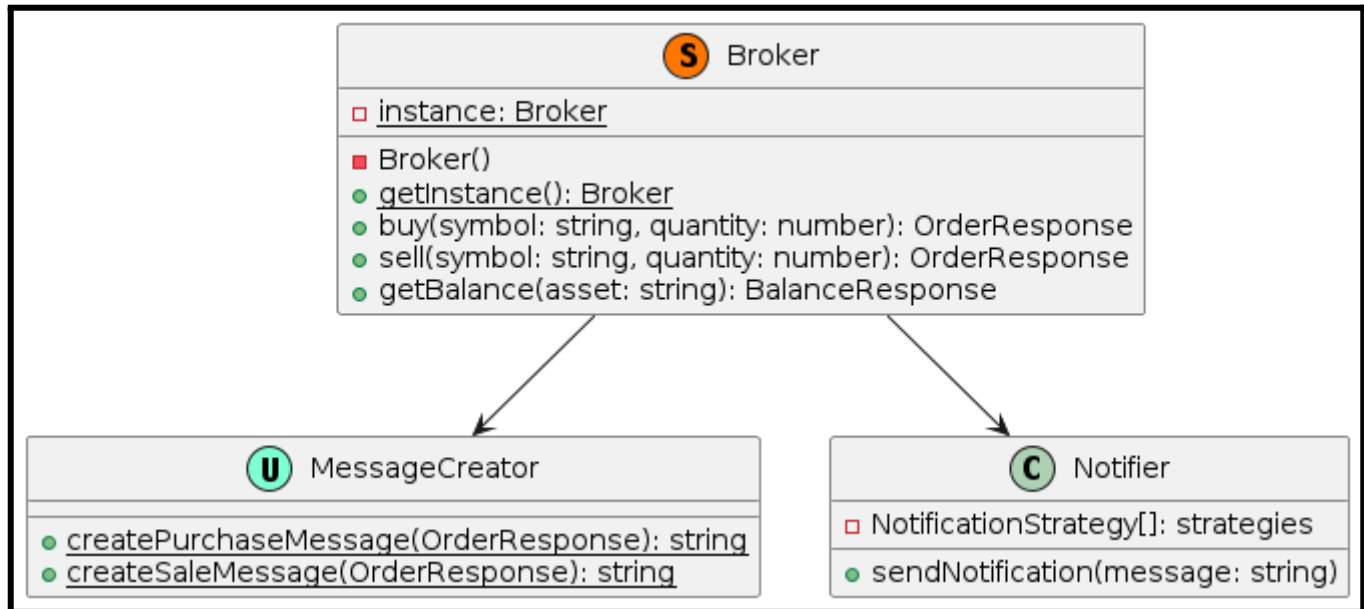
Recibe un par de criptomonedas en **symbol**, donde la primera indica la que deseo vender, mientras que la segunda indica la criptomoneda que recibo por dicha venta. El parámetro **quantity** indica la cantidad que deseo vender.

```
const broker = Broker.getInstance();
broker.sell("ethusdt", 0.1);
```

Indica que quiero vender 0.1 Ethereum, y a cambio voy a recibir USDT. La cantidad que recibo depende de la cotización del ETH en dólares al momento de realizar la venta. Asumiendo que en dicho momento 1 ETH vale 3900 dólares, como estoy vendiendo 0.1 ETH voy a ganar 390 USDT.

Message Creator

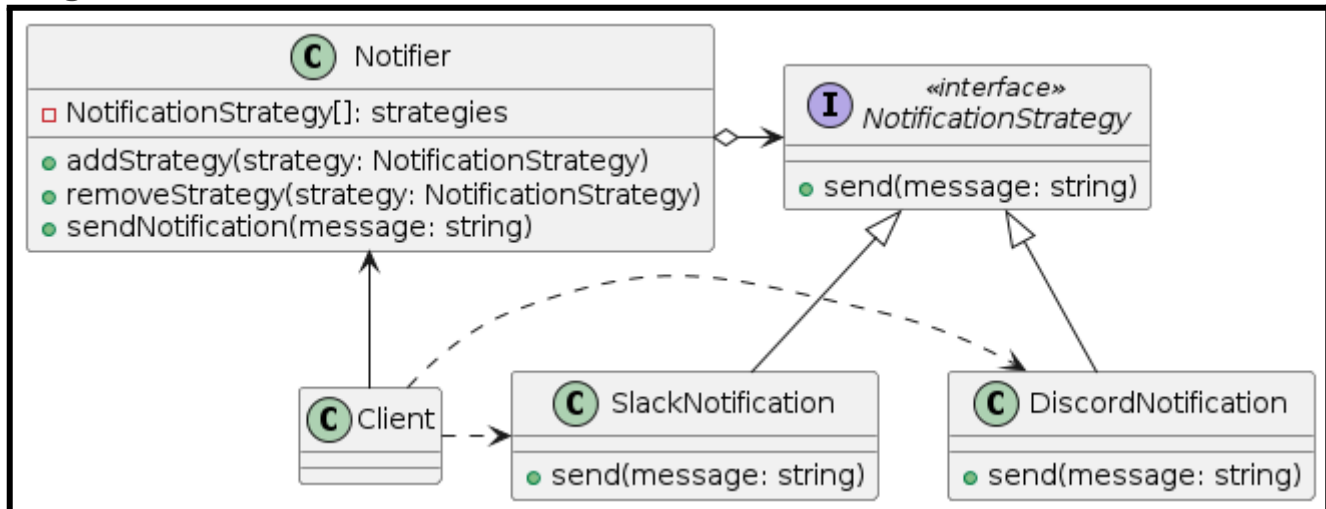
El módulo Message se encarga de construir mensajes apropiados, los cuales informan acerca de transacciones que hayan tenido lugar. Después de que el Broker realiza una compra o venta de una criptomoneda, este solicita a Message Creator un mensaje en un formato más legible con la información de la operación. Luego, dicho mensaje es pasado al Notifier, el cual se encarga de enviarlo a los distintos medios.



Notifier

El módulo de notificaciones se utiliza para informar al usuario sobre las acciones de compra y venta realizadas por el bot. La implementación sigue una variante del patrón Strategy, donde la diferencia radica en que se puede configurar al notificador con múltiples estrategias en simultáneo. En la entrega se implementan las estrategias para notificar por Discord y por Slack.

Diagrama de clases



Funcionamiento

Las estrategias deben implementar la interfaz **NotificationStrategy**, que determina cómo se realiza el envío del mensaje con cada medio. Por ejemplo, para las estrategias “**DiscordNotification**” y “**SlackNotification**”, se envía un mensaje HTTP de tipo POST, a una URL determinada, llamado “webhook”. Dichos “webhooks” se configuran desde el servidor de Discord o de Slack en el cual se quieren recibir las notificaciones, y se pasan a los constructores correspondientes al momento de crear las instancias de las estrategias.

La utilización del servicio de notificaciones se realiza mediante una instancia de **Notifier**. A esta se le pueden agregar o quitar estrategias, las cuales utilizará para enviar el mensaje cada vez que se llame a su método “`sendNotification`”.

Ejemplo de uso

```
const notifier = new Notifier();
notifier.addStrategy(new DiscordNotification(DISCORD_WEBHOOK));
notifier.addStrategy(new SlackNotification(SLACK_WEBHOOK));

await notifier.sendNotification("Hello world!")
```

En este ejemplo se crea una instancia del notificador, se le agregan dos estrategias y se envía el mensaje.

Link al servidor de discord:




<https://discord.gg/448594f2kf>

En las siguientes imágenes se muestra como coinciden los logs con el pusheo de mensaje al servidor de discord, es decir están sincronizados.

```
ⓧ abraham@abraham-Latitude-5480:~/Escritorio/TDD/tp2-tdd$ npm run dev

> tp2-tdd@1.0.0 dev
> ts-node-dev src/main.ts

[INFO] 18:14:02 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.2, typescript ver. 5.4.5)
[INFO] Cryptobot starting 🤖 with rule9.json
[INFO] WebSocket connected.
[INFO] New significant change for BNBUSDT: 598.1
[INFO] New significant change for BNBUSDT: 598.2
[INFO] [Broker]: We sold 📉 0.00198 btc and received 132.06204 usdt
Thu Jun 13 2024 18:14:12 GMT-0300 (hora estándar de Argentina)
[INFO] New significant change for LTCUSDT: 78.91
[INFO] [Broker]: We purchased 💰 0.00197 btc and paid 131.4022702 usdt
Thu Jun 13 2024 18:14:12 GMT-0300 (hora estándar de Argentina)
[INFO] [Broker]: We purchased 💰 0.00197 btc and paid 131.4022702 usdt
Thu Jun 13 2024 18:14:12 GMT-0300 (hora estándar de Argentina)
[INFO] [Broker]: We sold 📉 0.00198 btc and received 132.06204 usdt
Thu Jun 13 2024 18:14:12 GMT-0300 (hora estándar de Argentina)
[INFO] [Broker]: We sold 📉 0.00198 btc and received 132.06204 usdt
Thu Jun 13 2024 18:14:13 GMT-0300 (hora estándar de Argentina)
[INFO] [Broker]: We purchased 💰 0.00197 btc and paid 131.4022702 usdt
Thu Jun 13 2024 18:14:13 GMT-0300 (hora estándar de Argentina)
[INFO] New significant change for BNBUSDT: 598.1
[INFO] New significant change for BNBUSDT: 598.2
[INFO] [Broker]: We sold 📉 0.00198 btc and received 132.06162419999998 usdt
Thu Jun 13 2024 18:14:15 GMT-0300 (hora estándar de Argentina)
[INFO] [Broker]: We purchased 💰 0.00197 btc and paid 131.4196259 usdt
Thu Jun 13 2024 18:14:15 GMT-0300 (hora estándar de Argentina)
```

 TradingBot   hoy a las 18:14

We sold 📉 0.00198 btc and received 132.06204 usdt
Thu Jun 13 2024 18:14:12 GMT-0300 (hora estándar de Argentina)

We purchased 💰 0.00197 btc and paid 131.4022702 usdt
Thu Jun 13 2024 18:14:12 GMT-0300 (hora estándar de Argentina)

We purchased 💰 0.00197 btc and paid 131.4022702 usdt
Thu Jun 13 2024 18:14:12 GMT-0300 (hora estándar de Argentina)

We sold 📉 0.00198 btc and received 132.06204 usdt
Thu Jun 13 2024 18:14:12 GMT-0300 (hora estándar de Argentina)

We sold 📉 0.00198 btc and received 132.06204 usdt
Thu Jun 13 2024 18:14:13 GMT-0300 (hora estándar de Argentina)

We purchased 💰 0.00197 btc and paid 131.4022702 usdt
Thu Jun 13 2024 18:14:13 GMT-0300 (hora estándar de Argentina)

We sold 📉 0.00198 btc and received 132.06162419999998 usdt
Thu Jun 13 2024 18:14:15 GMT-0300 (hora estándar de Argentina)

We purchased 💰 0.00197 btc and paid 131.4196259 usdt
Thu Jun 13 2024 18:14:15 GMT-0300 (hora estándar de Argentina)

We purchased 💰 0.00197 btc and paid 131.46119290000001 usdt
Thu Jun 13 2024 18:14:16 GMT-0300 (hora estándar de Argentina)

We sold 📉 0.00198 btc and received 132.06162419999998 usdt
Thu Jun 13 2024 18:14:16 GMT-0300 (hora estándar de Argentina)

3) Ejecución de los test y el proyecto

En primer lugar:

- 1) Clonar el repositorio.
- 2) Ejecutar npm install.
- 3) crear el .env con el siguiente contenido:

```
DISCORD_WEBHOOK =  
https://discordapp.com/api/webhooks/1242218864545370132/25ivU1eGW5phEhuAIhKng7aM1IY7VI  
Ss1RVDhKRomOkRp24vrzFBS4UQoxQ9m04wrajg  
API_KEY = 3dP9uUrUFX9WxQCvfvpvQMhGH0c9LVk64j48M0VdGy9B8yL7S5tL7aPhapNfNBzO  
SECRET_KEY = jOcPLRcMEq2Jc1nccf62suRjO4IT6N0Pb8wW2JowkLKzXfGivcJFuFSFkl2joaF0  
BOT_CONFIG = rule8.json  
ACTIVATE_LOG = true  
SIGNIFICANT_CHANGE_THRESHOLD = 0.001
```

Variables de entorno:

- 1) **ACTIVATE_LOG**: donde le asignamos “true” si queremos ver los console.log o otro string en caso opuesto.
- 2) **BOT_CONFIG**: Le asignamos el nombre el archivo json que contiene las reglas a ejecutar, este se debe encontrar guardado en la carpeta **rules** (está en la ruta inicial del proyecto).
- 3) **SIGNIFICANT_CHANGE_THRESHOLD**: Le asignamos el cambio minimo de las monedas a detectar. Para no esperar tanto conviene asignarle 0.0001 a este campo.

Ejecutar los test

- 1) npm run test.

```
● abraham@abraham-Latitude-5480:~/Escritorio/TDD/tp2-tdd$ npm run test  
  
> tp2-tdd@1.0.0 test  
> jest  
  
PASS src/lib/parserZod/tests/ZodTest/ZodParser.test.ts (6.623 s)  
PASS src/lib/market/tests/DataService.test.ts  
PASS src/lib/parserZod/tests/CallTest/Call.test.ts (7.989 s)  
PASS src/lib/parserZod/tests/ValueTest/Value.test.ts  
PASS src/lib/parserZod/tests/ActionTest/SetVariableAction.test.ts  
PASS src/lib/parserZod/tests/ConditionTest/Condition.test.ts  
PASS src/lib/parserZod/tests/DataTest/Data.test.ts  
PASS src/lib/parserZod/tests/ZodTest/ZodParserData.test.ts  
PASS src/lib/notifier/tests/notifier.test.ts  
PASS src/lib/parserZod/tests/ActionTest/BuyMarketAction.test.ts  
PASS src/lib/parserZod/tests/VariablesTest/Variables.test.ts  
PASS src/lib/broker/tests/broker.test.ts (9.777 s)  
PASS src/lib/parserZod/tests/ActionTest/SellMarketAction.test.ts  
PASS src/lib/message/tests/message.test.ts  
A worker process has failed to exit gracefully and has been force exited. This is likely caused by  
. Active timers can also cause this, ensure that .unref() was called on them.  
  
Test Suites: 14 passed, 14 total  
Tests: 100 passed, 100 total  
Snapshots: 0 total  
Time: 10.763 s, estimated 11 s  
Ran all test suites.  
● abraham@abraham-Latitude-5480:~/Escritorio/TDD/tp2-tdd$
```

Para correr el proyecto con docker:

- 1) Crear la imagen de Docker mediante:

- docker build -t crypto-monitor .

2) Ejecutamos la imagen de docker:

- docker run crypto-monitor

```

abraham@abraham-Latitude-5480:~/Escritorio/TDD/tp2-tdd$ docker build -t crypto-monitor .
[+] Building 7.6s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 151B
=> [internal] load .dockerignore
=> => transferring context: 94B
=> [internal] load metadata for docker.io/library/node:lts
=> [1/6] FROM docker.io/library/node:lts@sha256:ab71b9da5ba19445dc5bb76bf99c218941db2c4d70ff4
=> [internal] load build context
=> => transferring context: 245.67kB
=> CACHED [2/6] WORKDIR /app
=> CACHED [3/6] COPY package.json .
=> CACHED [4/6] RUN npm install
=> [5/6] COPY . .
=> [6/6] RUN npm run build
=> exporting to image
=> => exporting layers
=> => writing image sha256:30a1cd79715e987cd42e163f7e10b9bd6e9835ca6c59a53b7000392dfcde4c60
=> => naming to docker.io/library/crypto-monitor
abraham@abraham-Latitude-5480:~/Escritorio/TDD/tp2-tdd$ docker run crypto-monitor

> tp2-tdd@1.0.0 start
> ts-node src/main.ts

[INFO] Cryptobot starting 🤖 wit the ruleName rule8.json
[INFO] WebSocket connected.
[INFO] New significant change for ETHUSDT: 3462.6
[INFO] New significant change for DOGEUSDT: 0.14128
[INFO] New significant change for BNBUSDT: 598.9
[INFO] New significant change for SOLUSDT: 147.21
[INFO] [Broker]: We sold 📉 0.00023 btc and received 15.425285800000003 usdt
Thu Jun 13 2024 15:38:09 GMT+0000 (Coordinated Universal Time)
[INFO] [Broker]: We purchased 📈 0.00022 btc and paid 14.7558884 usdt
Thu Jun 13 2024 15:38:09 GMT+0000 (Coordinated Universal Time)
[INFO] New significant change for BNBUSDT: 599
[INFO] [Broker]: We purchased 📈 0.00022 btc and paid 14.7558884 usdt
Thu Jun 13 2024 15:38:09 GMT+0000 (Coordinated Universal Time)

```

Como se observa en la imagen la aplicación se queda escuchando a que haya cambios significativos en el precio de las monedas monitoreadas para recién luego intentar aplicar las reglas.

Además se cumple la consigna. El programa puede ejecutar distintos archivos que contienen reglas en paralelo.

En la siguientes imagen se muestra dos docker corriendo diferentes archivos json:

```

abraham@abraham-Latitude-5480:~/Escritorio/TDD/tp2-tdd$ docker build -t crypto-monitor .
docker run crypto-monitor
[+] Building 0.7s (11/11) FINISHED
=> [internal] load .dockerignore
=> => transferring context: 94B
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 151B
=> [internal] load metadata for docker.io/library/node:lts
=> [1/6] FROM docker.io/library/node:lts@sha256:ab71b9da5ba19445dc5bb76bf99c218941db
=> [internal] load build context
=> => transferring context: 14.09kB
=> CACHED [2/6] WORKDIR /app
=> CACHED [3/6] COPY package.json .
=> CACHED [4/6] RUN npm install
=> CACHED [5/6] COPY . .
=> CACHED [6/6] RUN npm run build
=> exporting to image
=> => exporting layers
=> => writing image sha256:30a1cd79715e987cd42e163f7e10b9bd6e983
=> => naming to docker.io/library/crypto-monitor

> tp2-tdd@1.0.0 start
> ts-node src/main.ts

[INFO] Cryptobot starting 🤖 wit the ruleName rule8.json
[INFO] WebSocket connected.
[INFO] New significant change for BNBUSDT: 598.5
[INFO] New significant change for SOLUSDT: 146.82
[INFO] New significant change for DOGEUSDT: 0.14087

abraham@abraham-Latitude-5480:~/Escritorio/TDD/tp2-tdd$ docker build -t crypto-monitor .
docker run crypto-monitor
[+] Building 8.4s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 151B
=> [internal] load .dockerignore
=> => transferring context: 94B
=> [internal] load metadata for docker.io/library/node:lts
=> [1/6] FROM docker.io/library/node:lts@sha256:ab71b9da5ba19445dc5bb76bf99c218941db
=> [internal] load build context
=> => transferring context: 14.09kB
=> CACHED [2/6] WORKDIR /app
=> CACHED [3/6] COPY package.json .
=> CACHED [4/6] RUN npm install
=> [5/6] COPY . .
=> [6/6] RUN npm run build
=> exporting to image
=> => exporting layers
=> => writing image sha256:1be60476c542c4641216fb4e5314b5b0d6fc3ee039d0360b5cc9541dc
=> => naming to docker.io/library/crypto-monitor

> tp2-tdd@1.0.0 start
> ts-node src/main.ts

[INFO] Cryptobot starting 🤖 wit the ruleName rule10.json
[INFO] WebSocket connected.
[INFO] New significant change for DOGEUSDT: 0.14073
[INFO] New significant change for LTCUSDT: 78.01
[INFO] New significant change for SOLUSDT: 146.68
[INFO] New significant change for LTCUSDT: 78.02

```


Importante: Observar que al cambiar el .env con el nuevo nombre del json a ejecutar hay que **volver a ejecutar los docker build y run.**

4) Supuestos

Algunos supuestos para las ambigüedades que hubo en el enunciado:

1. Para calcular el **STDDEV** se usó la desviación estándar poblacional.
2. Si no hay reglas se lanza una excepción y se corta el programa.
3. Si a las variables le insertamos tipos de datos que no sean string, number o boolean, se lanzarán excepciones y se corta el programa.
4. Si no hay condición {} o el array de action está vacío se terminara el programa.

5) Patrones

Los patrones que aplicamos son:

- 1) **Singleton** en las clases: **Variables**, para tener una única instancia donde se almacena un diccionario de **BaseValue** para que sea fácil de setear nuevas variables en cualquier parte y que todas las reglas tengan la referencia a lo mismo. **Broker**, para garantizar que una sola instancia de la clase haga peticiones a la API de Binance y procese las respuestas apropiadamente.

```
4  export class Variables{
5      private static instance: Variables;
6      private variables: { [key:string] : BaseValue} = {};
7
8      private constructor(){}
9
10     public static getInstance(){
11         if (this.instance){
12             return this.instance;
13         }
14         this.instance = new Variables();
15         return this.instance;
16     }
17 }
```

- 2) **Strategy**: se utiliza una variante del patrón Strategy en el notificador, que utiliza varias estrategias en vez de una sola. Se implementa mediante la clase Notifier, que guarda las estrategias a utilizar, y la interfaz NotificationStrategy, la cual debe ser implementada por las estrategias de notificación llamadas en Notifier. También el patrón strategy en Action. Donde cada Rule tiene un array de **Action** (abstracción) (Ver diagrama de clase de Action).
- 3) **Template Method**: Por ejemplo en: la clase madre **ComparisionManyNumbers** se encapsula el algoritmo completo y cada hija solo implementa **el algoritmo placeholder** distinto que cambia en cada uno. (**getLogicalCondition**)

```

6
7 export abstract class ComparisionManyNumbers extends NumberFunction{
8
9     public async calculate(values: Value[]): Promise<boolean>{
10         const valuesCalculate = await this.getArrayNumbers(values);
11         for (let i = 0; i < valuesCalculate.length - 1; i++){
12             if ( this.getLogicalCondition(valuesCalculate, i) ){
13                 return false
14             }
15         }
16         return true;
17     }

```

```

export class Greater extends ComparisionManyNumbers{

    protected getLogicalCondition(numberValues: number[], index: number): boolean {
        return !(numberValues[index] > numberValues[index+1]);
    }
}

```

4) **Command**: En **CallValue** donde cada *función* está encapsula un “comando” a ejecutar.

```

25 export class CallValue implements Value{
26     private functionName:string;
27     private arrayArguments: CallArguments;
28     private static readonly EQUALS: string = "==";
29     private static readonly DISTINCT: string = "DISTINCT";
30     private static readonly LESS_THAN: string = "<";
31     private static readonly LESS_THAN_OR_EQUAL: string = "<=";
32     private static readonly GREATER_THAN: string = ">";
33     private static readonly GREATER_THAN_OR_EQUAL: string = ">=";
34     private static readonly NEGATE: string = "NEGATE";
35     private static readonly SUBTRACTION: string = "-";
36     private static readonly DIVIDE: string = "/";
37     private static readonly PLUS: string = "+";
38     private static readonly MULTIPLY: string = "*";
39     private static readonly MIN: string = "MIN";
40     private static readonly MAX: string = "MAX";
41     private static readonly AVERAGE: string = "AVERAGE";
42     private static readonly STDDEV: string = "STDDEV";
43     private static readonly FIRST: string = "FIRST";
44     private static readonly LAST: string = "LAST";
45     private static readonly NOT: string = "NOT";
46     private static readonly AND: string = "AND";
47     private static readonly OR: string = "OR";
48
49     private functions = new Map<string, CallFunction>();
50
51     public constructor( functionName:string, arrayArguments:CallArguments){
52         this.functionName = functionName;
53         this.arrayArguments = arrayArguments;
54         this.addFunctions();
55     }

```

```

72
73     private addFunctions(){
74         this.functions.set(CallValue.EQUALS, new Equals());
75         this.functions.set(CallValue.MULTIPLY, new Multiply());
76         this.functions.set(CallValue.DISTINCT, new Distinct());
77         this.functions.set(CallValue.LESS_THAN, new Lesser());
78         this.functions.set(CallValue.LESS_THAN_OR_EQUAL, new LesserOrEquals());
79         this.functions.set(CallValue.GREATER_THAN, new Greater());
80         this.functions.set(CallValue.GREATER_THAN_OR_EQUAL, new GreaterOrEquals());
81         this.functions.set(CallValue.NEGATE, new Negate());
82         this.functions.set(CallValue.DIVIDE, new Division());
83         this.functions.set(CallValue.SUBTRACTION, new Subtraction());
84         this.functions.set(CallValue.PLUS, new SumCall());
85         this.functions.set(CallValue.MIN, new Minimum());
86         this.functions.set(CallValue.MAX, new Maximum());
87         this.functions.set(CallValue.AVERAGE, new Average());
88         this.functions.set(CallValue.STDDEV, new StandardDeviation());
89         this.functions.set(CallValue.FIRST, new First());
90         this.functions.set(CallValue.LAST, new Last());
91         this.functions.set(CallValue.NOT, new Not());
92         this.functions.set(CallValue.AND, new And());
93         this.functions.set(CallValue.OR, new Or());
94     }

```

5) TDD (Test-driven development)

Fue un poco complejo hacer TDD al inicio de parseo debido a que directamente probamos métodos y no entidades (ej: leer el json y obtener un determinado objeto). Luego pudimos testear a **ZordParseo** que delega en SchemaFactory el parseo parcialmente.

Comenzar a testear de la clase más “**fácil/aislada**” a la más compleja fue una buena decisión. Por ejemplo testeamos los Value más simples como **Constant Value** o **VariableValue**.

Testear otra clase totalmente aislada como Condition que reciba una abstracción de Value fue clave para poder testear con la misma idea a otras entidades.