دانشگاه صنعتی بابل

Babol Noshirvani
University of Tech.

پایان‌نامه برای دریافت درجه کارشناسی در رشته مهندسی کامپیوتر
گرایش نرم‌افزار

عنوان

ربات معامله‌گر صرافی‌های رمز ارز دیجیتال

استاد راهنما

دکتر حمید جزایری

نگارنده

محمد دریانی

آبان ۱۴۰۰

سپاس گزاری

پروردگارا مرا یاری کن تا دانش اندکم نه نردبانی باشد برای فزونی و تکبر و غرور، نه حلقه‌ای برای اسارت و نه دست‌مایه‌ای برای تجارت، بلکه گامی باشد برای تجلیل از تو و تعالی ساختن زندگی خود و دیگران.

قبل از هر چیز، خداوند بزرگ را به خاطر لطفی که همواره شامل حال من نموده شاکرم. سپس، از زحمات استاد راهنمای محترم، جناب آقای دکتر حمید جزایری که نه تنها به عنوان استاد بلکه همچون همکاری در تمام مراحل انجام این تحقیق از رهنمودها و کمک‌های بیدریغ ایشان بهره‌مند شده‌ام، به ویژه به خاطر ساعت‌های طولانی که به بحث و تبادل نظر در مورد موضوع تحقیق بنده اختصاص داده‌اند که همواره برای من الهام‌بخش ایده و دیدگاهی تازه نسبت به موضوع بوده است، تشکر و قدردانی می‌کنم.

فهرست مطالب

فهرست شکل ها

چکیده

این پروژه راجع به معامله‌گر خودکار صرافی‌های رمزارز است که تعطیلی ندارند و شبانه روز فعالیت می‌کنند. یک بخش نوشتن استراتژی معامله است که از این پروژه خارج است و تنها سیگنال خرید و فروش آن به دست برنامه می‌رسد. یک بخش حساب صرافی و معامله کردن با آن است که از آن یک دسترسی با API داریم. حال این پروژه به عنوان واسطی بین استراتژی و صرافی عمل می‌کند تا سیگنال‌های صادر شده را در حساب‌های تعریف شده در پروژه اجرایی کند. این روند منجر به نظارت شبانه روزی بازار می‌شود. لحظه‌هایی که صفحه معاملاتی در دسترس ما نیست، می‌توانیم خیالمان راحت باشد که برنامه‌ای در حال اجرا است تا استراتژی ما را در لحظه اجرایی کند. ضمن این که این معاملات الگوریتمی با یک منطقی پیش می‌رود که در حالت عادی احتمالا اجرایی کردن آن از عهده انسان خارج است. امید است که بتوان با این برنامه سودهای خوبی را از این بازار کسب کرد.

کلمات کلیدی: رمز ارز، صرافی، سیگنال، کتابخانه، CCXT، کوکوین، kucoin، پایگاه داده، NodeJS، ExpressJS، ReactJS، MongoDB، Mongoose، API، سفارش استاپ، سفارش برداشت سود و توقف ضرر، سفارش پایه، TradingView، Webhook، localtunnel، PLA، سفارش limit/market

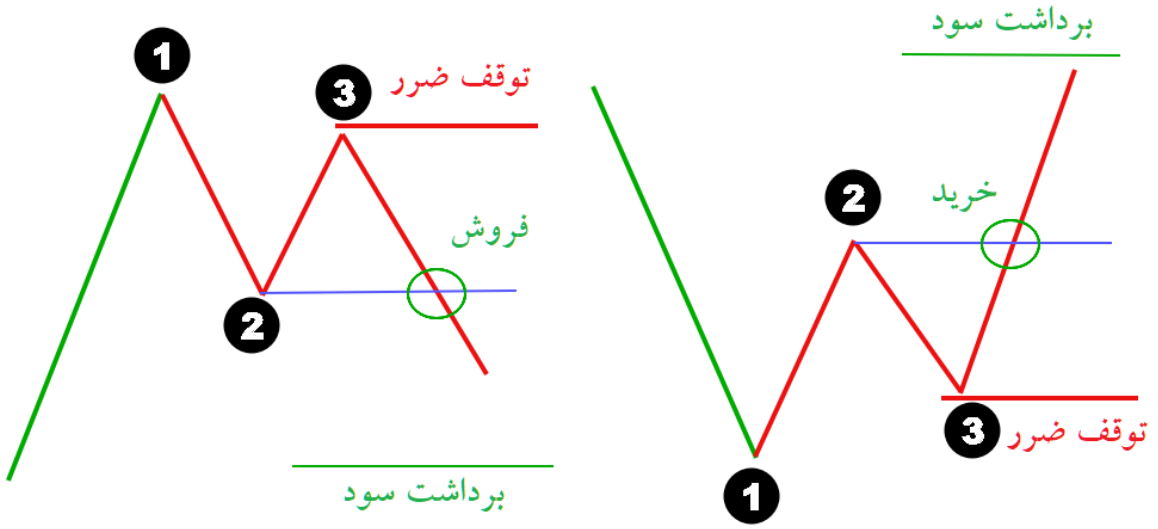
فصل اول

۱ مقدمه

با گذشت زمان کم کم همه‌ی ایده‌ها و ساخته‌های قدیمی بشر تحت تاثیر تکنولوژی قرار می‌گیرند و به شکلی نوین ظهور پیدا می‌کنند. نظام مالی هم از این قاعده مستثنا نیست و مدام به شکلی جدید برساخته می‌شود. در سال‌های اخیر شکلی جدید از ارز و اسناد ساخته شد، با این ایده که به صورت غیرمتمرکز کنترل شود که در ایران ما این ارزها را به نام ارز دیجیتال یا رمز ارز می‌شناسیم. هر چند این ایده دستخوش تغییراتی شد و با خود سیستم‌های همسان با نظام مالی قدیم را به همراه آورد. یکی از این سیستم‌ها صرافی‌های متعددی است که در سطح جهان برای بازار مالی رمز ارزها ساخته شد تا به عنوان واسطی برای به هم رساندن عرضه و تقاضا در این بازارها عمل کند. ویژگی متمایز این صرافی‌ها از بازارهای مالی سابق، باز بودن بازار در تمام روز و هفت روز هفته است. بنابراین دیگر به سختی می‌توان مثل قبل نیروی انسانی را پشت این بازار قرارداد تا در تمام ساعاتی که بازار باز است به استراتژی مورد نظر عمل کند. ضمن این که انسان دچار خطاهای روان شناختی می‌شود که شاید نتواند به راحتی به استراتژی تعیین شده عمل کند و بیشتر به طور هیجانی و غیرمنطقی معامله می‌کند. بی شک انسان در مقابل کامپیوتر با محدودیت‌هایی رو به رو است، از این رو خیلی از افراد و شرکت‌ها به این سمت رفتند که از کامپیوتر برای پیاده‌سازی استراتژی و عمل کردن به آن در این بازارهای مالی استفاده کنند. سایت‌های متعددی برای این کار وجود دارد اما کمتر سایتی دسترسی این را می‌دهد که استراتژی شخصی خودمان را پیاده‌سازی و با توجه به اطلاعات معاملات گذشته آزمایش کنیم. از طرفی کتابخانه‌های موجود در زبان‌های مختلف دسترسی اتصال به صرافی‌های مختلف را به راحتی می‌دهند اما به راحتی قابلیت پیاده‌سازی استراتژی شخصی را ندارند یعنی توابع مورد نیاز برای بررسی این بازارهای مالی به درستی تعریف نشده است که استراتژی پیاده‌سازی شود و با استفاده از اطلاعات معامله‌های گذشته، نرخ برد و دیگر گزارشات خروجی گرفته شود. در نتیجه در این پروژه با استفاده از پیوند این دو قسمت یعنی سایتی برای پیاده‌سازی و آزمایش استراتژی شخصی و دادن سیگنال، کتابخانه‌ای برای اتصال به صرافی برای معامله کردن، ربات معامله‌گری ساخته شد که بعد از دریافت سیگنال در هر دقیقه، برای هر کاربر متصل به صرافی، آن سیگنال را اجرا می‌کند.

۲ بررسی کلی روند پروژه

در حال حاضر این پروژه در فاز MVP قرار دارد و شاید هنوز به استراتژی بهینه‌ای نرسیده باشد، یا در صورت وجود کاربرها و سیگنال‌های زیاد به مشکل بخورد اما در هر حال تا این جای کار یک بخش برای اتصال به صرافی و معامله کردن داریم که به واسطه کتابخانه CCXT در NodeJS کنترل می‌شود، یک بخش برای پیاده‌سازی استراتژی داریم که به واسطه زبان Pine در سایت Trading View نوشته و آزمایش می‌شود. نهایت یک سیگنال از Webhook تعریف شده در هشدار متصل به استراتژی در سایت Trading View، هر دقیقه دریافت می‌شود که اطلاعات آن در پایگاه داده ساخته شده با MongoDB ذخیره می‌شود و در ادامه این سیگنال برای همه‌ی کاربران موجود در پایگاه داده عمل می‌کند. اما کار به این جا ختم نمی‌شود. در هر معامله‌ای که انجام می‌شود، چه خرید باشد چه فروش، باید قیمتی را جهت انجام معامله‌ای برای برداشت سود یا توقف ضرر در نظر گرفت. به شکل زیر:



۱ برداشت سود و توقف ضرر در خرید و فروش

در نتیجه ربات ما هم بدین شکل عمل خواهد کرد که تا زمان انجام یکی از این معاملات توقف ضرر یا برداشت سود، به صورت مداوم بررسی می‌کند که کدام یکی از این دو سفارش گذاشته شده معامله شده است، در این صورت سفارش متقابل را حذف کند. به طور مثال وقتی معامله فروش ارزی انجام شد، دو سفارش برای برداشت سود و توقف ضرر ثبت می‌شود که باید تنها یکی از این دو سفارش عمل کند و معامله انجام شود. به فرض این که این معامله منجر به سود شود، پس سفارش برداشت سود عمل می‌کند، حال ربات ما باید سفارش توقف ضرر را لغو کند تا در آینده به اشتباه عمل نکند.

در نهایت پس از تکمیل شدن ثبت این سری سفارش‌ها گزارشی از این سری سفارش‌ها در پایگاه داده ذخیره می‌شود که بعد از عمل کردن سفارش برداشت سود یا توقف ضرر این گزارش با محاسبه مقدار سود یا ضرر به روزرسانی می‌شود. به این شکل می‌توان پس از مدتی روشن گذاشتن ربات بررسی کرد که آیا این استراتژی که تعیین شده به طور میانگین سودده است یا خیر.

برای استفاده راحت‌تر از این ربات و دیدن گزارش معاملات هر کاربر و سیگنال‌ها رابطه کاربری ساده‌ای به کمک ReactJS ساخته شده است.

در ادامه به بررسی مسیر ساخت، ساختمان پایگاه‌داده، الگوریتم‌ها، کدها و تکنولوژی استفاده شده می‌پردازیم. امید است که در آینده این پروژه در سطح تجاری تکمیل شود و در دسترس عموم قرار بگیرد.

۳ طریقه اجرای پروژه

از آدرس ’github.com/7smd7/autotrader‘ پروژه را دانلود کنید.

در مرحله اول نیاز به نصب MongoDB و NodeJS دارد.

بعد باید با دستور ’npm install‘ کتابخانه‌های لازم را نصب کرد.

بعد با دستور ’npm start‘ فرانت‌اند اجرا می‌شود.

بعد با دستور ’node src/server/server.js‘ بک‌اند اجرا می‌شود.

فصل دوم

۱ بررسی ساختمان داده‌ها

ساختمان داده‌های این پروژه به سه بخش تقسیم می‌شود.

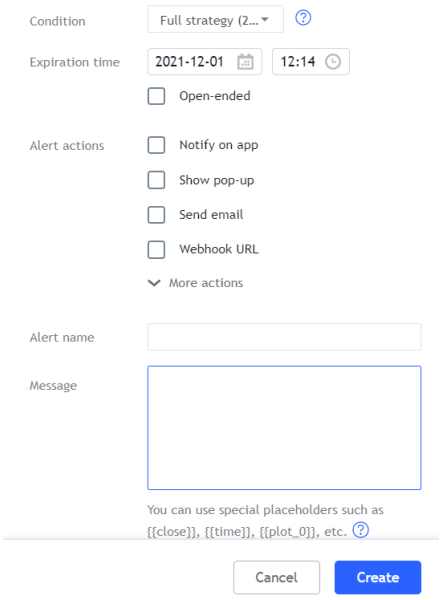
۱- ساختار سیگنال دریافتی از TradingView

۲- ساختار APIهای متصل به صرافی

۳- ساختار تاریخچه‌ی سری سفارش‌های هر API

در ادامه این ساختارها را به ترتیب بررسی می‌کنیم تا دید کلی نسبت به ساختار پروژه پیدا کنیم.

۱/۱ ساختار سیگنال دریافتی از TradingView



۲ تنظیم هشدار در TradingView

در این پروژه فرض شده است که استراتژی از سمت کاربری در TradingView نوشته می‌شود و کد ما کاری به آن استراتژی که به زبان Pine نوشته می‌شود ندارد. در واقع آن استراتژی باید سیگنال خرید یا فروش بدهد که آن سیگنال را میتوان در قسمت هشدار تعریف کرد که چگونه اطلاع رسانی شود. منطقی‌ترین روش برای برنامه‌نویسی استفاده از webhook است که نیاز به اکانت پرمیموم این سایت دارد. در این پروژه هم از این روش استفاده شده است. در قسمت Message هم ساختاری که به آدرس webhook باید ارسال شود تعریف می‌شود. ساختاری که در هر دقیقه برای این پروژه ارسال می‌شد به شکل زیر است.

```
{
    "name"   : "DPA 30m v1.0",
    "tiker"  : "TRX",
    "market" : "USDT",
    "signal" : "none",
    "limit"  : 0.1012466667,
    "SL"     : 0.6541393815,
    "TP"     : 1.0902323025,
}
```

که در اینجا در قسمت name، اسم استراتژی‌ای (DPA) که فعال است به همراه تایم فریم‌ای که روی آن پردازش می‌کند (30m) و نسخه استراتژی (v1.0) وجود دارد. در ادامه tiker (TRX) را داریم که نشان دهنده سمبل اختصاری رمزارز هست که توسط بازار پایه رمزارز market (USDT) معامله می‌شود. قسمت signal میتواند سه حالت (none / buy / sell) باشد که نشان دهنده این است که چه عملی را باید انجام دهیم. در واقع خیلی از مواقع ممکن است در یک دقیقه که سیگنال صادر می‌شود لازم نباشد که معامله‌ای انجام دهیم پس عملی که سیگنال به می‌گوید انجام دهیم none است. در قسمت بعد limit را می‌بینیم که قیمت سفارش لیمیت اصلی را مشخص می‌کند. در ادامه دو عدد می‌بینیم که درواقع به درصد نوشته شده است که به ترتیب SL مخفف Stop Loss به معنای درصدی که کاربر ضرر را متحمل می‌شود و بعد از آن ضرر خود را متوقف می‌کند، TP مخفف Take Profit به معنای درصدی که کاربر از سود خود راضی می‌شود و سود خود را برداشت می کند.

این اطلاعات که توسط webhook به کد ما می‌رسد در پایگاه داده به همراه دو پارامتر دیگر ذخیره می‌شود که یکی شناسه سیگنال است که به صورت خودکار ایجاد می‌شود و دیگری زمان است که مشخص می‌کند سیگنال چه زمانی به دست ما رسیده است. به شکل زیر می‌توان خلاصه‌ای از این ساختار را دید.


```
{
    "time"    : 1635670381536,
    "name"    : "DPA 30m v1.0",
    "tiker"   : "TRX",
    "market"  : "USDT",
    "signal"  : "none",
    "limit"   : 0.1012466667,
    "SL"      : 0.6541393815,
    "TP"      : 1.0902323025,
    "_id": "617e596d186609f22b554676",
    "__v": 0
}
```

این ساختار به صورت مدل در mongoose تعریف شده است که کد آن به شکل زیر می‌باشد.

```
const SignalSchema = new mongoose.Schema (
{
    time:{
        type: Number,
        required: true,
    },
    name:{
        type: String,
        required: true,
        trim: true,
    },
    tiker:{
        type: String,
        required: true,
        trim: true,
    },
    market:{
        type: String,
        required: true,
        trim: true,
    },
    signal: {
        type: String,
        required: true,
        trim: true,
        enum: ['buy', 'sell', 'none']
    },
    limit: {
        type: Number,
        required: true,
    },
    SL: {
        type: Number,
        required: true,
    },
    TP: {
        type: Number,
        required: true,
    },
});
```

۱/۲ ساختار APIهای متصل به صرافی

در این پروژه برای اتصال به صرافی از کتابخانه CCXT استفاده شده است. مزیت استفاده از این کتابخانه این است که به راحتی می‌شود کدی که اکنون (بنابه‌دلایلی که در ادامه اشاره می‌کنیم) برای صرافی Kucoin نوشته شده است را برای صرافی‌های دیگر تغییر داد و حتی این را میتوان به صورت متغیر تعریف کرد و نیاز به بازنویسی کل کد نیست.

اگر اطلاعات APIهای دریافتی از پایگاه داده را در config ذخیره کنیم، آن وقت تعریف صرافی در ccxt به شکل زیر می‌باشد.

```
const exchange = new ccxt.kucoin({
    'id' : config._id,
    'apiKey': config.apiKey,
    'secret' : config.secret,
    'password' : config.password,
    'timeout': 30000,
    'enableRateLimit': true,
})
```

تعریف این که ccxt به کدام صرافی متصل شود در اینجا به صورت ccxt.kucoin انجام شده است، اما برای انعطاف‌پذیری بیشتر کد و اتصال به صرافی‌های مختلف میتوان به شکل ccxt[“kucoin”] هم تعریف کرد که در این صورت اسم صرافی را هم می‌تواند از پایگاه داده بگیرد. اما به دلیل تحریم‌هایی که در ایران است و تغییرات جزئی که کد لازم داشت تا منعطف شود و کارمزد و محدودیت کمتری که کوکوین برای معامله داشت، فعلا صرافی ثابت کوکوین در نظر گرفته شده است.

برای هر صرافی که تعریف می‌شود یک شناسه در نظر گرفته شده است که تعریف آن اختیاری است و با شناسه‌ای که در پایگاه داده دارد مقدار دهی شده است.

سه مقدار 'apiKey'، 'secret' و 'password' برای اتصال به صرافی کوکوین الزامی است. در حالی که بعضی از صرافی‌ها 'password' را ندارند.

در ادامه 'timeout' به معنای مقدار زمانی که کتابخانه منتظر پاسخ صرافی بماند به میلی‌ثانیه است. که در اینجا تعریف شده است. مثلا در اینجا ۳۰ ثانیه در نظر گرفته شده است.

همیشه قوانینی برای استفاده پیاپی از API وجود دارد که اگر رعایت نشود امکان دارد سرور ما را مسدود کند و تا مدتی قادر به استفاده از آن API نباشیم. پس باید خودمان یک محدودیتی ایجاد کنیم که از آن قوانین پیرویت کند. کتابخانه CCXT در خود اطلاعات قوانین و محدودیت‌های همه‌ی صرافی‌هایی را که پشتیبانی می‌کند را دارد. برای این قضیه که فاصله‌ی زمانی بین دو درخواست API کوتاه‌تر از زمان

مورد نظر نشود، کافی است مقدار 'enableRateLimit' را برابر true قرار بدهیم که خود CCXT بین هر درخواست یک حداقل فاصله زمانی تعریف شده در صرافی را رعایت می‌کند.

کد تعریف مدل config در mongoose به شکل زیر می‌باشد که با خود تاریخچه‌ی سری سفارش‌ها را به همراه دارد که بررسی دقیق آن را به بخش بعد واگذار می‌کنیم.

```
const ConfigSchema = new mongoose.Schema (
{
    name: {
        type: String,
        required: true,
        trim: true
        },
    apiKey: {
        type: String,
        required: true,
        unique: true,
        trim: true
    },
    secret: {
        type: String,
        required: true,
        trim: true
    },
    password: {
        type: String,
        trim: true
    },
    allocation: {
        type: Number,
        required: true,
    },
    orderHistory : [{
        time:{
            type: Number,
            required: true,
        },
        status:{
            type: String,
            required: true,
            default: 'open',
            enum: ['open', 'closed', 'canceled', 'expired']
        },
        side: {
            type: String,
```

```
            required: true,
            trim: true,
            enum: ['buy', 'sell']
        },
        amount: {
            type: Number,
            required: true,
        },
        root: {
            type: String,
            required: true,
        },
        rootCost: {
            type: Number,
            required: true,
        },
        stopLoss: {
            type: String,
        },
        stopLossCost: {
            type: Number,
        },
        takeProfit: {
            type: String,
        },
        takeProfitCost: {
            type: Number,
        },
        PAL: {
            type: Number,
        },
        market:{
            type: String,
            required: true,
            trim: true,
        },
        tiker:{
            type: String,
            required: true,
            trim: true,
        },
        strategy:{
            type: String,
            required: true,
            trim: true,
        }
    }],});
```

این جا دو کلید جدید داریم. یکی name است که می‌تواند اسم صرافی باشد اما اکنون به عنوان اسمی برای نمایش و تشخیص این که برای کدام کاربر است استفاده می‌شود. دیگری allocation است که متغیری برای مدیریت سرمایه است به معنی این که ربات چند درصد از سرمایه را در معامله دخیل کند، که با توجه به ریسک هر کاربر میتوان این میزان را تغییر داد. فعلا چون هدف تست کد بود از این متغیر استفاده نشده است. اما کارکرد آن بدین صورت است که موقع سفارش گذاری، مقدار سفارش برابر با موجودی حساب ضربدر این متغیر می‌شود. استراتژی دیگر این است که موقع خروج از بازار کل مقدار رمزارز متغیر را به فروش برسانیم و به رمزارز ثابت تبدیل کنیم.

درادامه orderHistory را داریم که مربوط به تاریخچه سری سفارش‌های این کاربر می‌شود که در بخش بعد به بررسی این قسمت می‌پردازیم.

۱/۳ ساختار تاریخچه‌ی سری سفارش‌های هر API

با توجه به توضیحات بالا می‌توانیم مستقیم به بررسی پایگاه داده بپردازیم.

time: زمانی که این سری سفارش‌ها در پایگاه داده ثبت شدند.

status: وضعیت این سری سفارش‌ها که مشخص می‌کند سفارش های استاپ انجام شده‌اند یا خیر.

side: جهت معامله یعنی خرید یا فروش بودن سفارش اصلی را مشخص می‌کند.

amount: مقدار یا حجم سفارش اصلی را مشخص می‌کند.

root: شناسه سفارش اصلی را ذخیره می‌کند.

rootCost: هزینه‌ای که سفارش اصلی برداشته است را ذخیره می‌کند.

stopLoss: شناسه سفارش توقف ضرر را ذخیره می‌کند.

stopLossCost: هزینه‌ای که سفارش توقف ضرر برداشته است را ذخیره می‌کند.

takeProfit: شناسه سفارش برداشت سود را ذخیره می‌کند.

takeProfitCost: هزینه‌ای که سفارش برداشت سود برداشته است را ذخیره می‌کند.

PAL: به معنای Profit & Loss میزان سود یا ضرر در این سری سفارش را ذخیره مشخص می‌کند.

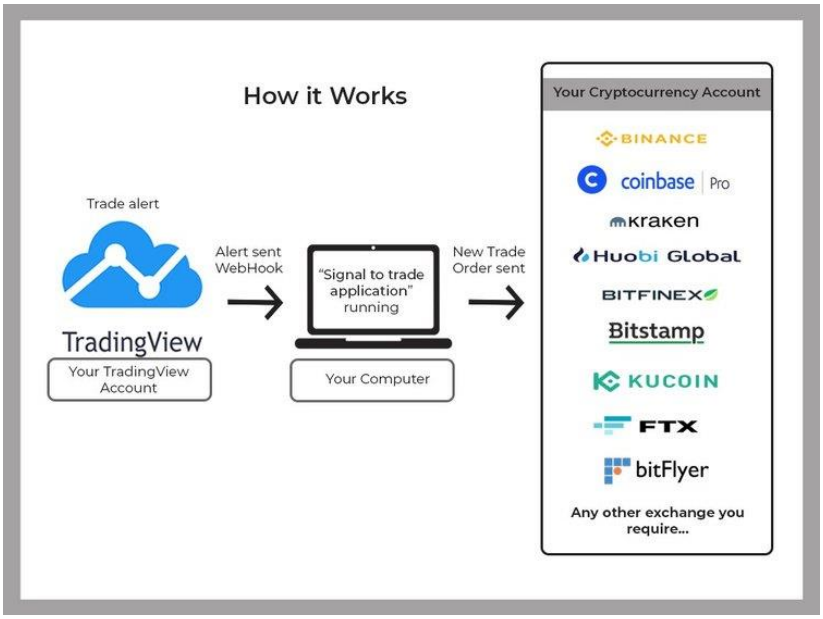
market: سمبل اختصاری رمزارز بازار پایه

tiker: سمبل اختصاری رمزارز بازار هدف

strategy: اسم استراتژی که استفاده شده (همان که در بخش سیگنال name یاد شد)

۲ شیوه عملکرد

همان طور که در بخش‌های قبل به چگونگی کارکرد این ربات اشاره شد، کل فرآیند معامله خودکار از سه بخش استراتژی برای دادن سیگنال، نرم‌افزاری جهت عمل کردن به سیگنال و نهایت خود صرافی که این فرآیند را تکمیل می‌کند. به شکل زیر می‌توان این فرآیند را خلاصه کرد.



۳ دیاگرام شیوه کار برنامه

کاری که مشخصا این پروژه انجام می‌دهد دقیقا همین نرم‌افزاری هست که واسط بین TradingView و صرافی‌ها است.

```
min of binance
TRX/BNB         amount: 0.1     price: 1e-7
TRX/BTC         amount: 0.1     price: 1e-8
TRX/BUSD        amount: 0.1     price: 0.00001
TRX/USDT        amount: 0.1     price: 0.00001

 min of coinex
TRX/BCH         amount: 50 ,    price: 1e-8
TRX/BTC         amount: 50 ,    price: 1e-10
TRX/ETH         amount: 50 ,    price: 1e-8
TRX/USDC        amount: 50 ,    price: 0.000001
TRX/USDT        amount: 50 ,    price: 0.000001
VET/BCH         amount: 10 ,    price: 1e-10
VET/BTC         amount: 10 ,    price: 1e-10
VET/ETH         amount: 10 ,    price: 1e-8
VET/USDC        amount: 10 ,    price: 0.000001
VET/USDT        amount: 10 ,    price: 0.000001

 min of kucoin
TRX/BTC         amount: 1 ,     price: 1e-9
TRX/ETH         amount: 1 ,     price: 1e-8
TRX/KCS         amount: 1 ,     price: 0.000001
TRX/USDT        amount: 1 ,     price: 0.000001
VET/BTC         amount: 10 ,    price: 1e-8
VET/ETH         amount: 10 ,    price: 1e-8
VET/KCS         amount: 1 ,     price: 0.000001
VET/USDT        amount: 10 ,    price: 1e-7
```

در این تصویر می‌بینیم که حداقل حجمی (amount) که در هر بازار صرافی‌های مختلف با حداقل قیمت (price) یا حداکثر دقت قیمت گذاری می‌توان سفارش‌گذاری کرد چقدر است. قبل از شروع پروژه باید انتخاب می‌شد که از چه صرافی و چه رمزارزی استفاده شود که کمترین هزینه را برای هر معامله داشته باشد. با توجه به این که رمزارز TRX و VET قیمت کم و کارمز کمی دارد، در بین صرافی‌ها هم باگذر از بایننس که بزرگترین صرافی است اما مثل اغلب صرافی‌ها ایران را تحریم کرده است، کوکوین و کوینکس انتخاب شدند که با توجه به مقادیر رو به رو استفاده از بازار

TRX/USDT در صرافی کوکوین منطقی به نظر رسید. البته لازم به ذکر است که پروژه با همه‌ی رمزارزها کار می‌کند اما فعلا هشداری که در TradingView تنظیم شده، برای این که هزینه تست کردن کم باشد بر روی این مقادیر تنظیم شده است.

۳ توابع مورد نیاز

بعد از این که روند پروژه مشخص شد، یک سری توابع برای پیاده سازی تعیین شد.

۳/۱ دریافت سیگنال معاملاتی به صورت webhook

استفاده از webhook نیاز به یک آدرس دارد که از سمت TradingView به آن سیگنال بفرستد و از سمت نرم‌افزار سیگنال دریافت شود. برای این که این آدرس در سطح وب ساخته شود و از همه جا دسترسی ایجاد شود، از کتابخانه localtunnel استفاده شده است که آدرس localhost:3000/ را به آدرس daryani.loca.lt تونل می‌کند.

۳/۲ انجام اعمال پایه خرید و فروش سفارشی (Buy / Sell Limit)

در کتابخانه CCXT برای انجام این اعمال توابع مشخصی وجود دارد اما برای این که کد تکراری کم شود، از یک تابع سفارش گذاری استفاده شده است که هم برای خرید و هم برای فروش استفاده می‌شود. بعد از این که سفارش ثبت شد باید روند منتظر انجام معامله این سفارش بماند. تا زمانی که معامله انجام نشده نباید به مرحله بعد یعنی ثبت سفارش برداشت سود و توقف ضرر برود.

۳/۳ انجام خرید و فروش مبتنی بر حد ضرر و حد سود (Buy / Sell Stop Limit)

بعد از انجام معامله پایه باید دو سفارش ثبت شود. یکی برای برداشت سود و دیگری برای توقف ضرر. اسم این دو سفارش stop limit می‌باشد.

۳/۴ گرفتن موجودی

این تابع با گرفتن سمبل اختصاری مورد نظر، موجودی رمزارز کاربر را اعلام می‌کند.

۳/۵ توقف سفارش استاپ متقابل

پس از انجام یکی از سفارش‌های استاپ باید سفارش متقابل آن توسط این تابع لغو شود.

فصل سوم

۱ بررسی کد سمت سرور

برای پیاده سازی سمت سرور این پروژه از ExpressJS و Mongoose استفاده شد که با NodeJS و MongoDB اجرا می‌شوند.

به طور کلی این بخش را می‌توان به پنج بخش تقسیم کرد.

۱- مدل‌های پایگاه داده
۲- تنظیمات پایگاه داده
۳- کنترل کننده درخواست‌ها
۴- تنظیمات سرور
۵- الگوریتم ربات

در ادامه به ترتیب به این بخش‌ها می‌پردازیم.

۱/۱ مدل‌های پایگاه داده

در فصل دوم به صورت مفصل و دقیق به این قسمت پرداخته شد و کدهای این قسمت بررسی شد.

۱/۲ تنظیمات پایگاه داده

File: db\configSeeds.js

```
const Config = require('../models/model-config')
const seedData = require('./configSeeds.json')

const configSeeds = () => {Config.deleteMany({})
  .then(() => {
    return Config.insertMany(seedData)
 })
 .then(console.log("seedData added"))
 .catch(console.error)
 .finally(() => {
  // process.exit()
 })}

module.exports = configSeeds
```

File: db\connection.js

```
const mongoose = require('mongoose');
const url = 'mongodb://localhost/AutoTrader'
const configSeeds = require('./configSeeds.js')

mongoose.connect(url, {useNewUrlParser: true})
const con = mongoose.connection


con.on('open',function(){
    console.log("Database connected...")
    // configSeeds()
})
module.exports = mongoose
```

File: db/configSeeds.json

```
[
    {
        "name": "M.Daryani",
        "apiKey": "*****",
        "secret": "*****",
        "password": "*****",
        "allocation": 0.05,
        "orderHistory": []
      }
   ]
```

وظیفه فایل configSeeds.js این است که هر دفعه که پایگاه داده شروع به کار می‌کند اطلاعات سابق مدل Config را پاک می‌کند و اطلاعات موجود در configSeeds.json وارد آن می‌کند.

وظیفه فایل connection.js این است هم راه‌اندازی سرور محلی پایگاه داده روی آدرس 'mongodb://localhost/AutoTrader' است. بعد هم در صورت نیاز می‌توان از configSeeds برای وارد کردن داده اولیه استفاده کرد.

۱/۳ کنترل کننده درخواست‌ها

File: controllers/controller-signal.js

```
const express = require('express');
const router = express.Router();
const Signal = require('../models/model-signal')

router.get('/', async(req,res) => {
  try{
      const signals = await Signal.find()
      res.json(signals)
  }catch(err){
      res.send('Error ' + err)
  }
})

router.get('/:id', async(req,res) => {
  try{
         const signal = await Signal.findById(req.params.id)
         res.json(signal)
  }catch(err){
      res.send('Error ' + err)
  }
})


router.post('/', async(req,res) => {
  const signal = new Signal(req.body)
  try{
      const a1 =  await signal.save()
      res.json(a1)
  }catch(err){
      res.send(err)
  }
})

module.exports = router
```

این فایل درخواست‌هایی که به سیگنال فرستاده می‌شود را کنترل می‌کند. بدین صورت که اگر

۱- هیچ پارامتری نداشته باشد و روش get باشد، همه‌ی سیگنال‌های موجود را برمی‌گرداند.

۲- شناسه‌ی سیگنالی را با روش get بفرستد، سیگنالی که این شناسه را داشت را برمی‌گرداند.

۳- در body مشخصات کامل یک سیگنال موجود باشد و به روش post بفرستد، یک سیگنال با مشخصات موجود در body را در پایگاه داده ذخیره می‌کند.

File: controllers/controller-config.js

```
const express = require('express');
const router = express.Router();
const Config = require('../models/model-config')

router.get('/', async(req,res) => {
  try{
      const configs = await Config.find()
      res.json(configs)
  }catch(err){
      res.send('Error ' + err)
  }
})

router.get('/:id', async(req,res) => {
  try{
         const config = await Config.findById(req.params.id)
         res.json(config)
  }catch(err){
      res.send('Error ' + err)
  }
})


router.post('/', async(req,res) => {
  const config = new Config(req.body)

  try{
      const a1 =  await config.save()
      res.json(a1)
  }catch(err){
      res.send('Error')
  }
})

router.patch('/:id/orders',async(req,res)=> {
  try{
      const config = await Config.findById(req.params.id)
      config.orderHistory.push(req.body)
      const a1 = await config.save()
      res.json(a1)
  }catch(err){
      res.send(err)
  }
})
```

```
router.patch('/:id/orders/:id2',async(req,res)=> {
  try{
      const config = await Config.findById(req.params.id)
      for (let i =0 ; i< config.orderHistory.length;i++){
        if (config.orderHistory[i].root == req.params.id2){
          config.orderHistory[i] = req.body
        }
      }
      const a1 = await config.save()
      res.json(a1)
  }catch(err){
      res.send(err)
  }
})

module.exports = router
```

این فایل درخواست‌هایی که به سیگنال فرستاده می‌شود را کنترل می‌کند. بدین صورت که اگر

۱- بدون پارامتر و به روش get درخواست کند، همه‌ی configها برگردانده می‌شوند.

۲- با شناسه و به روش get درخواست کند، config با آن شناسه برگردانده می‌شود.

۳- در body مشخصات کامل یک config موجود باشد و به روش post بفرستد، یک config با مشخصات موجود در body را در پایگاه داده ذخیره می‌کند.

۴- در body مشخصات کامل یک سری سفارش موجود باشد و به روش patch به نشانی `'/:id/orders'` بفرستد، یک سری سفارش با مشخصات موجود در body را در پایگاه داده ذخیره می‌کند.

۵- به نشانی `'/:id/orders/:id2'` با روش patch، در body مشخصات یک سری سفارش را بفرستد، سری سفارشی که در body است را با سری سفارش به شناسه id2 که برای config به شناسه id است جایگزین می‌کند.

۱/۴ تنظیمات سرور

File: server/server.js

```
const express = require('express');
const connection = require('../db/connection');

const app = express();
app.use(express.json())

const configController = require('../controllers/controller-
config.js');
```

```
app.use('/configs', configController)

const signalController = require('../controllers/controller-
signal.js');
app.use('/signals', signalController)

const robot = require('./robot.js');
app.use('/', robot)

app.listen(3001,function(){
    console.log('Server started')
})

module.exports = app;
```

در این جا به پیاده سازی ExpressJS می‌پردازیم. بدین صورت که بعد تعریف ExpressJS، آدرس کنترلرها را تعریف می‌کنیم. برای دسترسی به جدول config و فعال کردن کنترلر آن، آدرس '/configs' را برایش تعریف می‌کنیم. همچنین برای سیگنال هم آدرس '/signals'. برای این که کد ربات فعال شود، آدرس '/' را به آن نسبت می‌دهیم.

۱/۵ الگوریتم ربات

در این بخش فایل server/robot.js را به صورت قطعه قطعه بررسی می‌کنیم.

اول به کتابخانه‌هایی که برای پیاده‌سازی الگوریتم استفاده شده است می‌پردازیم.

```
const ccxt = require('ccxt');                          اتصال به صرافی
const express = require("express");                    راه‌اندازی سرور
const localtunnel = require('localtunnel');    تونل کردن شبکه داخلی به اینترنت
const router = express.Router();                       دسترسی به درخواست‌ها
const axios = require('axios');                        ارسال درخواست
```

برای برقراری با پایگاه داده یک متغیر آدرس پایه را در نظر می‌گیریم تا کد منعطف شود.

```
base_url = "http://localhost:3001"
```

برای این که راحت یک سری سفارش را ثبت کنیم از تابع زیر استفاده می‌کنیم.

```
function Order(strategy ,side, tiker, market, amount, root, rootCost,
               stopLoss, takeProfit){
    this.time = (new Date()).getTime()
    this.PAL = - rootCost;
    this.status = "open"
    this.side = side;
```

```
    this.tiker = tiker;
    this.market = market;
    this.amount = amount;
    this.root = root;
    this.rootCost = rootCost;
    this.stopLoss = stopLoss;
    this.stopLossCost = 0;
    this.takeProfit = takeProfit;
    this.takeProfitCost = 0;
    this.strategy = strategy;
}
```

در js تابع مستقیمی برای انتظار یا sleep تعریف نشده. پس ما به این شکل تعریف می‌کنیم.

```
function sleep(ms) {
    return new Promise(resolve => setTimeout(resolve, ms));
  }
```

برای این که همه‌ی configها را در ccxt به عنوان صرافی تعریف کنیم.

```
exchanges = []

axios.get(base_url + "/configs")
.then((response) => {
    response.data.forEach(config =>{
        const exchange = new ccxt.kucoin({
            'id' : config._id,
            'apiKey': config.apiKey,
            'secret' : config.secret,
            'password' : config.password,
            'timeout': 30000,
            'enableRateLimit': true,
        })
        exchanges.push(exchange)
    })
}).catch(function (error) {
    console.log(error);
})
```

برای این که موجودی یک رمزارز را چک کنیم از تابع زیر استفاده می‌کنیم.

```
const fetchBalance = async (currency, exchange) =>{
    const balances = await exchange.fetchBalance();
    const balance = balances.free[currency];
    return balance;
}
```

حال به بخش ثبت سفارش می‌رویم.

```
const createOrder = async function (exchange,config){
    const {id, strategy, allocation, takeProfit, stopLoss, base, quot
e, side, price} = config;
    const market = `${base}/${quote}`;
    let balance = await fetchBalance(base,exchange);
    let baseBalance = balance ? balance : 0 ;
    balance = await fetchBalance(quote, exchange);
    let quoteBalance = balance ? balance : 0 ;

    // const amount = (side == 'buy') ? (allocation * quoteBalance)/p
rice: baseBalance; //origin
    // const amount = (side == 'buy') ? (allocation * quoteBalance)/p
rice: baseBalance*allocation ; //test
    amount =1;

    const rootOrder = await exchange.createOrder(market, 'limit', sid
e, amount, price);
    console.log(id + " root order " + rootOrder.id);

    valid = true;
    while (valid){
        const order = await exchange.fetchOrder(rootOrder.id);
        if (order.status != 'open')
            valid = false;
    }

    const order = await exchange.fetchOrder(rootOrder.id);
    if (order.status != 'closed')
        return

    const invertedSide = (side == 'buy') ? 'sell' : 'buy';

    const stopLossParams = {
        stop : (side == 'buy') ? 'loss' : 'entry' ,
        stopPrice: (side == 'buy') ? price*(1-
stopLoss) : price*(1+stopLoss),
    }
    const stopLossOrder = await exchange.createOrder(market, 'market'
, invertedSide, amount, null, stopLossParams);
    console.log(id + " stop loss order " + stopLossOrder.id);

    const takeProfitParams = {
        stop : (side == 'buy') ? 'entry' : 'loss',
        stopPrice: (side == 'buy') ? price*(1+takeProfit) : price*(1-
takeProfit),
```

```
    }
    const takeProfitOrder = await exchange.createOrder(market,
      'market', invertedSide, amount, null, takeProfitParams);
    console.log(id + " take profit order " + takeProfitOrder.id);


    const orders = new Order(strategy, side, base, quote, amount,
       rootOrder.id, order.cost, stopLossOrder.id, takeProfitOrder.id)

    axios.patch(base_url + "/configs/"+id+"/orders",orders)
    .then((response) => {
        console.log( id + " submit new set orders.")
    }).catch(function (error) {
        console.log(error);
      })
}
```

در این تابع اول متغیرهای config دریافت می‌شود و بعد به پیاده‌سازی استراتژی مدیریت سرمایه می‌رود. برای این که مشخص شود چه مقداری باید است معامله شود، نیاز است مشخص شود موجودی چقدر است تا با توجه به مقدار allocation، درصدی از آن موجودی را در معامله دخیل کنیم.

این جا سه استراتژی در نظر گرفته شده.

۱- موقع خرید تنها درصدی از سرمایه را رمزارز مورد نظر می‌خریم. موقع فروش تمام رمزارز را به رمزارزی ثابت تبدیل می‌کنیم.

۲- هم در خرید و هم در فروش تنها درصدی از موجودی را دخیل می‌کنیم.

۳- برای تست ربات مقدار را مستقیما برابر یک در نظر می‌گیریم.

در مرحله بعد با تابع createOrder سفارش پایه ثبت می‌شود. بعد باید روند آن قدر در حلقه منتظر بماند تا سفارش ثبت شده، معامله و تکمیل شود. در غیر این صورت روند متوقف می‌شود.

بعد از این که سفارش پایه تکمیل شد، به ترتیب سفارش توقف ضرر و برداشت سود ثبت می‌شود. برای انجام این عملیات از همان تابع createOrder استفاده می‌شود، با این تفاوت که باید پارامترهایی را برایش بفرستیم. یک پارامتر stop است که دو حالت دارد، یکی loss که اگر قیمت از عددی کمتر شد سفارش عمل می‌کند، دیگر هم entry که اگر بیشتر از عددی شد عمل می‌کند. با توجه به این که جهت سفارش خرید یا فروش بوده این دو حالت را تنظیم می‌کنیم. پارامتر دیگر هم عدد قیمت مورد نظر برای برداشت سود یا توقف ضرر است. در مستندات ccxt اشاره شده که این قسمت به صورت یکتا نوشته نشده و ممکن است برای هر صرافی کمی این کد متفاوت باشد. برای عمومی سازی ربات باید این قسمت را در نظر گرفت که شاید دچار مشکل شود.

دیگر موضوعی که باید در مورد این سری سفارش دقت کنیم این است که سفارش پایه با روش limit انجام می‌شود، یعنی انقدر می‌ماند تا کسی همگام با قیمت ما سفارش ثبت کند، اما در سفارش‌های استاپ، روش market است تا بعد که شرط فعال شد، در هر حالتی معامله را انجام دهد.

در انتها هم این سری سفارش را در پایگاه داده ثبت می‌کند.

در کد زیر به دریافت سیگنال از webhook می‌پردازیم. فرض بر این است که آدرس پایه ما، آدرس webhook است. در واقع آدرس پایه ما به آدرس webhook تونل شده است.

```
router.use("",express.text())

router.all("", (req, res) => {
    const body = req.body.replace("[","").replace("]","");
    query = JSON.parse(body);
    query.time = (new Date()).getTime();
    query.signal = query.signal.toLowerCase();

    axios.post(base_url + "/signals",query)
    .then((response) => {
        console.log(response.data)
    }).catch(function (error) {
        console.log(error);
      })
    config={}
    if (query.signal != 'none'){
        exchanges.forEach(exchange =>{
            config.strategy = query.name;
            config.side = query.signal;
            config.price = query.limit;
            config.base = query.tiker;
            config.quote = query.market;
            config.takeProfit = query.TP/100;
            config.stopLoss = query.SL/100;
            config.id = exchange.id;
            createOrder(exchange,config);
        })
    }
});
```

اول یک سری اصلاحاتی برای استاندارد شدن داده انجام شده است. بعد اگر سیگنال none نبوده و برابر buy یا sell بوده، ثبت سفارش انجام شده است.

```
(async () => {
    const tunnel = await localtunnel({ port: 3001,
                                       subdomain: "daryani"});
    console.log(tunnel.url);
    // tunnel.close();
})();
```

در این جا هم پورت ۳۰۰۱ شبکه محلی به دامین loca.lt با سابدامین daryani تونل شده است.

```
module.exports = router;
checkStopOrders();
```

در انتها هم که روتر به عنوان ماژول خروجی این فایل انتخاب شده است و بعد تابع بررسی سفارش‌های استاپ صدا زده شده است که در ادامه به این تابع می‌پردازیم.

شاید تنها بخش چالشی کد مربوط به این تابع می‌شد که هزینه‌ی زیادی دارد تا همه‌ی سری سفارش‌های باز بررسی شوند. در نسخه‌های قبلی کدی که برای این قسمت استفاده می‌شد به شکل زیر بود که هر دفعه با توجه به شناسه‌ای که از سفارش‌های استاپ داشت، گزارشی از سفارش‌های استاپ سری سفارش‌هایی که هنوز سفارش‌های استاپشان عمل نکرده بود را درخواست می‌کرد تا ببیند اکنون آیا یکی از دو سفارش برداشت سود یا توقف ضرر عمل کرده است یا خیر. اگر عمل کرده بود دیگری را لغو و اطلاعات کنونی را ذخیره می‌کرد. هزینه‌ی این تعداد درخواست زیاد بود. گاهی هم مانع از ارسال درخواست‌های دیگر می‌شد. این شد که بعد الگوریتم این تابع به صورت دیگر با کمی ریسک بیشتر نوشته شد. فعلا به الگوریتم اولیه‌ی این تابع می‌پردازیم. (در آن نسخه از برنامه هنوز پایگاه داده پیاده سازی نشده بود و برای نوشتن گزارش از فایل استفاده می‌شد.)

```
const checkStopOrders = async (configs)=>{
  var fs = require('fs');
  while(true){
    try {
      for (let i = 0; i < configs.length; i++){
        await sleep(5000);
        // await configs[i].exchangeCCXT.load_markets (true);
        for (let j = 0; j < configs[i].orderHistory.length; j++){
          if (configs[i].orderHistory[j].status == "open"){
            const stopLoss = await configs[i].exchangeCCXT.fetchOrder(configs[i].orderHistory[j].stopLoss);
            const takeProfit = await configs[i].exchangeCCXT.fetchOrder(configs[i].orderHistory[j].takeProfit);
            console.log(i,j, stopLoss.status, takeProfit.status);
            if (stopLoss.status == "closed"){
              const cancel = await configs[i].exchangeCCXT.cancelOrder(takeProfit.id);
              console.log(cancel);
              configs[i].orderHistory[j].status="closed";
```

```
            configs[i].orderHistory[j].PAL = -Math.abs(configs[i].orderHistory[j].PAL + stopLoss.cost);
            configs[i].orderHistory[j].stopLossCost = stopLoss.cost;
            console.log(configs[i].orderHistory[j]);
            fs.appendFile('log'.concat(i,'.txt'), configs[i].orderHistory[j].toString(), function (err) {
                if (err) throw err;
                console.log('Saved!');
              });
          }
          else if (takeProfit.status == "closed"){
            const cancel = await configs[i].exchangeCCXT.cancelOrder(stopLoss.id);
            console.log(cancel);
            configs[i].orderHistory[j].status="closed";
            configs[i].orderHistory[j].PAL = Math.abs(configs[i].orderHistory[j].PAL + takeProfit.cost);
            configs[i].orderHistory[j].takeProfitCost = takeProfit.cost;
            console.log(configs[i].orderHistory[j]);
            fs.appendFile('log'.concat(i,'.txt'), configs[i].orderHistory[j].toString(), function (err) {
                if (err) throw err;
                console.log('Saved!');
              });
          }
        }
      }
    }
    catch(err) {
      console.log(err);
    }
  }
}
```

در نسخه‌ی بعدی این تابع به شکل زیر پیاده‌سازی شد.

```
const checkStopOrders = async ()=>{
    while(true){
        await sleep(5000)
        axios.get(base_url + "/configs")
        .then((response) => {
            response.data.forEach(config => {
                const exchange = new ccxt.kucoin({
                    'id' : config._id,
                    'apiKey': config.apiKey,
                    'secret' : config.secret,
                    'password' : config.password,
                    'timeout': 30000,
                    'enableRateLimit': true,
                })
```

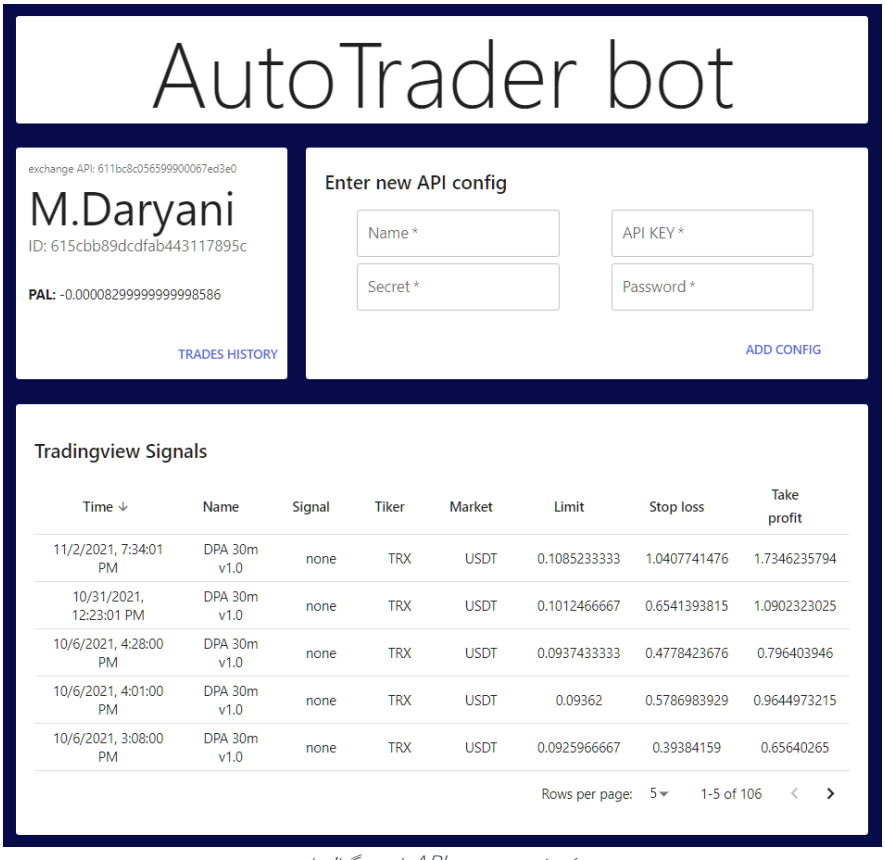
```
                exchange.fetchClosedOrders().then(closedOrders =>{
                    let cancelid = []
                    for (let i of closedOrders){
                        cancelid.push(i.id)
                    }
                    config.orderHistory.forEach(orders =>{
                        if (orders.status == "open"){
                            if (cancelid.includes(orders.stopLoss)){
                                exchange.cancelOrder(orders.takeProfit).then(cancel => {
                                    orders.status="closed";
                                    exchange.fetchOrder(orders.stopLoss).then(stopLoss =>{
                                        orders.PAL = -Math.abs(orders.PAL + stopLoss.cost);
                                        orders.stopLossCost = stopLoss.cost;
                                        axios.patch(base_url + "/configs/"+config._id+"/orders/"+orders.root,orders)
                                        .then((response) => {
                                            console.log( orders.stopLoss + " cancelled.")
                                        }).catch(function (error) {
                                            console.log(error);
                                          })
                                    })
                                });
                            }
                            else if (cancelid.includes(orders.takeProfit)){
                                exchange.cancelOrder(orders.stopLoss).then( cancel =>{
                                    orders.status="closed";
                                    exchange.fetchOrder(orders.takeProfit).then(takeProfit => {
                                        orders.PAL = Math.abs(orders.PAL + takeProfit.cost);
                                        orders.takeProfitCost = takeProfit.cost;
                                        axios.patch(base_url + "/configs/"+config._id+"/orders/"+orders.root,orders)
                                        .then((response) => {
                                            console.log( orders.takeProfit + " cancelled.")
                                        }).catch(function (error) {
                                            console.log(error);
                                          })
                                    });
                                });
                            }

                        }
                    })
                })
            })
        }).catch(function (error) {
            console.log(error);
        })
    }
}
```

در این الگوریتم اول ۵۰ سفارش بسته شده‌ی اخیر را درخواست می‌کند بعد بررسی می‌کند که آیا در بین این ۵۰ معامله انجام شده، شناسه سفارش‌های استاپ وجود دارد یا خیر که اگر وجود دارد سفارش متقابل را لغو می‌کند. کار منطقی‌تر این بود که از بین سفارش‌های باز بررسی شود اما متاسفانه سفارش‌های استاپ در پاسخ درخواست لیست سفارش‌های باز ثبت نمی‌شد. نهایت با این روش کمی بهینه‌سازی شد اما اگر ۵۰ سفارش تکمیل شود و هنوز سفارش استاپ بازی داشته باشیم، روند به مشکل می‌خورد و به روایتی آن سفارش‌ها یتیم می‌شوند. شاید بتوان در آینده در صورت اتفاق افتادن همچین حالتی هر دو سفارش استاپ را لغو کرد.

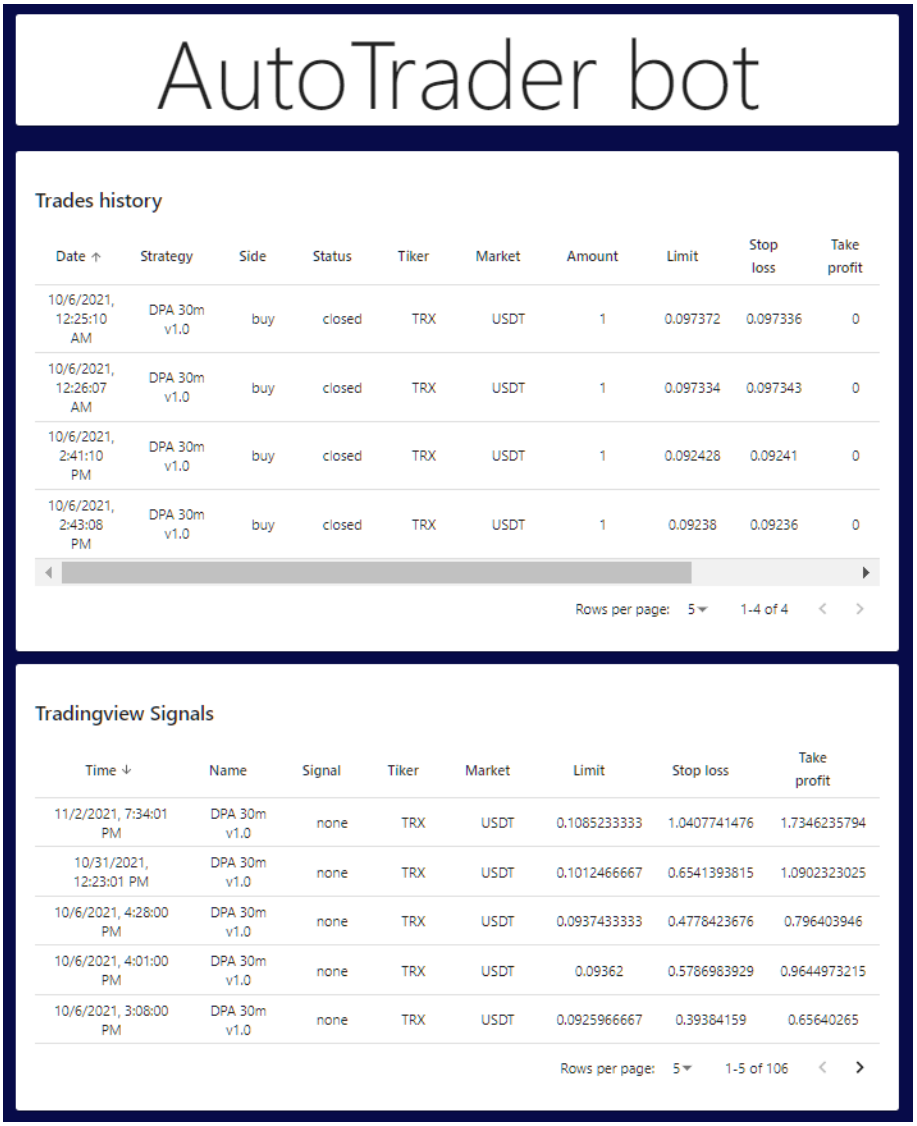
این بود الگوریتم ربات. در ادامه به بخش ظاهر نرم‌افزار می‌پردازیم.

۲ رابطه کاربری

برای پیاده‌سازی یک رابطه کاربری که بتوان سیگنال‌ها و معاملات هر API را دید و بررسی کرد که PLA مجموع معاملات چقدر شده است، از تکنولوژی ReactJS استفاده شد. یک صفحه اولیه برای نشان دادن APIها و یک صفحه هم برای نشان دادن سری معاملات آن‌ها در نظر گرفته شد، به طوری که زیر هر دو صفحه آخرین سیگنال‌ها را نمایش بدهد. این دو صفحه به شکل زیر طراحی شد.



٤ صفحه مدیریت APIها و سیگنال‌ها



۵ صفحه معاملات اخیر یک API

برای پیاده‌سازی این تم رابطه‌کاربری از کتابخانه MaterialUI کمک گرفته شد.

با توجه به تصاویر چهار کامپوننت و دو کانتینر داشتیم. (./src/views)

۱- دو کانتینر هم یکی کلیت نرم‌افزار بود و دیگری شامل‌شونده این کامپوننت‌ها (./App.js , ./Dashboard.js)

۲- سیگنال‌های اخیر (./Signals.js)

۳- APIها به همراه مجموع PLA آن (./ConfigCard.js)

۴- ثبت یک API جدید (./InputConfigCard.js)

۵- معاملات اخیر یک API (./Trades.js)

در ادامه به بررسی کد تک تک این‌ها می‌پردازیم.

۲/۱ کانتینر

کانتینر در کلمه یعنی شامل شونده که در واقع کامپوننت‌ها را شامل می‌شود.

```
import * as React from 'react';
import Container from '@mui/material/Container';
import Box from '@mui/material/Box';
import Dashboard from './Dashboard';
import {
  BrowserRouter as Router,
  Switch,
  Route,
  Redirect,
} from "react-router-dom";


export default function App() {
  return (
    <Container maxWidth="lg">
      <Box sx={{ my: 4 }}>
        <Router>
          <Switch>
            <Route exact path="/">
              <Dashboard/>
            </Route>
            <Route path="/">
              <Redirect to="/" />;
            </Route>
          </Switch>
        </Router>
      </Box>
    </Container>
  );
}
```

بخش مهمی که در این کد وجود دارد سوییچ و روتر است که هر آدرسی که وارد کنیم را به آدرس پایه هدایت می‌کند و این باعث می‌شود که به صفحه اصلی برنامه هدایت شود. در آدرس پایه هم کامپوننت داشبورد صدا زده شده است.

در ادامه به بررسی داشبورد می‌پردازیم.

```
import React from "react";
import Container from '@mui/material/Container';
import Typography from "@mui/material/Typography";
import Signals from "./Signals";
import Trades from "./Trades";
import InputConfigCard from "./InputConfigCard";
import ConfigCard from "./ConfigCard";
import Grid from '@mui/material/Grid';
import Paper from '@mui/material/Paper';
import axios from 'axios';
import {
  BrowserRouter as Router,
  Switch,
  Route,
  Link,
  useRouteMatch,
  useParams,
} from "react-router-dom";

export default function Dashboard() {
  const [configs, setConfigs] = React.useState([]);
  let match = useRouteMatch();

  React.useEffect(() => {
    const fetchData = async () => {
      axios.get("/configs")
      .then((response) => {
          setConfigs(response.data)
      }).catch(function (error) {
          console.log(error);
        })
    };
    fetchData();
  }, [config]);


  return (
    <Container sx={{ width: "100%" }}>
      <Paper sx= {{my:4}}>
        <Grid container justifyContent="center">
          <Typography
            variant="h1"
            id="title"
            component="div"
          >
            AutoTrader bot
          </Typography>
```

```
        </Grid>
      </Paper>

      <Switch>
        <Route path={`${match.path}:configId`}>
          <Trades/>
        </Route>
        <Route exact path={`${match.path}`}>
          <Grid container sx= {{my:4}} spacing={{ xs: 2, md: 3 }}
                                columns={{ xs: 4, sm: 8, md: 12 }}>
            {
              configs.map((config, index) =>{
                return(
                  <Grid item xs={2} sm={4} md={4} key={index}>
                    <ConfigCard config = {config}/>
                  </Grid>
                )
              })
            }
            <Grid item xs={4} sm={8} md={8}>
              <InputConfigCard/>
            </Grid>
          </Grid>
        </Route>
      </Switch>
      <Signals/>
    </Container>
  );
}
```

با استفاده از useEffect متغیر config را مدام با پایگاه داده به روزرسانی می‌کنیم.

در قسمت خروجی تابع طراحی را مشاهده می‌کنیم. اول یک هدر برای برنامه می‌نویسیم که اسم برنامه را نشان دهد. بعد با توجه به این که در چه آدرسی هستیم، تصمیم می‌گیرد که معاملات یا APIهای مختلف و وارد کردن API جدید را نشان دهد. match کمک می‌کند که پارامتری از آدرس را به عنوان متغیر دریافت کند تا بعد بتوانیم معاملات یک API با شناسه خاص را نشان دهیم.

۲/۲ سیگنال‌های اخیر

```
import React from "react";
import PropTypes from "prop-types";
import Box from "@mui/material/Box";
import Table from "@mui/material/Table";
import TableBody from "@mui/material/TableBody";
import TableCell from "@mui/material/TableCell";
import TableContainer from "@mui/material/TableContainer";
import TableHead from "@mui/material/TableHead";
import TablePagination from "@mui/material/TablePagination";
import TableRow from "@mui/material/TableRow";
import TableSortLabel from "@mui/material/TableSortLabel";
import Typography from "@mui/material/Typography";
import Paper from "@mui/material/Paper";
import { visuallyHidden } from "@mui/utils";
import axios from "axios";

function descendingComparator(a, b, orderBy) {
  if (b[orderBy] < a[orderBy]) {
    return -1;
  }
  if (b[orderBy] > a[orderBy]) {
    return 1;
  }
  return 0;
}

function getComparator(order, orderBy) {
  return order === "desc"
    ? (a, b) => descendingComparator(a, b, orderBy)
    : (a, b) => -descendingComparator(a, b, orderBy);
}

// This method is created for cross-browser compatibility, if you don't
// need to support IE11, you can use Array.prototype.sort() directly
function stableSort(array, comparator) {
  const stabilizedThis = array.map((el, index) => [el, index]);
  stabilizedThis.sort((a, b) => {
    const order = comparator(a[0], b[0]);
    if (order !== 0) {
      return order;
    }
    return a[1] - b[1];
  });
  return stabilizedThis.map((el) => el[0]);
}
```

```
const headCells = [
  {
    id: "time",
    numeric: false,
    disablePadding: true,
    label: "Time",
  },
  {
    id: "name",
    numeric: false,
    disablePadding: false,
    label: "Name",
  },
  {
    name: "signal",
    numeric: false,
    disablePadding: false,
    label: "Signal",
  },
  {
    id: "tiker",
    numeric: false,
    disablePadding: false,
    label: "Tiker",
  },
  {
    id: "market",
    numeric: false,
    disablePadding: false,
    label: "Market",
  },
  {
    id: "limit",
    numeric: true,
    disablePadding: false,
    label: "Limit",
  },
  {
    id: "stopLoss",
    numeric: true,
    disablePadding: false,
    label: "Stop loss",
  },
  {
    id: "takeprofit",
    numeric: true,
```

```
    disablePadding: false,
    label: "Take profit",
  },
];


function EnhancedTableHead(props) {
  const { order, orderBy, onRequestSort } = props;
  const createSortHandler = (property) => (event) => {
    onRequestSort(event, property);
  };

  return (
    <TableHead>
      <TableRow>
        {headCells.map((headCell) => (
          <TableCell
            key={headCell.id}
            align="center"
            padding={headCell.disablePadding ? "none" : "normal"}
            sortDirection={orderBy === headCell.id ? order : false}
          >
            <TableSortLabel
              active={orderBy === headCell.id}
              direction={orderBy === headCell.id ? order : "asc"}
              onClick={createSortHandler(headCell.id)}
            >
              {headCell.label}
              {orderBy === headCell.id ? (
                <Box component="span" sx={visuallyHidden}>
                  {order === "desc" ? "sorted descending" : "sorted ascending"}
                </Box>
              ) : null}
            </TableSortLabel>
          </TableCell>
        ))}
      </TableRow>
    </TableHead>
  );
}

EnhancedTableHead.propTypes = {
  numSelected: PropTypes.number.isRequired,
  onRequestSort: PropTypes.func.isRequired,
  onSelectAllClick: PropTypes.func.isRequired,
  order: PropTypes.oneOf(["asc", "desc"]).isRequired,
  orderBy: PropTypes.string.isRequired,
```

```
  rowCount: PropTypes.number.isRequired,
};




export default function Signals() {
  const [order, setOrder] = React.useState("desc");
  const [orderBy, setOrderBy] = React.useState("time");
  const [rows, setRows] = React.useState([]);
  const [page, setPage] = React.useState(0);
  const [dense, setDense] = React.useState(true);
  const [rowsPerPage, setRowsPerPage] = React.useState(5);

  React.useEffect(() => {
    const fetchData = async () => {
      axios.get("/signals/")
      .then((response) => {
        setRows(response.data);
      }).catch(function (error) {
          console.log(error);
        })
    };
    fetchData();
  }, [rows]);


  const handleRequestSort = (event, property) => {
    const isAsc = orderBy === property && order === "asc";
    setOrder(isAsc ? "desc" : "asc");
    setOrderBy(property);
  };

  const handleChangePage = (event, newPage) => {
    setPage(newPage);
  };

  const handleChangeRowsPerPage = (event) => {
    setRowsPerPage(parseInt(event.target.value, 10));
    setPage(0);
  };

  // Avoid a layout jump when reaching the last page with empty rows.
  const emptyRows =
    page > 0 ? Math.max(0, (1 + page) * rowsPerPage - rows.length) : 0;

  return (
```

```
<Box sx={{ width: "100%" }}>
      <Paper sx={{ width: "100%", mb: 2 , p: 3}}>
      <Typography
        sx={{ flex: "1 1 100%", my: 2}}
        variant="h6"
        id="tableTitle"
        component="div"
      >
        Tradingview Signals
      </Typography>
      <TableContainer>
        <Table
          sx={{ minWidth: 750 }}
          aria-labelledby="tableTitle"
          size={dense ? "small" : "medium"}
        >
          <EnhancedTableHead
            order={order}
            orderBy={orderBy}
            onRequestSort={handleRequestSort}
          />
          <TableBody>
            {stableSort(rows, getComparator(order, orderBy))
              .slice(page * rowsPerPage, page * rowsPerPage + rowsPerPage)
              .map((row, index) => {
                return (
                  <TableRow
                    hover
                    tabIndex={-1}
                    key={index}
                  >
                    <TableCell align="center">
                         {new Date(row.time).toLocaleString()}
                    </TableCell>
                    <TableCell align="center">{row.name}</TableCell>
                    <TableCell align="center">{row.signal}</TableCell>
                    <TableCell align="center">{row.tiker}</TableCell>
                    <TableCell align="center">{row.market}</TableCell>
                    <TableCell align="center">{row.limit}</TableCell>
                    <TableCell align="center">{row.SL}</TableCell>
                    <TableCell align="center">{row.TP}</TableCell>
                  </TableRow>
                );
              })}
            {emptyRows > 0 && (
              <TableRow
                style={{
```

```
                    height: (dense ? 33 : 53) * emptyRows,
                  }}
                >
                  <TableCell colSpan={6} />
                </TableRow>
              )}
            </TableBody>
          </Table>
        </TableContainer>
        <TablePagination
          rowsPerPageOptions={[5, 10, 25]}
          component="div"
          count={rows.length}
          rowsPerPage={rowsPerPage}
          page={page}
          onPageChange={handleChangePage}
          onRowsPerPageChange={handleChangeRowsPerPage}
        />
      </Paper>
    </Box>
  );
}
```

ساخت جدول در materialUI یک نمونه‌ای دارد تا منعطف باشد و راحت بتوان ستون دیگری اضافه کرد و تنها چند سطر نمایش داده شود و یا جدول بر اساس ستونی مرتب شود. از این نمونه برای سیگنال‌های اخیر و معاملات اخیر استفاده شده است. تقریبا تنها لازم است که هدر جدول را درست وارد کنیم، بعد مقادیر را در تابع useEffect از پایگاه داده بگیریم و به روزرسانی کنیم. سپس این مقادیر را در جدول به درستی وارد کنیم.

۲٫۳ APIها به همراه مجموع PLA آن

```
import * as React from 'react';
import Card from '@mui/material/Card';
import CardActions from '@mui/material/CardActions';
import CardContent from '@mui/material/CardContent';
import Button from '@mui/material/Button';
import Typography from '@mui/material/Typography';
import { Grid } from '@mui/material';
export default function ConfigCard(props) {
  const [config, setConfig] = React.useState(props.config);
  React.useEffect(() => {
    const fetchData = async () => {
      let PAL = 0;
```

```
      config.orderHistory.forEach(order=>{
        PAL += order.PAL;
      })
      setConfig({...config, PAL:PAL})
    };
    fetchData();
  }, [config]);
  return (
    <Card sx={{ minWidth: 275 , height: 300 }}>
      {console.log(config)}
      <CardContent>
        <Typography sx={{ fontSize: 14 }}
                    color="text.secondary" gutterBottom>
          exchange API: {config.apiKey}
        </Typography>
        <Typography variant="h3" component="div">
          {config.name}
        </Typography>
        <Typography sx={{ mb: 1.5 }} color="text.secondary">
          {"ID: " + config._id}
        </Typography>
        <Typography variant="body2">
          <br/>
          <b>{'PAL: '}</b> {config.PAL}
          <br/><br/>
        </Typography>
      </CardContent>
      <CardActions>
        <Grid
          container
          justifyContent="flex-end"
          alignItems="flex-end"
        >
          <Button size="small" href = {"/"+config._id} >
                Trades history
           </Button>
        </Grid>
      </CardActions>
    </Card>
  );
}
```

این جا مشخصات config یا API نمایش داده می‌شود. در تابع useEffect هم مجموع PAL سری سفارش‌ها بدست می‌آید.

۲/۴ ثبت یک API جدید

```
import * as React from 'react';
import Box from '@mui/material/Box';
import Card from '@mui/material/Card';
import CardActions from '@mui/material/CardActions';
import CardContent from '@mui/material/CardContent';
import Button from '@mui/material/Button';
import Typography from '@mui/material/Typography';
import TextField from '@mui/material/TextField';
import Grid from '@mui/material/Grid';

export default function InputConfigCard() {
  const [state, setState] = React.useState({
    name: "",
    apiKey: "",
    secret: "",
    password: "",
    allocation: 0.05,
    orderhistory: []
  })
  function handleChange(evt) {
    const value = evt.target.value;
    setState({
      ...state,
      [evt.target.name]: value
    });
  }
  function handleClick(evt) {
    axios.post("/configs/",state)
    .then((response) => {
        console.log( response.data_id + " submit new config.")
    }).catch(function (error) {
        console.log(error);
      })
  }
  return (
    <Card sx={{ minWidth: 275 ,height: 300}}>
      <CardContent       sx={{'& > :not(style)': { m: 1}, width : "95%"}}>
        <Typography  variant="h6" component="div">
            {"Enter new API config"}
        </Typography>
        <Grid container  spacing={{ xs: 1, md: 1 }} >
          <Grid container item direction="row" alignItems="center"
                  justifyContent="space-around" >
            <TextField
                required
```

```
              id="outlined-required"
              label="Name"
              name="name"
              value={state.name}
              onChange={handleChange}
            />
          <TextField
              required
              id="outlined-required"
              label="API KEY"
              name="apiKey"
              value={state.apiKey}
              onChange={handleChange}
            />
        </Grid>
        <Grid container item direction="row" alignItems="center"
              justifyContent="space-around"  >
          <TextField
            required
            id="outlined-required"
            label="Secret"
            name="secret"
            value={state.secret}
            onChange={handleChange}
          />
          <TextField
            required
            id="outlined-required"
            label="Password"
            name="password"
            value={state.password}
            onChange={handleChange}
          />
        </Grid>
      </Grid>
    </CardContent>
    <CardActions sx = {{ mr : 6}}>
      <Grid container justifyContent="flex-end">
        <Button size="small" onClick = {handleClick}>Add config</Button>
      </Grid>
    </CardActions>
  </Card>
 );
}
```

اطلاعات لازم گرفته شده و نهایت با فشردن کلیک در پایگاه داده ذخیره می‌شود.

۲/۵ معاملات اخیر یک API

```
import React from "react";
import PropTypes from "prop-types";
import Box from "@mui/material/Box";
import Table from "@mui/material/Table";
import TableBody from "@mui/material/TableBody";
import TableCell from "@mui/material/TableCell";
import TableContainer from "@mui/material/TableContainer";
import TableHead from "@mui/material/TableHead";
import TablePagination from "@mui/material/TablePagination";
import TableRow from "@mui/material/TableRow";
import TableSortLabel from "@mui/material/TableSortLabel";
import Typography from "@mui/material/Typography";
import Paper from "@mui/material/Paper";
import { visuallyHidden } from "@mui/utils";
import axios from 'axios';
import {
  useParams
} from "react-router-dom";

function descendingComparator(a, b, orderBy) {
  if (b[orderBy] < a[orderBy]) {
    return -1;
  }
  if (b[orderBy] > a[orderBy]) {
    return 1;
  }
  return 0;
}

function getComparator(order, orderBy) {
  return order === "desc"
    ? (a, b) => descendingComparator(a, b, orderBy)
    : (a, b) => -descendingComparator(a, b, orderBy);
}

// This method is created for cross-browser compatibility, if you don't
// need to support IE11, you can use Array.prototype.sort() directly
function stableSort(array, comparator) {
  const stabilizedThis = array.map((el, index) => [el, index]);
  stabilizedThis.sort((a, b) => {
    const order = comparator(a[0], b[0]);
    if (order !== 0) {
      return order;
    }
    return a[1] - b[1];
```

```
  });
  return stabilizedThis.map((el) => el[0]);
}

const headCells = [
  {
    id: "date",
    numeric: false,
    disablePadding: true,
    label: "Date",
  },
  {
    id: "strategy",
    numeric: false,
    disablePadding: false,
    label: "Strategy",
  },
  {
    name: "side",
    numeric: false,
    disablePadding: false,
    label: "Side",
  },
  {
    id: "status",
    numeric: false,
    disablePadding: false,
    label: "Status",
  },
  {
    id: "tiker",
    numeric: false,
    disablePadding: false,
    label: "Tiker",
  },
  {
    id: "market",
    numeric: false,
    disablePadding: false,
    label: "Market",
  },
  {
    id: "amount",
    numeric: true,
    disablePadding: false,
    label: "Amount",
  },
```

```
  {
    id: "limit",
    numeric: true,
    disablePadding: false,
    label: "Limit",
  },
  {
    id: "stopLoss",
    numeric: true,
    disablePadding: false,
    label: "Stop loss",
  },
  {
    id: "takeprofit",
    numeric: true,
    disablePadding: false,
    label: "Take profit",
  },
  {
    id: "PAL",
    numeric: true,
    disablePadding: false,
    label: "PAL",
  },
];

function EnhancedTableHead(props) {
  const { order, orderBy, onRequestSort } = props;
  const createSortHandler = (property) => (event) => {
    onRequestSort(event, property);
  };

  return (
    <TableHead>
      <TableRow>
        {headCells.map((headCell) => (
          <TableCell
            key={headCell.id}
            align="center"
            padding={headCell.disablePadding ? "none" : "normal"}
            sortDirection={orderBy === headCell.id ? order : false}
          >
            <TableSortLabel
              active={orderBy === headCell.id}
              direction={orderBy === headCell.id ? order : "asc"}
              onClick={createSortHandler(headCell.id)}
            >
```

```
              {headCell.label}
              {orderBy === headCell.id ? (
                <Box component="span" sx={visuallyHidden}>
                  {order === "desc" ? "sorted descending" : "sorted ascending"}
                </Box>
              ) : null}
            </TableSortLabel>
          </TableCell>
        ))}
      </TableRow>
    </TableHead>
  );
}

EnhancedTableHead.propTypes = {
  numSelected: PropTypes.number.isRequired,
  onRequestSort: PropTypes.func.isRequired,
  onSelectAllClick: PropTypes.func.isRequired,
  order: PropTypes.oneOf(["asc", "desc"]).isRequired,
  orderBy: PropTypes.string.isRequired,
  rowCount: PropTypes.number.isRequired,
};

export default function Trades() {
  let { configId } = useParams();
  const [order, setOrder] = React.useState("asc");
  const [orderBy, setOrderBy] = React.useState("date");
  const [rows, setRows] = React.useState([]);
  const [page, setPage] = React.useState(0);
  const [dense, setDense] = React.useState(true);
  const [rowsPerPage, setRowsPerPage] = React.useState(5);

  React.useEffect(() => {
    const fetchData = async () => {
      console.log(configId)
      axios.get("/configs/"+configId)
      .then((response) => {
          setRows(response.data.orderHistory)
      }).catch(function (error) {
          console.log(error);
        })
    };
    fetchData();
  }, [rows]);
  const handleRequestSort = (event, property) => {
    const isAsc = orderBy === property && order === "asc";
    setOrder(isAsc ? "desc" : "asc");
```

```
  setOrderBy(property);
};
const handleChangePage = (event, newPage) => {
  setPage(newPage);
};

const handleChangeRowsPerPage = (event) => {
  setRowsPerPage(parseInt(event.target.value, 10));
  setPage(0);
};
// Avoid a layout jump when reaching the last page with empty rows.
const emptyRows =
  page > 0 ? Math.max(0, (1 + page) * rowsPerPage - rows.length) : 0;

return (
  <Box sx={{ width: "100%" }}>
        <Paper sx={{ width: "100%", mb: 2 , p: 3}}>
        <Typography
          sx={{ flex: "1 1 100%", my: 2}}
          variant="h6"
          id="tableTitle"
          component="div"
        >
          Trades history
        </Typography>
        <TableContainer>
          <Table
            sx={{ minWidth: 750 }}
            aria-labelledby="tableTitle"
            size={dense ? "small" : "medium"}
          >
            <EnhancedTableHead
              order={order}
              orderBy={orderBy}
              onRequestSort={handleRequestSort}
            />
            <TableBody>
              {stableSort(rows, getComparator(order, orderBy))
                .slice(page * rowsPerPage, page * rowsPerPage + rowsPerPage)
                .map((row, index) => {
                  return (
                    <TableRow
                      hover
                      tabIndex={-1}
                      key={index}
                    >
                      <TableCell align="center">
```

```
                        {new Date(row.time).toLocaleString()}
                      </TableCell>
                      <TableCell align="center">{row.strategy}</TableCell>
                      <TableCell align="center">{row.side}</TableCell>
                      <TableCell align="center">{row.status}</TableCell>
                      <TableCell align="center">{row.tiker}</TableCell>
                      <TableCell align="center">{row.market}</TableCell>
                      <TableCell align="center">{row.amount}</TableCell>
                      <TableCell align="center">{row.rootCost}</TableCell>
                      <TableCell align="center">{row.stopLossCost}</TableCell>
                      <TableCell align="center">{row.takeProfitCost}</TableCell>
                      <TableCell align="center">{row.PAL}</TableCell>
                    </TableRow>
                  );
                })}
              {emptyRows > 0 && (
                <TableRow
                  style={{
                    height: (dense ? 33 : 53) * emptyRows,
                  }}
                >
                  <TableCell colSpan={6} />
                </TableRow>
              )}
            </TableBody>
          </Table>
        </TableContainer>
        <TablePagination
          rowsPerPageOptions={[5, 10, 25]}
          component="div"
          count={rows.length}
          rowsPerPage={rowsPerPage}
          page={page}
          onPageChange={handleChangePage}
          onRowsPerPageChange={handleChangeRowsPerPage}
        />
      </Paper>
    </Box>
  );
}
```

مثل جدول سیگنال‌ها از نمونه کد جدول استفاده شده و برای معامله‌ها شخصی سازی شده است.

فصل چهارم

۱ بررسی برنامه

صبر و دید بلند مدت در سرمایه‌گذاری و بازارهای مالی معمولا منجر به سود می‌شود اما گاهی به دلیل طمع یا دید کوتاه مدت و نداشتن صبر تا دیدن نتیجه به دنبال تلاش برای سود بیشتر در کوتاه مدت هستیم. شاید بهترین کار این باشد که یک استراتژی روندها را تشخیص دهد و در روند صعودی وارد بازار شود و در روند نزولی خروج کند. تشخیص این روند از عهده انسان خارج است و به هر باید از ابزارهایی که کامپیوتر ارائه می‌کند استفاده کند. اگر این فرآیند تشخیص و ورود و خروج از بازار سرمایه به صورت خودکار انجام شود، کلی وقت، تمرکز و انرژی انسان ذخیره می‌شود. ضمن این که بر خلاف باور عموم مردم، انسان اصلا موجود منطقی‌ای نیست و غالبا بر اساس هیجان تصمیم می‌گیرد. این است که برای برداشت سود و توقف ضرر نیاز به یک سیستم منطقی است تا دچار ترس و طمع نشود. این جا کماکان کامپیوتر می‌تواند به کمک انسان بیاید. برنامه‌ای هم که این جا ارائه شد حداقل کار ممکن از این فرآیند را به صورت کامل انجام می‌دهد. یعنی به خوبی می‌توان با زبان pine از اندیکاتورهای مختلف استفاده کرد تا روندها را در بلند مدت تشخیص داد و به موقع در شروع روند نزولی بازار سیگنال فروش و در شروع روند صعودی سیگنال خرید را به نرم‌افزار صادر کرد و نرم‌افزار هم با گرفتن این سیگنال، در تمام حساب‌هایی که برایش تعریف شده، به سیگنال عمل می‌کند و سفارش‌گذاری می‌کند و تا لحظه برداشت سود یا توقف ضرر بر آن معامله نظارت می‌کند. قطعا معایب و مزایایی دارد که باید به آن‌ها پرداخته شود تا به نرم‌افزاری قابل اعتماد تبدیل شود. در ادامه به بررسی این معایب و مزایا می‌پردازیم.

۲ معایب و مزایا

۲/۱ مزایا

۱- دقیق بودن

۲- منطقی عمل کردن

۳- مدام درحال حساب و بررسی استراتژی و دادن سیگنال بودن

۴- امکان اتصال به اکثر صرافی‌ها

۵- امکان معامله خودکار با همه‌ی رمزارزهایی که صرافی پشتیبانی می‌کند

۶- امکان برداشت سود و توقف ضرر خودکار با تعیین درصد ریسک توسط استراتژی

۷- امکان اتصال به چند حساب هم زمان
۸- اتصال به سایت قدرتمند TradingView برای پیاده‌سازی استراتژی
۹- هزینه کم
۱۰- ثبت گزارش معاملات و سیگنال‌ها در پایگاه داده
۱۱- امکان توسعه برای همیشه روشن بودن
۱۲- لغو خودکار سفارش متقابل برداشت سود و توقف ضرر
۱۳- ادامه دادن فرآیند نظارت و بررسی مورد قبل حتی پس از شروع مجدد برنامه
۱۴- داشتن رابطه کاربری ساده و جدوالی گویا
۱۵- سعی به رعایت معماری MVC

۲/۲ معایب

۱- در حد MVP بودن
۲- دیپلوی نشده
۳- امنیت پایین
۴- راه حلی برای حفظ امنیت APIها اندیشیده نشده
۵- وابسته به localtunnel بودن که هر ٦ ساعت احتمالا دچار مشکل شود
۶- محدودیت و هزینه‌ی زیاد نظارت بر سفارش‌های توقف ضرر و برداشت سود
۷- نیازمند خرید اشتراک TradingView
۸- وابستگی به کتابخانه CCXT
۹- عدم امکان انتخاب استراتژی دلخواه
۱۰- نیاز به یک سیستم که دائما روشن باشد و برنامه را اجرا کند

۳ نتیجه‌گیری و جمع بندی

ایده‌ی خوبی است و خیلی به کسانی که با TradingView استراتژی می‌نویسند در معاملات شخصی کمک می‌کند تا استراتژی‌هایشان را به صورت خودکار پیاده‌سازی کنند و ببینند در واقعیت آیا سودده است یا زیان‌ده. امکان توسعه و تجاری‌سازی برنامه وجود دارد، تنها باید بر روی نقاط ضعف برنامه متمرکز شد و آن مسئله‌ها را حل کرد. نهایت اگر به تکامل برسد می‌توان یک سیستم را برای سرمایه‌گذازی و مدیریت سرمایه به این برنامه سپرد و خود به باقی ابعاد زندگی رسید.

۴ پیشنهادها

باید به نقاط ضعف این برنامه بیشتر فکر کرد و سعی به حل این مسائل کرد تا در ابعاد بزرگ دچار مشکل نشود و بتوان استفاده کرد.

بتوان از بین استراتژی‌های مختلف که به آدرس مخصوص به خودشان سیگنال می‌فرستند، استراتژی دلخواه را انتخاب کرد که پیاده‌سازی آن به سادگی انتخاب آدرس webhook هشدار آن استراتژی است.

به روش‌های دیگر سرمایه‌گذاری در رمزارز پرداخت مثل سپرده گذاری، معاملات آتی، وام‌دهی، بازارسازی و آربیتراژ صرافی‌های مختلف.

توابع برداشت سود و توقف ضرر را شاید بتوان بهینه‌تر و دقیق‌تر پیاده‌سازی کرد مثلا با نظارت خودکار قیمت و ثبت سفارش پایه در موقع برقراری شرط، در واقع بدون ثبت سفارش استاپ در صرافی.

اتصال به صرافی‌های ایران برای خرید به موقع USDT.

گسترش داد و این الگوریتم را به کارگزاری‌های بازارهای مالی دیگر مثل فارکس برد و بررسی کرد.

منابع

[1] "Exchanges — ccxt 1.60.18 documentation," [Online]. Available: https://ccxt.readthedocs.io/en/latest/manual.html.

[2] "KuCoin API Documentation," [Online]. Available: https://docs.kucoin.com/.

[3] "MUI Documentation," [Online]. Available: https://mui.com/getting–started/.

[4] "localtunnel github," [Online]. Available: https://github.com/localtunnel/localtunnel.

[5] "Mongoose ODM," [Online]. Available: https://mongoosejs.com.

[6] "TradingView," [Online]. Available: https://www.tradingview.com/.

[7] "Pine language quickstart," [Online]. Available: https://www.tradingview.com/pine–script–docs/en/v4/Quickstart_guide.html.