Instituto Federal do Triângulo Mineiro
Análise e Desenvolvimento de Sistemas

Redes de Computadores
Documentação do Trabalho Prático

Patrocínio - MG
2025

ALEXSSANDER JOSÉ DE OLIVEIRA DE CASTRO

LUCAS AMARAL LUCIANO

PABLO VINÍCIUS LIMA SOUZA

Redes de Computadores
Documentação do Trabalho Prático

Envio de documento em PDF com o objetivo de documentar um Sistema com aplicação de API da Binance no curso de Análise e Desenvolvimento de Sistemas

Prof: Júnio.

Patrocínio – MG
2025

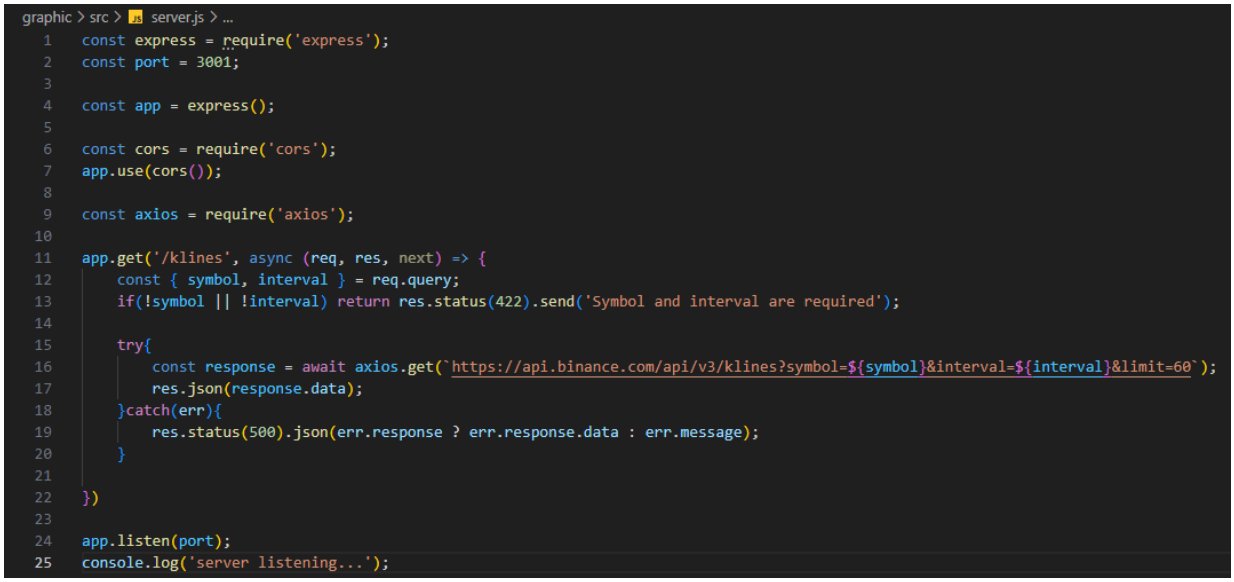
1.Introdução.

Neste trabalho, você encontrará a documentação relacionada ao desenvolvimento de um site voltado para a exibição de informações sobre criptomoedas, utilizando dados fornecidos pela API oficial da Binance. O objetivo principal deste projeto foi praticar o consumo de uma API pública, integrando-a a uma aplicação web capaz de apresentar valores e gráficos atualizados de diferentes ativos do mercado de criptoativos. Através do uso da API da Binance, foi possível acessar dados em tempo real, interpretar respostas no formato JSON e transformá-las em visualizações claras e dinâmicas para o usuário final. Este documento tem como função registrar as principais etapas do processo de desenvolvimento, desde a integração com a API até a estruturação da interface do site

2.Criação do servidor Backend.

Criar um servidor backend é importante quando usamos uma API como a da Binance porque ajuda a manter o sistema mais seguro e organizado. Com o backend, dá pra esconder informações sensíveis, como chaves de acesso, que não deveriam ficar visíveis no navegador. Além disso, ele serve pra filtrar e preparar os dados antes de mandar pro site, e também ajuda a controlar a quantidade de requisições feitas, evitando problemas com limites da API. Outro ponto importante é que, em muitos casos, o próprio navegador bloqueia requisições feitas diretamente do frontend por questões de segurança, como as regras de CORS. O backend resolve esse problema servindo como intermediário autorizado entre o navegador e a API.

2.1 - Implementação do Server.js (SERVIDOR BACKEND):



Esse código cria um servidor backend usando Node.js com o framework Express, com o objetivo de buscar dados da API pública da Binance e repassá-los para o frontend. Ele roda na porta 3001 e permite que o navegador acesse seus dados ativando o CORS. Ao acessar a rota /klines, o servidor espera receber dois parâmetros pela URL: symbol (símbolo da criptomoeda) e interval (intervalo de tempo dos candles). Se esses parâmetros não forem informados, o servidor responde com erro. Caso os parâmetros estejam corretos, ele usa a biblioteca Axios para fazer uma requisição à API da Binance e busca os últimos 60 candles da criptomoeda informada. Os dados retornados são então enviados diretamente como resposta para o frontend. Se houver algum erro na requisição, o servidor responde com uma mensagem de erro e o status 500. Esse backend funciona como intermediário entre o navegador e a API da Binance, evitando problemas com CORS, escondendo detalhes técnicos da API e organizando melhor a estrutura da aplicação.

2.2 Requisição dos dados

Nos focamos no gráfico de velas, então na requisição de dados para facilitar a implementação do mesmo dividimos os dados em estruturas chamadas “Candle”, vela em inglês, e cada vela possui tempo de abertura (openTime), valor de abertura (open), valor mais alto (high), valor mais baixo (low), e valor de fechamento (close).

```
export default class Candle{
    constructor(openTime, open, high, low, close){
        this.x = new Date(openTime);
        this.y = [parseFloat(open),parseFloat(high),parseFloat(low),parseFloat(close)];
    }
}
```

E então fazemos a requisição, consumimos a API utilizando o Axios e criamos um array de Candles que é retornado com todas as velas necessárias para o gráfico:

```
import axios from 'axios';
import Candle from './Candle';

export async function getCandles(symbol, interval){
    const response= await axios.get(`http://localhost:3001/klines?symbol=${symbol.toUpperCase()}&interval=${interval}`);
    const candles = response.data.map(k =>{
        return new Candle(k[0], k[1], k[2], k[3], k[4]);
    })
    return candles;
}
```

2.3 Criação do gráfico

Este componente React cria um gráfico de velas (candlestick) utilizando a biblioteca react-apexcharts, que serve como wrapper para o ApexCharts. O gráfico é renderizado de forma dinâmica, recebendo os dados por meio da propriedade props.data, o que permite reutilização e atualização automática conforme os dados mudam.A configuração do gráfico é feita através do objeto options, onde o eixo X (xaxis) é definido como do tipo datetime e formatado para exibir as datas no padrão brasileiro (dd/MM HH:mm), ajustado para o fuso horário de São Paulo. Isso é feito usando um formatter que converte os timestamps para um formato legível com toLocaleString.

O eixo Y (yaxis) possui tooltips ativados para facilitar a leitura dos valores ao passar o mouse sobre as velas. Os dados são organizados dentro da series, com cada ponto contendo um objeto com x (data/hora) e y (um array com os valores de abertura, máxima, mínima e fechamento).Por fim, o gráfico é renderizado com tipo candlestick, largura de 800 pixels e altura de 600 pixels

```jsx
import ApexCharts from 'react-apexcharts';

export default function Chart(props){

    const options = {
        xaxis: {
            type: 'datetime',
            labels: {
              datetimeUTC: false,
              format: 'dd/MM HH:mm',
              formatter: function (value, timestamp) {
                return new Date(timestamp).toLocaleString('pt-BR', {
                  timeZone: 'America/Sao_Paulo',
                  hour: '2-digit',
                  minute: '2-digit',
                  day: '2-digit',
                  month: '2-digit',
                });
              },
            },
          },
        yaxis:{
            tooltip: {
                enable: true
            }
        }
    }

    const series =[{
        data: props.data
      }]

    return(
        <ApexCharts options={options}
                    series={series}
                    type="candlestick"
                    width={800}
                    height={600}/>
    )
}
```

3. Implementação do gráfico no Front

O código começa importando os hooks useEffect e useState do React, além do componente Chart, a função getCandles, o hook useWebSocket da biblioteca react-use-websocket, e a classe Candle. Em seguida, o componente App é definido com dois estados: data, que armazena os candles a serem exibidos no gráfico, e moeda, que representa o par de criptomoeda e moeda selecionadas (inicialmente 'BTCBRL').

Dentro do useEffect, que roda apenas uma vez ao carregar o componente, o código lê a moeda a partir dos parâmetros da URL (por exemplo: ?moeda=ETHBRL) e

atualiza o estado moeda. Logo após, ele chama a função getCandles, que busca os candles históricos de 1 minuto da moeda escolhida, e armazena esses dados no estado data. Se ocorrer algum erro, ele exibe um alerta com a mensagem.

```
import{useEffect, useState } from 'react';
import Chart from './Chart';
import { getCandles } from './DataService';
import useWebSocket from 'react-use-websocket';
import Candle from './Candle';

function App() {
  const [data, setData] = useState([]);
  const [moeda, setMoeda] = useState('BTCBRL');

  useEffect(() => {

    const urlParams = new URLSearchParams(window.location.search);
    const moedaSelecionada = urlParams.get('moeda');
    setMoeda(moedaSelecionada);

    getCandles(moedaSelecionada, '1m')
      .then(data => setData(data))
      .catch(err => alert(err.response ? err.response.data : err.message));
  }, []);
```

Na sequência, é estabelecida uma conexão WebSocket com a API da Binance, usando o par da moeda em minúsculas e o intervalo de 1 minuto. Ao abrir a conexão, é exibida uma mensagem no console. A configuração permite reconectar automaticamente a cada 3 segundos em caso de falha. O onMessage é executado sempre que uma nova mensagem chega, representando um novo candle.

Dentro do onMessage, o código cria um novo objeto Candle com os dados da mensagem (abertura, fechamento, máxima, mínima, etc.). Ele copia os dados atuais (data) para um novo array newData. Se o candle ainda estiver se formando (k.x === false), ele substitui o último candle do array. Se o candle estiver fechado, remove o primeiro da lista e adiciona o novo no final, mantendo o gráfico sempre atualizado com os últimos candles. Por fim, atualiza o estado data.

O return do componente exibe um título com o nome da moeda e renderiza o componente Chart, passando o estado data como propriedade para desenhar o gráfico.

```
const {lastJsonMessage} = useWebSocket(`wss://stream.binance.com:9443/ws/${moeda.toLowerCase()}@kline_${'1m'}`,{
  onOpen: () =>console.log('Connected to Binance'),
  onError: (err) =>console.log(err),
  shouldReconnect: () => true,
  reconnectInterval: 3000,
  onMessage: ()=>{
    if(lastJsonMessage){
      const newCandle = new Candle(lastJsonMessage.k.t,lastJsonMessage.k.o,lastJsonMessage.k.h,lastJsonMessage.k.l,lastJsonMessage.k.c);
      const newData = [...data];
      if(lastJsonMessage.k.x === false){
        newData[newData.lengh - 1] = newCandle;
      }
      else{
        newData.splice(0, 1);
        newData.push(newCandle);
      }
      setData(newData);
    }
  }
,
  })

  return (
    <div>
      <h2>Gráfico: {moeda}</h2>
      <Chart data={data} />
    </div>
  );
}

export default App;
```

3.1 Preço médio atualizado mostrado na tela:

Para mostrar o preço e o atualizar em tempo real fizemos 4 funções de acordo com as Criptomoedas a serem mostradas e atualizar o texto em HTML:

```
graphic > src > callGraphic.js > ...
window.addEventListener('DOMContentLoaded', () => {
    setInterval(() => {
        fetch("https://api.binance.com/api/v3/avgPrice?symbol=BTCBRL")
            .then(response => response.json())
            .then(data => {
                const preco = data.price;
                const elementoPreco = document.getElementById('preco-btc');
                if (elementoPreco) {
                    elementoPreco.textContent = `R$ ${Number(preco).toLocaleString('pt-BR', { minimumFractionDigits: 2 })}`;
                }
            })
            .catch(error => console.error("Erro ao buscar preço:", error));
    }, 1000);
});
```

O código começa registrando um ouvinte de evento com window.addEventListener('DOMContentLoaded', ...), que garante que o JavaScript só será executado depois que todo o HTML tiver sido carregado. Dentro desse evento, é criado um setInterval que executa uma função a cada 1000 milissegundos (ou seja, **a cada 1 segundo**). Essa função faz uma requisição HTTP com fetch para a API pública da Binance, no endpoint https://api.binance.com/api/v3/avgPrice?symbol=BTCBRL. Esse endpoint retorna o preço médio atual do Bitcoin em reais (BRL).

Depois da resposta chegar, ela é convertida em JSON com .json(), e então o valor do campo price é extraído e guardado na constante preco.

Em seguida, o script tenta localizar no HTML um elemento com o ID preco-btc usando document.getElementById. Se esse elemento existir, o código atualiza seu conteúdo

de texto (textContent) com o preço formatado em reais (R$) com duas casas decimais, usando toLocaleString com a localidade pt-BR.

3.2 Resultado na tela

E por ultimo apenas utilizamos um Iframe para poder exibir o gráfico na tela que implementamos e estilizamos.