Ročníkový projekt

26. dubna 2024

Gymnázium, Praha 6, Arabská 14

Arabská 14, Praha 6, 160 00

Předmět: Programování

Téma: Simulace automatizovaného obchodování s bitcoinem

Autoři: *Jiří Petřík, Vítězslav Procházka, Adam Trávníček*

Třída: 3.E

Vyučující: Mgr. Jan Lána

Tř. vyučující: Mgr. Blanka Hniličková

Čestné prohlášení:

Prohlašujeme, že jsem jedinými autory tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené.

Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne ....

V Praze dne ....

V Praze dne ....

Poděkování

Chtěli bychom poděkovat Davidovi Fibigerovi a Matějovi Fibigerovi za pomoc se strojovým učením, Josefu Mitošinkovi za poskytnutí informací o krypto-měnách a krypto-burzách. Dále bychom chtěli poděkovat Mgr. Jiřímu Procházkovi za pomoc při debugování obzvláště zapeklitých problémů. Velké díky také patří Ing. Václavu Chalupníčkovi za velmi srozumitelný úvod do tvorby webových aplikací.

Anotace

Tato dokumentace popisuje týmový ročníkový projekt, na téma: simulace automatizovaného obchodování s Bitcoinem. Cílem tohoto projektu je vytvořit Python desktop aplikaci, která má přístup k online krypto burze a komunikuje s ní. V jádru aplikace bude AI, které sleduje různá reálná data z online krypto burzy. Na základě rozhodnutí tohoto AI bude aplikace hospodařit se simulovaným portfoliem. Aplikace poběží lokálně a bude mít plně funkční webové uživatelské rozhraní.

Klíčová slova

Simulace; Bitcoin; strojové učení; webová aplikace; trh; kryptoměna; umělá inteligence; obchodovaní; graf;

Abstract (English)

This documentation describes a team year-long project on the topic: simulation of automated trading with Bitcoin. The goal of this project is to create a Python desktop application that has access to an online crypto exchange and communicates with it. At the core of the application will be AI that monitors various real-time data from the online crypto exchange. Based on the decisions of this AI, the application will manage a simulated portfolio. The application will run locally and will have a fully functional web user interface.

Keywords

Simulation; bitcoin; machine learning; web application; market; cryptocurrency; artificial intelligence; trading; graph;

Obsah

1 Úvod

Bitcoin, jako průkopník v oblasti kryptoměn, přilákal pozornost nejenom investičních entuziastů, ale i vědecké komunity. Jeho volatilita a potenciál na trzích poskytují zajímavou základnu pro výzkum automatizovaných obchodních strategií. Automatizované obchodování, často označované jako algoritmické obchodování, je proces využívající počítačové programy k vytváření a řízení obchodních rozhodnutí na finančních trzích. Tato práce se zabývá simulací automatizovaného obchodování s Bitcoinem, s cílem zjistit, zdali je AI schopna porozumět velkému ruchu ve finančních datech. Bitcoin, jako decentralizovaná kryptoměna, není čistě ovlivňován tradičními ekonomickými faktory jako jsou zprávy o ekonomickém růstu nebo politické události. Jeho cena je spíše závislá na nabídce a poptávce na burzách – tyto cenové výkyvy se lidem po většinu času zdají náhodné nebo alespoň nepředvídatelné; zejména pak při nízkém časovém rozmezí (například jedna minuta). To z Bitcoinu činí zajímavý objekt pro studium algoritmického obchodování. Simulace takového obchodování může poskytnout cenné poznatky pro investory, kteří hledají efektivní strategie pro obchodování s Bitcoinem.

Jméno projektu Prophet (z angličtiny prorok) mělo vyznačovat vidění do budoucna a předpovídání v našem případě ceny Bitcoinu.

Obrázek 1: Prophet

1.1 Zadání projektu

Cílem tohoto projektu je vytvořit Python desktop aplikaci, která má přístup k online krypto burze a komunikuje s ní. V jádru aplikace bude AI, které sleduje různá reálná data z online krypto burzy. Na základě rozhodnutí tohoto AI bude aplikace hospodařit se simulovaným portfoliem. Aplikace poběží lokálně a bude mít plně funkční webový interface.

1.2 Důvod výběru tématu

Toto téma jsme si vybrali, protože jsme chtěli zkusit udělat něco, co by nám rozšířilo poznatky a motivovalo k získání nových znalostí, o kterých jsme v začátcích nic nevěděli. Dále nám téma přišlo velmi zajímavé z hlediska potřebných technologií, ze jména pak práce s API a tvorbou vlastních AI modelů. Důležité je také zmínit, že nám přišlo, že je možné projekt rozdělit na několik důležitých částí tak, abychom si příliš nepřekáželi. A v neposlední řadě to byla touha zkusit, zdali vůbec něco takového lze naprogramovat.

2 Bitcoin

Bitcoin je snad nejznámější kryptoměnou na světě, kterou vytvořil anonymní vývojář nebo skupina vývojářů pod pseudonymem Satoshi Nakamoto. Bitcoin byl uveden na trh v roce 2009 jako open-source software. Tento vynález vyvolal revoluci v oblasti digitálního měnového systému a přispěl k vývoji nových technologií, jako je blockchain.

2.1 Co je Bitcoin?

Bitcoin je digitální měna, která umožňuje peer-to-peer transakce bez potřeby zprostředkovatelů, jako jsou banky nebo vlády. Transakce jsou zaznamenávány v distribuované veřejné knize zvané blockchain. Tato technologie zajišťuje transparentnost a bezpečnost, a to proto, že jakýkoliv pokus o manipulaci s transakcemi by byl snadno odhalen.

2.2 Jak Bitcoin funguje?

Bitcoin funguje na technologii blockchain, což je decentralizovaný systém, kde každý účastník má kopii všech transakcí. Nové transakce jsou ověřovány těžaři, kteří používají výpočetní výkon k řešení složitých kryptografických problémů. Když těžaři úspěšně ověří transakci, je přidána do bloku, který se následně přidá do blockchainu.

2.3 Vlastnictví a anonymita

Vlastnictví Bitcoinů je určeno pomocí kryptografických klíčů. Každý uživatel má veřejný a soukromý klíč. Veřejný klíč je adresa, na kterou mohou být Bitcoiny odesílány, zatímco soukromý klíč slouží k přístupu k těmto prostředkům. Bitcoin umožňuje určitou úroveň anonymity, protože transakce jsou spojeny s adresami, a ne se jmény konkrétních lidí. Nicméně s dostatečným množstvím informací mohou být transakce zpětně sledovány.

2.4 Výhody

- Decentralizace: Bitcoin není řízen centrální autoritou, což znamená, že není závislý na žádné vládě ani finanční instituci. Transparentnost: Všechny transakce jsou veřejné, což umožňuje kontrolu a sledování.
- Rychlé a globální transakce: Bitcoin umožňuje rychlé a levné mezinárodní transakce bez zprostředkovatelů.
- Ochrana proti inflaci: Celkový počet Bitcoinů je omezen na 21 milionů, což by mělo pomoci zachovat jeho hodnotu v dlouhodobém horizontu.

2.5 Nevýhody

- Volatilita: Bitcoin je známý pro svou cenovou nestabilitu, což může být rizikem pro investory. Bezpečnostní rizika: Uživatelé jsou zodpovědní za bezpečnost svých soukromých klíčů. Ztráta klíče znamená ztrátu Bitcoinů.
- Právní nejistota: Bitcoin a kryptoměny obecně jsou předmětem různých regulací, které se mohou lišit podle země.
- Environmentální dopad: Těžba Bitcoinu vyžaduje velké množství energie, což má dopad na životní prostředí.

2.6 Budoucnost Bitcoinu

Bitcoin stále prochází vývojem, a to jak technologickým, tak i z hlediska adopce. Mnoho společností začalo přijímat Bitcoin jako platbu, a také se objevují nové aplikace založené na blockchainu. Nicméně budoucnost Bitcoinu zůstává nejistá, zejména kvůli regulačním výzvám a obavám o bezpečnost. Bitcoin je průkopníkem v oblasti kryptoměn a ovlivnil mnoho aspektů finančního světa. Jeho decentralizovaný charakter a inovativní technologie ho činí atraktivním, ale zároveň představuje výzvy a rizika, která je třeba pečlivě zvážit. [1] [2]

3 Použité technologie

V našem projektu jsme se rozhodli využít tyto programovací jazyky, frameworky a významné knihovny.

1. Python 3.11 - Backendová část, logika automatizovaného obchodování, učení AI a stahování dat z online burzy
    - PyTorch - knihovna pro strojové učení
    - Flask - mikroframework pro manipulaci s HTTP požadavky a odpověďmi
2. Javascript – Logika uživatelského rozhraní - výpočty zobrazovaných údajů, funkcionalita tlačítek, překlad
    - Chart js - vykreslení grafu
    - JQuery - hledání elementů pro překlad
3. HTML - Struktura frontendové aplikace
    - Bootstrap v.5.3.2 - front-endový framework pro vývoj responzivních webových stránek a aplikací
4. CSS - Vzhled a design uživatelského rozhraní

4 Závislosti

Zde se nachází seznam všech Python knihoven, které byly využity v našem projekty. K nalezení jsou v souboru *requirements.txt* . Pro jejich rychlou instalaci se dá použít příkaz do příkazové řádky: *pip3 install -r requirements.txt*

4.1 Flask

Pro spojení backendu a frontendu jsme využili mikroframework Flask, který se využívá pro vývoj webových aplikací v jazyce Python. Flask poskytuje základní funkce pro routování URL, správu šablon, manipulaci s HTTP požadavky a odpověďmi. Jeho používání je velmi jednoduché a pro naši velikost projektu více než dostačující. Po spuštění aplikace si Flask vytvoří vlastní lokální server, který může uživatel v prohlížeči otevřít. [3, 4]

4.2 NumPy

Běžně používaná knihovna, která poskytuje výkonné nástroje pro práci s vícerozměrnými poli a matematickými funkcemi. Tuto knihovnu jsme využili hlavně při manipulaci s tržními daty, především jejich transformaci na požadovaný formát časových sekvencí. [5]

4.3 Pandas

Jedná se o rozsáhle používanou knihovnu vhodnou pro manipulaci s daty a datovými strukturami. Tato knihovna byla často používána právě v kombinaci s NumPy pro manipulaci a transformaci dat. [6]

4.4 Matplotlib

Matplotlib je knihovna pro vizualizaci dat v Pythonu, která umožňuje tvorbu statických, animovaných a interaktivních grafů; tato knihovna byla využita u tvorby grafů po trénování modelu a zpětném testování. [7, 8]

4.5 Pytorch

PyTorch je open-source knihovna strojového učení, kterou vyvinul Facebook's AI Research lab (FAIR). Je to jedno z nejpopulárnějších nástrojů pro vývoj a výzkum v oblasti umělé inteligence díky své komplexnosti a flexibilitě[9]; zpětně si ale nejsme jistí, zda to byla ta nejlepší cesta, kterou se vydat. Pytorch hodnotíme až příliš jako komplexní a komplikovaný nástroj pro začátečníky v oboru.

Pro trénování AI modelu jsme použili platformu CUDA od společnosti NVIDIA, která nám umožnila výrazně zrychlit tento proces. Díky možnosti provádět trénovací cyklus na grafické kartě. CUDA pro PyTorch je pro instalaci k nalezení zde. [10]

5 Zdroj tržních dat Bitcoinu

Získat zdarma solidní zdroj tržních údajů o bitcoinu se zprvu zdálo jako velký problém, obzvláště data nízkého časového rozmezí. Zprvu jsme data brali z různých amatérských projektů na Kaggle, ale to se velmi rychle ukázalo jakožto nevhodný způsob. Problém byl v tom že data byla jen z omezeného období, různého formátu, a hlavně jejich věrohodnost byla velmi pochybná. Rozhodli jsme se tedy použít API, nakonec se jako nejlepší možnost ukázala Binace API. [11] Toto API je velmi lehké na používání díky oficiální dokumentaci, která je k nalezení zde. Velké pozitivum jsme shledali v tom, že nabízí velmi úzké časové rozmezí dat, a to až jednu vteřinu. Další výhody této API jsou, že se jedná o oficiální API od největší krypto burzy a že využití API je zdarma. Toto API jsme nakonec použili, jednak pro stahování historických dat na trénování AI, jednak pro získávání aktuálních tržních údajů o bitcoinu pro simulování chování AI na současných datech.
Na ukázce 1 popsaný kousek kódu, který stahuje data v minutovém rozmezí za určitou dobu.

Funkce dostane jako parametr začátek období *start date string* a konec období *end date string*(ve formátu '1 Jan 2022'), string v tomto formátu se převede na požadovaný údaj, pokud se nenastaví konec období, funkce pak bude jako konec období brát aktuální datum. , dále se zde dá nastavit jaký interval *interval* neboli v jakém časové rozhraní mají data být (pro data, která jsou od sebe vzdálená jednu minutu se použije "1M").

Na řádku 9 se odešle sestavený dotaz na Binance, který odpoví požadovanými daty; kline stojí pro candlesitck data. Dále se zahodí nepotřebné sloupky údajů a data se uloží do csv souboru.

Ukázka 1: binance_data_fetcher.py, get_historical_data

```
19 def get_historical_data(symbol, interval, start_date_string,
20                         end_date_string = 'not_given', ouput_file = 'not_given'):
21     start_date = datetime.strptime(start_date_string, '%d %b %Y')
22     if end_date_string == 'not_given':
23         today = datetime.now(utc_timezone)
24         end_date = today
25     else:
26          end_date = datetime.strptime(end_date_string, '%d %b %Y')
27     klines = bclient.get_historical_klines(symbol, interval, start_date.strftime("%d %b %Y %H:%M:%S"),
28                                            end_date.strftime("%d %b %Y %H:%M:%S"), 1000)
29     data = pd.DataFrame(klines, columns = ['timestamp', 'open', 'high', 'low', 'close',
30                                            'volume', 'close_time', 'quote_av', 'trades'
31                                            , 'tb_base_av', 'tb_quote_av', 'ignore' ])
32     # converts time stamp to date
33     data['Date'] = pd.to_datetime(data['timestamp'], unit='ms')
34     data = data.drop('timestamp', axis=1)
35     # removes no important columns
36     drop_column_names = ['close_time', 'quote_av', 'trades', 'tb_base_av', 'tb_quote_av', 'ignore']
37     for column_name in drop_column_names:
38         data = data.drop(column_name, axis= 1)
39     data.to_csv(filename)
```

6 Model umělé inteligence

Náš predikční model jsme naprogramovali v jazyce Python pomocí knihovny PyTorch. Jeho jádrem je LSTM vrstva, která se zaměřuje na časové sekvence dat. Trénování modelu a ostatní věci okolo něj se nachází v souboru *best_brain.py*.

6.1 Řešení problematiky s AI

Trhy s akciemi a kryptoměnami denně vykazují ohromné množství dat, tyto data jsou často velmi chaotické a člověk v nich hledá relevantní informace velmi složitě, byť si napomáhá grafy a různými finančními indikátory. Tato nesmysluplnost údajů se zvětšuje s nižším časovým rozmezím; například velmi běžná, avšak velmi chaotická, situace je případ, kdy si někdo koupí v jeden moment ohromné množství Bitcoinu, projeví se to výrazně více na tvaru grafu s minutovým časovým rozmezím nežli na grafu s denním časovým rozmezím, kde se taková transakce (byť je sebe větší) ztratí. Navíc dělání obchodů v krátkých časových intervalech není možné pro člověka dělat konzistentně. Tudíž často ani nemá pro člověka smysl se zabývat těmito daty.
Proto jsme se rozhodli udělat systém, který by byl schopen nalézt a naučit se skryté souvislosti a vzorce v časové posloupnosti cen Bitcoinu. Tyto naučené vzorce by potom aplikoval na data v reálném čase a na základě výsledku predikčního modelu by simuloval obchody a portfolio toho, jak by to teoreticky vypadalo, kdyby měl přístup k reálným penězům.
Jakožto vstupní data do modelu jsme se rozhodli použít časové sekvence čistě finančních údajů bitcoinu a nepoužívali jsme externí nástroje na vyhodnocování nálady veřejnosti jako jsou googletrends. Jak dále je zmíněno v kapitole 6.3 *Architektura AI*, rozhodli jsme se pro časový úsek 10 minut, kde po sobě jdou rozdíly uzavíracích cen Bitcoinu pro danou minutu. Model se pak snaží předpovědět budoucí vývoj Bitcoinu pro příští minutu.

6.2 Vstupní data do modelu

Jak již bylo zmíněno, model se učil na sekvencích o délce 10 minut, kde po minutě po sobě šli jednotlivé údaje. Jakožto velice nevhodná data do modelu se ukázaly ceny jako takové, kdy se model naučil hádat cenu *t+1* až téměř totožnou s jeho výchozí pozicí *t*, protože tím pak průměrně udělal nejmenší chybu.
Proto jsme se rozhodli do modelu posílat sekvence procentuálních rozdílů za sebou jdoucích cen BTC. Udělal se procentuální rozdíl minuty [*t-9 od t-8, t-8 od t-7,.....t-1 od t*] model se pak snažil předpovědět rozdíl *t od t+1* .[12] Toto transformování dat probíhá v souboru *data_manager.py*, kde za vypíchnutí stojí funkce *prepare_dataframe_for_lstm2*, která z posloupnosti cen vyrobí posloupnost rozdílů.

6.3 Architektura AI

Jádrem našeho predikčního modelu je LSTM vrstva, která je vhodná pro předpovídaní dalších hodnot x na základě vstupních hodnot X; dalo by se říct, že se jedná o stejné proměnné, jenom jsou zaznamenány v jiném čase. Síla této vrstvy tkví v tom, že je schopna rozlišit dlouhodobé souvislosti mezi daty od těch krátkodobých, nepodstatné informace je schopna zapomenout. [13, 14]
Na ukázce 2 je vidět kus kódu, kde je definovaný náš LSTM model. Na řádku 14 se definuje první (vstupní), je typu lstm a má parametry: *input_size* - počet různých údajů pro každý bod v časové sekvenci(u nás tedy 1), *hidden_size* počet neuronů v této vrstvě a *num_stacked_layers* který nastavuje kolik lstm vrstev bude model mít. Na řádku 17 se přidá lineární vrstva, která zároveň slouží jako výstupní vrstva, ta dostane jako parametr taky počet neuronů *hidden_size*.

Ukázka 2: best_brain.py, tříá LSTM

```
9  class LSTM(nn.Module):# this class inherits from nn.Module
10     def __init__(self, input_size, hidden_size, num_stacked_layers):
11         super().__init__()
12         self.hidden_size = hidden_size
13         self.num_stacked_layers = num_stacked_layers
14         self.lstm = nn.LSTM(input_size, hidden_size, num_stacked_layers,
15                             batch_first=True)
16         # defines linear function with single ouput neuron
17         self.fc = nn.Linear(hidden_size, 1)
```

V našem modelu jsme se nakonec rozhodli pro konfiguraci 8 neuronů a 1 lstm vrstvy, jednalo se o dobrý kompromis mezi výkonem a časové náročnosti pro vytrénování. Dále na ukázce 3 je k nalezení funkce *forward*, která určuje jakým způsobem prochází data skrz neurální síť. Na řádku 25 lstm vrstva vrátí výsledek *out* a skrytý stav _ , který je zapomenut a dále se počítá už jen s výstupem *out*.

Ukázka 3: best_brain.py, class LSTM , forward

```
18    # function that describes how the data move throgh the model
19    def forward(self, x):
20        batch_size = x.size(0)
21        # Initial hidden and cell states of the LSTM
22        h0 = torch.zeros(self.num_stacked_layers, batch_size, self.hidden_size).to(device)
23        c0 = torch.zeros(self.num_stacked_layers, batch_size, self.hidden_size).to(device)
24        # "_" means that we will denote tuple that contains hidden and cell state at the last step
25        out, _ = self.lstm(x, (h0, c0))
26        out = self.fc(out[:, -1, :])
27        return out
```

Hyperparametry modelu byly nastaveny tak, aby tvořily kompromis mezi výkonem a náročností trénování, proto byl zvolen *look_back* = *9* (kolik údajů bude v sekvenci do minulosti: *t* - *9*). Velikost batchů *batch_size* doprovázela stejnou strategii kompromisu. Žádné hyperparametry ani parametry nebyly vybrány pomocí optimalizační metody.

6.4 Trénování AI

Trénování se spouští v souboru *model_trainer.ypnb* kde se dají nastavit trénovací parametry, jako je počet epoch (kolikrát modelem projdou všechna data), learning_rate (jak agresivní učení bude). Samotný training loop je napsaný v souboru *best_brain.py*. Trénovací data po všech transformacích jsou přeměna do tensoru a jsou rozdělena na batche; batch je sada, u nás 16, vstupních dat, po projetí jednoho batche se aktualizují parametry neurální sítě. Tato metoda zrychluje učení. Při trénování jsme využili MSE loss funkce od PyTorch. [12]

7 Zpětné testování

Zpětné testování, tzv. backtesting, je metoda zpětného ověření funkcionality automatizovaného systému na obchodování. [15] I my jsme si takový systém napsali a to v jazyce Python, v projektu se jedná o soubor *back_tester.py*. Tento script umožňuje simulovat chování našeho modelu na historických datech; takových, která neviděl při trénování, jinak by proces neměl moc smysl. Modelu jsou zde ve smyčce posílány jednotlivé sekvence cenových posunů, které jsou následně transformovány na požadovaný formát a odeslány do modelu. Model sekvenci vyhodnotí a vrátí odhadovaný pohyb od aktuální ceny. Na základě tohoto údaje program nasimuluje otevření pozice (samotné otevření pozice nastane v souboru: *trader.py*).

Na ukázce 4 se nachází smyčka uvnitř funkce *back_test_loop* která je klíčovou funkcionalitou tohoto kódu. Jako vstupní parametry funkce *back_test_loop* dostane *start_usd_balance* - s kolika americkými dolary bude algoritmus začínat (my jsme vždy používali 10 000 dolarů), *leverage* a *commission_rate*; vysvětlení těchto parametrů je k nalezení v kapitole 8.4 *Algoritmus pro simulování obchodů*. Při spouštění této smyčky jsou již definované proměnné například kolik bitcoinu má algoritmus, který Bitcoin koupí pouze jednou a následně ho jenom drží (buy and hold) - *bh_btc_balance*, obdobně je na tom *sh_btc_balance* (nejdříve prodá bitcoin a pak pozici drží).
Smyčka za sebou prochází jednotlivé sekvence procentuálních rozdílů cen a zároveň iteruje i skrze index, aby se pak zpětně dala snadněji dohledat cena bitcoinu pro aktuální datovou sekvenci.
Na řádku 83 se aktualizuje cena Bitcoinu pro danou iteraci. Dále se spočítá, kolik by každá ze tří strategií měla čistě v amerických dolarech, pokud by uzavřela všechny obchody. Po sléze se přidají tyto hodnoty do listů, kde má každá strategie uchovanou celou historii hodnot svého portfolia.
v následujících řádkách kódu se vyhodnotí předchozí obchod, zjistí se zdali to byl úspěšný obchod či nikoliv a kolik vydělal nebo ztratil. Na řádku 106 se nastaví minulá hodnota portfolia na tu aktuální. Aby se statistky mohly znovu propočítat v příští iteraci.

Na řádku 109 a 110 se pak už jenom sekvence 10 cenových rozdílů převede na tensor.

Na řádku 112 se tensor odešle na vyhodnocení do modelu, který vrátí předpovídaný posun ceny. Na základě této předpovědi se na řádku 114 upraví obchodní pozice; fungování tohoto systému je k nalezení v kapitole 8.4 *Algoritmus pro simulování obchodů.*

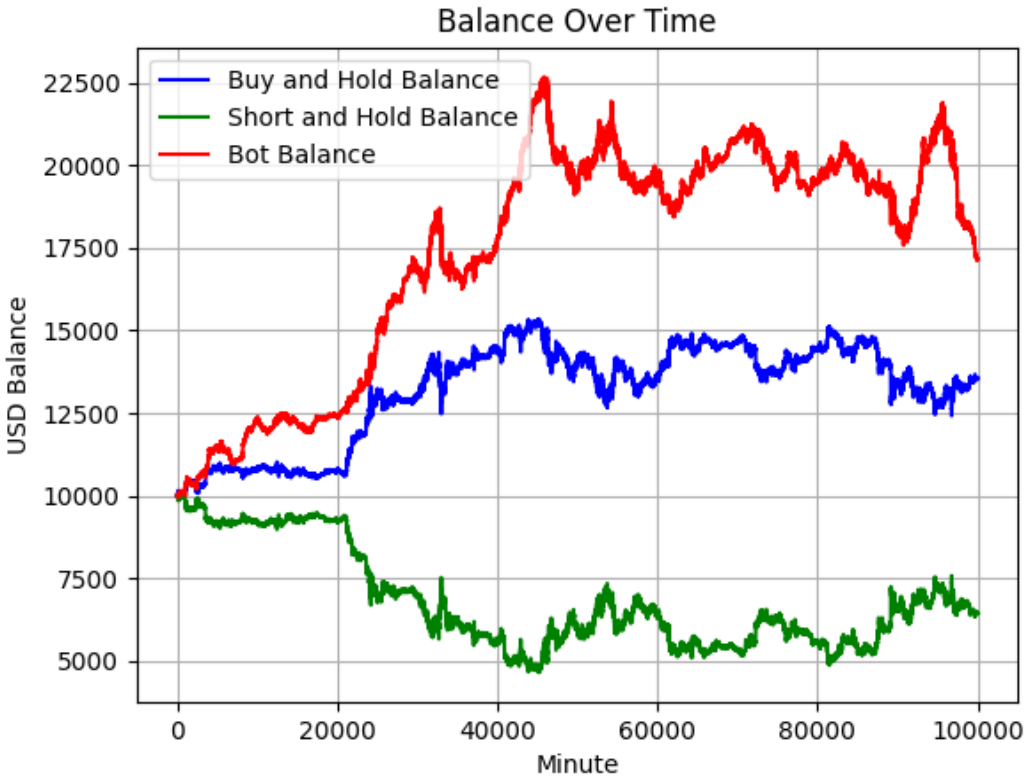
Ukázka 4: back_tester.py, smyčka uvnitř back_test_loop

```
82  for index, one_sequence in enumerate(prepared_data_as_np):
83      current_btc_price = get_btc_price_for_current_sequence(index)
84      usd_balance_after_close,_ = trader.close_trade(usd_balance,btc_balance
85                                                  ,current_btc_price, commission_rate)
86      bh_usd_balance_test, _ = trader.close_trade(bh_usd_balance, bh_btc_balance
87                                                , current_btc_price, commission_rate)
88      sh_usd_balance_test, _ = trader.close_trade(sh_usd_balance,sh_btc_balance
89                                                ,current_btc_price, commission_rate)
90      usd_balance_history.append(usd_balance_after_close) # Remembers usd balance of algo
91      bh_usd_balance_history.append(bh_usd_balance_test) # Remembers usd balance of buy and hold
92      sh_usd_balance_history.append(sh_usd_balance_test) # Remembers usd balance of short and hold
93      if (last_trade == 'long'):
94          long_count += 1
95      elif (last_trade == 'short'):
96          short_count += 1
97      else:
98          hold_count +=1
99
100     if( last_usd_balance > usd_balance_after_close):
101         bad_trade_count += 1
102         usd_lost = last_usd_balance - usd_balance_after_close
103     else:
104         good_trade_count += 1
105         usd_gained = usd_balance_after_close - last_usd_balance
106     last_usd_balance = usd_balance_after_close
107
108     # Tranforms sequence to correct form
109     one_sequence = one_sequence.reshape((-1, bb.look_back * 1 + 1 , 1))
110     one_sequence_tensor = torch.tensor(one_sequence).float()
111
112     prediction = bb.make_one_prediction(one_sequence_tensor)
113
114     usd_balance, btc_balance, last_trade = trader.make_one_trade(prediction,usd_balance,btc_balance
115                                                                 ,current_btc_price, commission_rate
116                                                                 , last_trade, leverage)
```

7.1 Výsledky zpětného testování

Model se jménem *historical_model.pth*, který byl trénován na datech od začátku 2022 do dne 11. února 2024. Díky tomu jsme si nechali cca 70 dnů (100 000 minut) dat na zpětné testování.

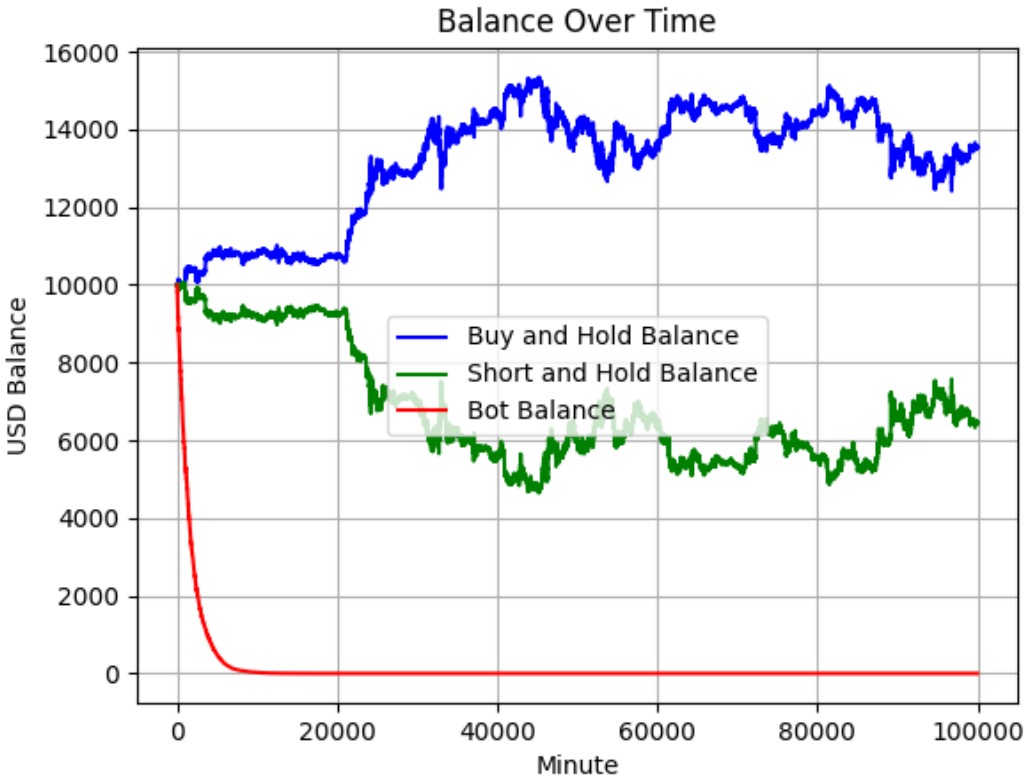
Na obrázku 2 je vidět červený graf našeho algoritmu, který během 100 000 minut poměrně konzistentně překoná buy and hold strategii (modrá) i short and hold (zelená). Ke konci červené křivky můžeme pozorovat zhoršení výsledku, které by se dalo vysvětlit příliš velkou mezerou od trénování.



Obrázek 2: Výsledek zpětného testování při 0% poplatků

Během této doby bot udělal 25391 dlouhých pozic, 25391 krátkých pozic a 49207krát pozici držel. Jeho úspěšnost předpovědi byla 51.06% a z jednoho dolaru udělal 1.7144 dolaru tedy z 10 000 dolaru udělal 17 144 dolarů.

Z obrázku 3 je patrné, že byť se jedná o malý poplatek (0.05%), algoritmus to totálně potopí. To se dá vysvětlit tím, že i když bychom udělali 100% správných obchodů, za každé otevření pozice a její zavření bychom ve většině času zaplatili víc, než bychom vydělali. Je tedy jasné, že takovýto systém nemá šanci v reálných podmínkách uspět; a to náš systém nepočítá s věcmi jako je spread, či s pohybem cen během jedné minuty (náš bot bere v potaz jenom uzavírací ceny z dokončené minuty).



Obrázek 3: Výsledek zpětného testování při poplatku 0.05% za 1 obchod

8 Backend Aplikace

Backend naší aplikace je napsán v jazyce Python, jeho hlavní funkcí je simulace automatizovaného obchodování v reálném čase. Jádrem této aplikace je model se jménem *acutal_model.pth*, který byl trénován na datech od začátku roku 2022 do dne 25.4.2024 18:00. Je tak vytrénován na co nejvíce aktuálních datech.

8.1 Ukládání dat do Jsonu

Jelikož je naše aplikace zamýšlena tak, aby se simulace na aktuálních datech dala kdykoliv vypnout. Bylo potřeba zařídit ukládání klíčových dat do souboru. Pro jednoduchost jsme se rozhodli použít Soubor typu json. Nevyužíváme žádné šifrování a soubor se dá kdykoliv ručně přepsat, ale jelikož se jedná jen o simulaci vlastně se jedná o pozitivní vlastnost; uživatel si pak může zadat libovolná data.
Na ukázce 5 je vidět příklad toho jak vypadá json po delší době simulování obchodů.

1. time_spent_trading - čas ve vteřinách, který algoritmus celkově strávil obchodováním
2. USD_balance - kolik amerických dolarů obsahuje portfolio (nikoliv celková hodnoto portfolia)
3. BTC_balance - kolik bitcoinu obsahuje portfolio
4. long_count - počet dlouhých pozic (nákup bitcoinu)
5. short_count - počet krákých pozic (prodání bitcoinu, kterého nevlastním)
6. hold_count - počet kolikrát algoritmus držel aktuální pozici
7. bad_trade_count - kolik prodělečných obchodů algoritmus udělal
8. good_trade_count - kolik výdělečných obchodů algoritmus udělal
9. total_profit - kolik peněz algoritmus celkově vydělal při výdělečných obchodech
10. total_loss - kolik peněz algoritmus celkově prodělal při prodělečných obchodech
11. leverage - páka se kterou systém simuluje obchody

12. commission_rate - kolik procent z každého obchodu zaplatí systém na poplatcích

13. is_trading - boolean hodnota, která určuje zda je aktuálně systém aktivní, hodí se hlavně v kombinaci s GUI

Z těchto hodnot se dále počítá winrate (úspěšnost) a další zajímavé statistiky již na lokální webové stránce.

Ukázka 5: data/trade_data.json

```
{
    "time_spent_trading": 24628,
    "USD_balance": 19457.649752040597,
    "BTC_balance": -0.153258110838379,
    "long_count": 91,
    "short_count": 92,
    "hold_count": 224,
    "bad_trade_count": 202,
    "good_trade_count": 205,
    "total_profit": 3424.75748277779,
    "total_loss": 3695.932606757491,
    "leverage": 1,
    "commission_rate": 0,
    "is_trading": true
}
```

V ukázce 6 je možné nalézt jednoduchou funkci na aktualizování konkretní hodnoty v jsonu. Funkce nejdříve json načte jakožto dictionary, aktualizuje danou hodnotu a aktualizovaný dictionary uloží. [16]

Ukázka 6: json_data_handler.py, update_trading_data

```
6  def update_trading_data(key, value, filename=trade_data_file_name):
7      # Ensure that the data directory exists
8      os.makedirs(os.path.dirname(filename), exist_ok=True)
9
10     # Load existing data if the file exists
11     try:
12         with open(filename, 'r') as file:
13             trading_data = json.load(file)
14     except FileNotFoundError:
15         trading_data = {}
16
17     # Update trading data with new values
18     trading_data[key] = value
19
20     # Open the file in write mode and save the updated data as JSON
21     with open(filename, 'w') as file:
22         json.dump(trading_data, file, indent=4)
```

Při načítání dat z jsonu se občas vyskytne error:

```
Traceback (most recent call last):
  File "<python_library>/threading.py", line 1038, in _bootstrap_inner
    self.run()
  File "<your_project_directory>/live_trading_bot.py", line 28, in run
    json_data_handler.update_trading_data("is_trading", True)
  File "<your_project_directory>/json_data_handler.py", line 13, in update_trading_data
    trading_data = json.load(file)
                   ^^^^^^^^^^^^^^^
  File "<python_library>/json/__init__.py", line 293, in load
    return loads(fp.read(),
           ^^^^^^^^^^^^^^^^
  File "<python_library>/json/__init__.py", line 346, in loads
    return _default_decoder.decode(s)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "<python_library>/json/decoder.py", line 337, in decode
    obj, end = self.raw_decode(s, idx=_w(s, 0).end())
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "<python_library>/json/decoder.py", line 355, in raw_decode
    raise JSONDecodeError("Expecting value", s, err.value) from None
json.decoder.JSONDecodeError: Expecting value: line 1 column 1 (char 0)
```

O kterém nevíme proč nastane nebo jak ho vyřešit. Naštěstí error není fatální a resetování jsonu pomocí funkce na webové stránce umožní programu pokračovat.

8.2 Ukládání historie portfolia do csv

Výsledky simulace na reálných současných datech jsou ukládány do csv souboru *data/balance_history.csv*. Na každém řádku v tomto souboru je k nalezení celková hodnota portfolia v USD pro daný moment a timestamp udávaný ve vteřinách.
Zde na ukázce 7 se nachází jednoduchá funkce na přidání nového řádku do csv souboru. Funkce soubor nejdříve otevře a pomocí knihovní funkce *csv.writer* soubor upraví. [17]

Ukázka 7: csv_data_handler.py, append_row_to_csv

```
17 def append_row_to_csv(usd_balance, timestamp):
18     # Open the CSV file in append mode
19     with open(file_path, 'a', newline='') as csvfile:
20         # Create a CSV writer object
21         csv_writer = csv.writer(csvfile)
22 
23         # Write the new row to the CSV file
24         csv_writer.writerow([usd_balance, timestamp])
```

8.3 Získávání aktuálních tržních dat

Pro získávání dat v reálném čase jsme použili Binance API, již zmíněnou v kapitole 5 *Zdroj tržních dat Bitcoinu*. S tím rozdílem, že tentokrát jsme na Binance posílali dotaz jen na data za posledních 11 minut. Protože na to abychom spočítali sekvenci deseti procentuálních změn potřebujeme 11 cenových údajů.

8.4 Algoritmus pro simulování obchodů

Tento algoritmus se nachází v souboru *trader.py*, v tomto souboru jsou funkce na simulované otevření dlouhé pozice, krátké pozice a zavření pozice. Jako další se tu nachází algoritmus, který rozhoduje jaký, obchod bot udělá. V neposlední řadě je zde k nalezení funkce pro výpočet poplatků.

8.4.1 Dlouhá pozice

Dlouhá pozice,tzv. long position, je druh obchodu kde dochází ke koupi aktiva. V případě větší páky než 1 dochází k zadlužení v penězích; například při páce 2 a zůstatku 100 dolarů si bot koupí bitcoin za 100 dolarů + za dalších 100 dolarů, jeho výsledné portfolium se bude skládat z x bitcoinu a -100 dolarů. Vyšší páka tak zvýrazňuje pohyby, ale i ztráty.[18]

Ukázka 8 vyobrazuje funkci pro simulování dlouhé pozice. Vypočítá množství USD k nákupu: Z aktuálního zůstatku USD a zvolené páky (leverage) spočítá částku v USD, kterou lze použít k nákupu Bitcoinů. Nakoupí Bitcoiny: Tuto částku využije k nákupu Bitcoiny za aktuální cenu. Aktualizuje zůstatky: Sníží zůstatek v USD o částku použitou k nákupu, a zvýší zůstatek v Bitcoinech o množství nakoupených Bitcoinů. Pokud je zadaná provize větší než 0, zavolá pomocnou funkci pro výpočet zůstatků po započtení provize. Vrátí aktualizované zůstatky.

Ukázka 8: trader.py, long_position

```
16 def long_position(usd_balance, btc_balance,leverage,current_btc_price, comission_rate):
17     usd_to_buy_with = usd_balance * leverage
18     btc_bought = usd_to_buy_with / current_btc_price
19 
20     usd_balance -= usd_to_buy_with
21     btc_balance += btc_bought
22     if comission_rate > 0:
23         usd_balance,btc_balance = balance_after_commission(usd_balance,
24                                                            btc_balance
25                                                            ,comission_rate, Buy=True)
26     return usd_balance, btc_balance
```

8.4.2 Krátká pozice

Krátká pozice, známá také jako short position, je opakem dlouhé pozice. Při krátké pozici investor prodává aktivum, které ještě nevlastní, v naději, že jeho cena klesne. Když cena skutečně klesne, investor si může opět koupit aktivum za nižší cenu, než za kolik ho prodal, a tím si zajistí zisk. Také zde může docházet k využívání finanční páky, což zvyšuje potenciální zisk i riziko ztrát. [19]

Ukázka 9 vyobrazuje funkci pro simulování krátké pozice. Vypočítá množství USD k prodeji: Z aktuálního zůstatku USD a zvolené páky (leverage) spočítá částku v USD, kterou lze použít k prodeji Bitcoinů. Provede prodej Bitcoinů: Tuto částku využije k prodeji Bitcoinů za aktuální cenu. Aktualizuje zůstatky: Zvýší zůstatek v USD o částku získanou prodejem a sníží zůstatek v Bitcoinech o množství prodaných Bitcoinů. Zohlední provize: Pokud je zadaná provize větší než 0, zavolá pomocnou funkci pro výpočet zůstatků po započtení provize. Vrátí aktualizované zůstatky: Vrátí nové hodnoty zůstatku v USD a počtu Bitcoinů.

Ukázka 9: trader.py, long_position

```
25 def short_position(usd_balance, btc_balance,leverage,current_btc_price, comission_rate):
26     usd_to_sell_with = usd_balance * leverage
27     btc_sold = usd_to_sell_with / current_btc_price
28 
29     usd_balance += usd_to_sell_with
30     btc_balance -= btc_sold
31     if comission_rate > 0:
32         usd_balance,btc_balance = balance_after_commission(usd_balance,
33                                                            btc_balance,
34                                                            comission_rate, Buy=False)
35     return usd_balance, btc_balance
```

8.4.3 Zavření obchodu

Tato funkce na ukázce 10 nasimuluje zavření předchozí pozice. Ve své podstatě jenom prodá bitcoin (případně dluh bitcoinu - zápornou hodnotu BTC) za aktuální cenu. To přidá k zůstatky v USD a zůstatek BTC nastaví na 0.

Ukázka 10: trader.py, close_trade

```
34 def close_trade(usd_balance, btc_balance, current_btc_price, comission_rate):
35     usd_will_get = btc_balance * current_btc_price
36     usd_balance += usd_will_get
37     btc_balance = 0
38     if comission_rate > 0:
39         usd_balance, btc_balance = balance_after_commission(usd_balance,
40                                                             btc_balance,
41                                                             comission_rate, Buy=False)
42     return usd_balance, btc_balance
```

8.4.4 Rozhodnutí do jaké pozice vstoupit

Klíčovým vstupním parametrem funkce na ukázce 11 je parametr *prediction*, to je predikce procentuální změny ceny BTC v příští minutě. Pokud je tato predikce větší než 0 a v případě, že bot má nezáporný počet Bitcoinu (tedy před tím již nebyl v dlouhé pozici), bot uzavře předchozí pozici a vstoupí do dlouhé pozice. Pokud je predikce menší než 0 a bot vlastní nezáporný počet Bitcoinů (tedy již není v krátké pozici), vystoupí z předchozí pozice a vstoupí do krátké pozice.

Ukázka 11: trader.py, make_one_trade

```
def make_one_trade(prediction, usd_balance, btc_balance,
                current_btc_price, comission_rate, last_trade, leverage):
    # Opens long position - buys btc
    if prediction > 0 and btc_balance <= 0:
        usd_balance, btc_balance = close_trade(usd_balance, btc_balance,
                                            current_btc_price, comission_rate)
        usd_balance, btc_balance = long_position(usd_balance, btc_balance,
                                              leverage, current_btc_price, comission_rate)
        last_trade = 'long'
    #Opens short position - sells what I dont have -> gets negative btc balance
    elif prediction < 0 and btc_balance >= 0:
        usd_balance, btc_balance = close_trade(usd_balance,btc_balance,
                                            current_btc_price, comission_rate)
        usd_balance, btc_balance = short_position(usd_balance, btc_balance,
                                               leverage,current_btc_price, comission_rate)
        last_trade = 'short'
    else:
        last_trade = 'hold'
    return usd_balance, btc_balance, last_trade
```

8.5 Simulace jakožto celek

Celý backend je spojen v souboru *live_trading_bot.py*. Zde se po zavolání funkce *start_trading*, vytvoří vlákno, ve kterém se vše odehrává. Je zde jedna hlavní velká smyčka, kterou se iteruje, dokud není zavolána funkce *stop_trading*. Procesy zde probíhají velmi podobně jakožto v *back_testeru*, který je k nalezení v kapitole 7 *Zpětné testování*.

Rozdíl je v tom, že data se načítají z jsonu pomocí scriptu *json_data_handler.py*, s nimi se pak dále pracuje jako s dictionary *td*. Další rozdíl tkví v tom, že smyčka neiteruje přes časové sekvence dat, nýbrž iterace je spuštěna každých 60 vteřin.

Sekvence posledních 11 cen BTC je stažena z Binance pomocí Binance API a pak je transformovaná na sekvenci 10 cenových rozdílů. Celý *td* je po každé uložen. Dále na ukázce 12 je mechanismus, jak přerušit simulaci během okamžiku. Kdyby v programu bylo napsáno *time.sleep(60)* tak to, že se simulace přeruší by bylo poznat, až když vyprší časovač 60ti vteřin. Proto je časovač rozdělen do 60 jedno vteřinových, je tak zaručena větší responzivnost, i když je to na úkor časové náročnosti. Dále je na ukázce vidno, že se čas každých 5 vteřin uloží do jsonu.

Ukázka 12: live_trading_bot.py, systém pro rychlé zastavení

```
intervals = 0
            while intervals < 60 and not self._stop_event.is_set():
                intervals += 1
                time.sleep(1)
                if intervals % 5 == 0 :
                    # Calculates time from begging of the oone iteration of trading loop
                    time_spend = round(time.time() - start_time)
                    # Saves the time spend
                    self.save_time_spent(time_spend)
```

9 Propojení backend a frontend

Jak už bylo zmíněno, tak náš program používá k propojení Flask (4.1)

9.1 Lokální server

Pro vyvíjení naší aplikace jsme zapínali náš lokální server s atributem app.run(debug=True), který při editaci kódu se spuštěným serverem zajišťuje automatické restartování serveru pro plynulé testování v průběhu programování a také detailní zprávy o errorech, které nevyhnutelně nastaly. Flask vykresluje statické HTML ze složky "templates"na výchozím lokálním portu http://127.0.0.1:5000. Samotné HTML ale na většinu moderních webových rozhraní nestačí, a tak i zde bylo zapotřebí využít dalších nástrojů jako JavaScript, css a další. Tyto externí soubory pro správné fungování musejí být ve vlastní složce static.

9.2 Žádost a odpověď

Hlavní výhodou používání flasku je relativně jednoduché posílání žádostí z frontendu a přijímání odpovědí.

Ukázka 13: app.py, stop_trading_route

```
from live_trading_bot import start_trading, stop_trading
@app.route('/stop_trading', methods=['POST'])
def stop_trading_route():
    try:
        stop_trading()
    except:
        print("There was nothing to stop")
    return jsonify({'message': 'Trading stopped'})
```

Každá funkce která má být zpracována se musí prvně importovat z python souboru ve kterém se nachází. Dále se u ní nastaví koncový bod (endpoint), který při navštívení dané cesty (route) provede definovanou funkci a vrátí odpověď. Dané cesty mohou obsahovat různé typy HTTP metod, my jsme využívali výchozí metodu GET pro získání informací a POST pro zavolání funkcí.

9.3 Status code

Odpovědi jsou pak vypsány do terminálu serveru, se status kódem. V našem případě terminál konstantně vypisuje požadavky přicházející z frontendu. Pokud se liší od 200 se podle něhož dá řešit chyby v routování. Nejvíce chyb nastávalo při samotném programování, kde je velmi jednoduché udělat překlep v definování jmen do stringu, který IDE nedokáže odhalit. Dále jsme se setkávali se špatným nákladem (payload), což znamená že obsah neodpovídal požadovaným parametrům.

```
127.0.0.1 - - [24/Apr/2024 19:13:14] "GET /load_trading_data HTTP/1.1" 200 -
127.0.0.1 - - [24/Apr/2024 19:13:15] "GET /prepare_hold_array HTTP/1.1" 200 -
127.0.0.1 - - [24/Apr/2024 19:13:15] "GET /get_last_hour_values HTTP/1.1" 200 -
```

10 Frontend Aplikace

V této kapitole jsou vysvětleny zajímavé problémy a nápady, které jsme řešili, abychom dostali co nejvíce responzivní a uživatelsky příjemnou aplikaci.

10.1 Logika frontendu

Všechna funkčnost uživatelského rozhraní je udělán v Javascriptu, který zpravuje zpracování dat a jejich následné zobrazení, aktualizaci stránky, funkčnost tlačítek a dalších prvků. Níže je vysvětleno několik zajímavých funkcí, které v projektu používáme.

10.1.1 Zobrazování aktuálních dat

Pro aktualizaci dat na statické stránce používáme rekurzivní funkci SiteRefresh [kód: 14]. Tato funkce získá potřebné informace k vykreslení aktuálního grafu (updateChart) jako je historie obchodování z uloženého csv souboru (history_array), a hodnoty bitcoinu za poslední hodinu (fetch('/get_last_hour_values'))[obr. 4]. [20, 21, 22]

Také se zeptá pomocí funkce loadData na data jako je třeba aktuální počet peněz, jaký čas se obchoduje, nebo jestli se právě obchoduje; tyto hodnoty jsou uloženy v souboru trade_data.json [kód: 5]. Funkce se dále sama volá pomocí setTimeout s časový odstupem nastavitelným přímo na stránce.

Ukázka 14: logic.js, SiteRefresh

```
(function SiteRefresh() {
  history_array();
  loadData();
  fetch('/get_last_hour_values')
    .then(response => response.json())
    .then(data => {
      btcVal = Object.values(data)[0];
      // Current btc price
      document.getElementById('btc_value').textContent = btcVal[btcVal.length - 1];
      // Last hour diff
      const btcHourDiff = (100 - ((100 / btcVal[btcVal.length - 1]) * btcVal[0])).toFixed(4);
      if (btcHourDiff >= 0) {
        document.getElementById('btcHourDiff').textContent = " +" + btcHourDiff;
      } else {
        document.getElementById('btcHourDiff').textContent = btcHourDiff;
      }
      // Chart
      updateChart();
      resetTimePassedInterval();
      startTime = Date.now(); // Reset the start time
      setTimeout(SiteRefresh, (refreshTime * 1000));
    });
})();
```

Obrázek 4: Graf: obchodování bota a koupit & držet, pouze bitcoin

10.1.2 Překlad

Tato funkce umožňuje překlad stránky z anglického jazyka do českého a zpátky. Po nastavení správného jazyka se pomocí JQuerry naleznou všechny elementy v HTML souboru s atributem data-translation a podle souboru translations.json [kód: 16] souboru se každý prvek přeloží. [23]

Funkce displayToggleBtn, displayBtcBtn, updateChart se musí řešit zvlášť, kvůli jejich dynamickému textu nastaveném v jiných funkcích Javascriptu; tyto funkce nastaví text sami podle zvoleného jazyku a aktualizují jej na stránce. Pokud je zvolen například český jazyk tak text vedle tlačítka je schválně v angličtině, aby lidé nemluvící česky mohli najít tlačítko podle svého jazyka.

Ukázka 15: logic.js, switchLanguage

```
482 //Language
483 function switchLanguage() {
484   var currentLang = document.documentElement.lang;
485   var targetLang = currentLang === "en" ? "cz" : "en";
486   currentLanguage = targetLang;
487
488   displayToggleBtn();
489   displayBtcBtn();
490   updateChart();
491
492   fetch('/static/translations.json')
493     .then(response => response.json())
494     .then(data => {
495       // Get all elements with data-translation attribute
496       var elements = document.querySelectorAll('[data-translation]');
497
498       // Update content of each element with the corresponding translation
499       elements.forEach(element => {
500         var translationKey = element.getAttribute('data-translation');
501         element.textContent = data[targetLang][translationKey];
502       });
503
504       // Set the document language to the target language
505       document.documentElement.lang = targetLang;
506     })
507     .catch(error => console.error('Error fetching translations:', error));
508 }
```

Ukázka 16: translations.json

```
{
    "en": {
        "tmoney": "Money",
        "teval": "Evaluation",
        "tsettings": "Settings",
        "tlinks": "Links",
    },
    "cz": {
        "tmoney": "Penize",
        "teval": "Evaluace",
        "tsettings": "Nastaveni",
        "tlinks": "Odkazy",
    }
}
```

10.1.3 Barevná témata

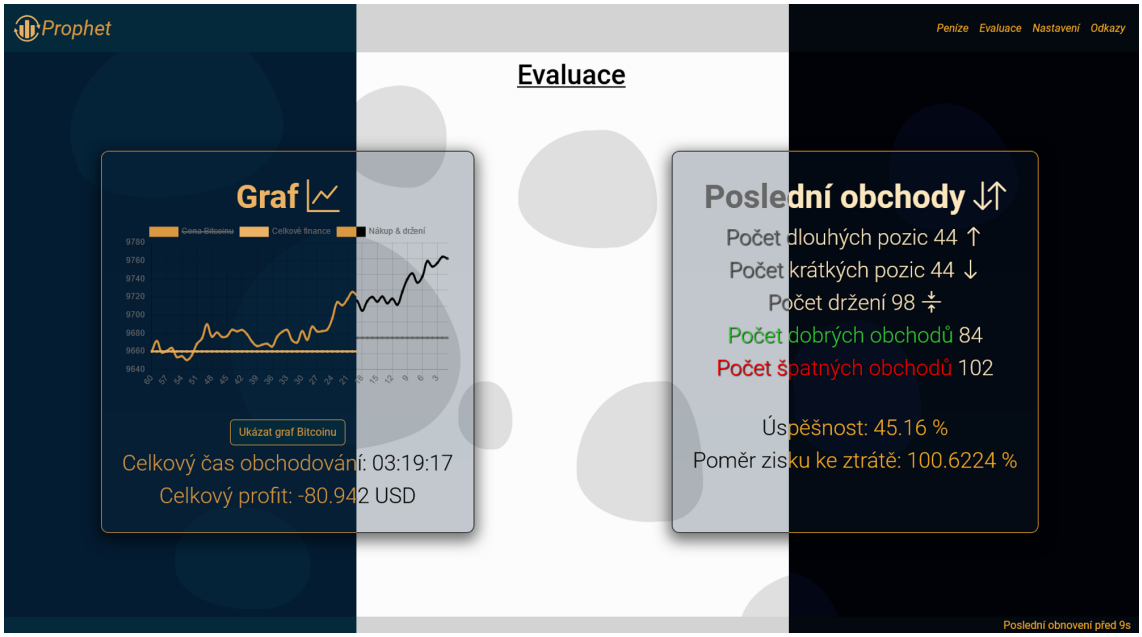
Naše stránka obsahuje také různé předem definovaná barevná témata, které si může uživatel přepínat v přímo ze stránky. V ukázce jednoho ze tří módů; funkce darkMode [kód: 17] nastavuje Javascriptem proměnné v :root souboru css [kód: 18], které se v něm dále používají, pomocí style.setProperty. Tento přístup zajišťuje princip DRY (do not repeat yourself) a také to, že barvy nejsou ve vzájemném rozporu (viz obr. 5).[24] Při přepínaní barevných režimů se také mění favikona stránky, aby kontrastovala s pozadím. Na poslední řádce se ještě zavolá funkce pro znovu vykreslení grafu s novými barvami. [25, 26, 27]

Ukázka 17: logic.js, darkMode

```
function darkMode() {
  document.documentElement.style.setProperty('--darkBlue', '#010207')
  document.documentElement.style.setProperty('--darkerBlue', '#000000bd')
  document.documentElement.style.setProperty('--yellow', '#fce6bd',)
  document.documentElement.style.setProperty('--orange', '#feb025',)
  document.documentElement.style.setProperty('--darkCyan', '#04090f',)
  document.documentElement.style.setProperty('--blobColor', '#04081a')
  document.getElementsByClassName("icon")[0].src = "static/img/Logo.png";
  updateChart();
}
```

Ukázka 18: style.css, :root

```
:root { --darkBlue: #041c32;
        --darkerBlue: #021a31bd;
        --yellow: #ecb365;
        --orange: #feb025;
        --darkCyan: #04293a;
        --blobColor: #06314576;}
```



Obrázek 5: Barevná témata stránky

10.2 Struktura uživatelského rozhraní

Hlavní struktura frontendové stránky tvoří statické HTML, které samo o sobě pouze zobrazuje základní strukturu. [28]

10.2.1 Hlavní rozložení sekcí

Stránka tvořena sekcemi o dvou sloupcích, ty jsou tvořeny v HTML pomocí class=”row”a class=”col”. Tento způsob organizace zabezpečuje responzivnost a ikdyž nebyl na mobil původně zamýšlen, tak i na menších obrazovkách s různými rozlišeními a velikostmi je zcela čitelný. Samotná data se zobrazují v containerech span s příslušným id, kterým hodnoty poskytuje JavaScript. Dále je máme vždy na horní straně ”přilepený”navigation bar pro lepší a rychlejší orientaci a na dolní proužek s aktuálním časem od posledního obnovení stránky.

Ukázka 19: base.html, struktura

```
<div class="container-fluid separator">
    <h1 class="header" id="home">
        <span data-translation="tmoney">Penize </span>
    </h1>
    <div class="row">
      <div class="col-md-6 order-1 order-sm-1 d-flex  justify-content-center">
        <div class="card d-flex flex-column align-items-center">
          <h2 class="miniHeader">
            <span data-translation="twallet">Penezenka </span>
            <i class="bi bi-wallet2"> </i>
          </h2>
          <p class="numbers">
            <span id="USD_balance"> </span> USD <br>
            <span id="BTC_balance"> </span> BTC
          </p>
        </div>
      </div>
      <div class="col-md-6 order-2 order-sm-2 d-flex justify-content-center">
        <div class="bg3 d-flex flex-column align-items-center">
          <h2 class="miniHeader">Bitcoin
            <i class="bi bi-currency-bitcoin"> </i>
          </h2>
          <p class="numbers">
            <span data-translation="tPriceBTC">Nynejsi cena </span>
                <span id="btc_value"> </span> USD <br>
            <span data-translation="tLastHourBTC"> Posledni hodina </span>
                <span id="btcHourDiff"> </span>% <br>
          </p>
        </div>
      </div>
    </div>
  </div>
```

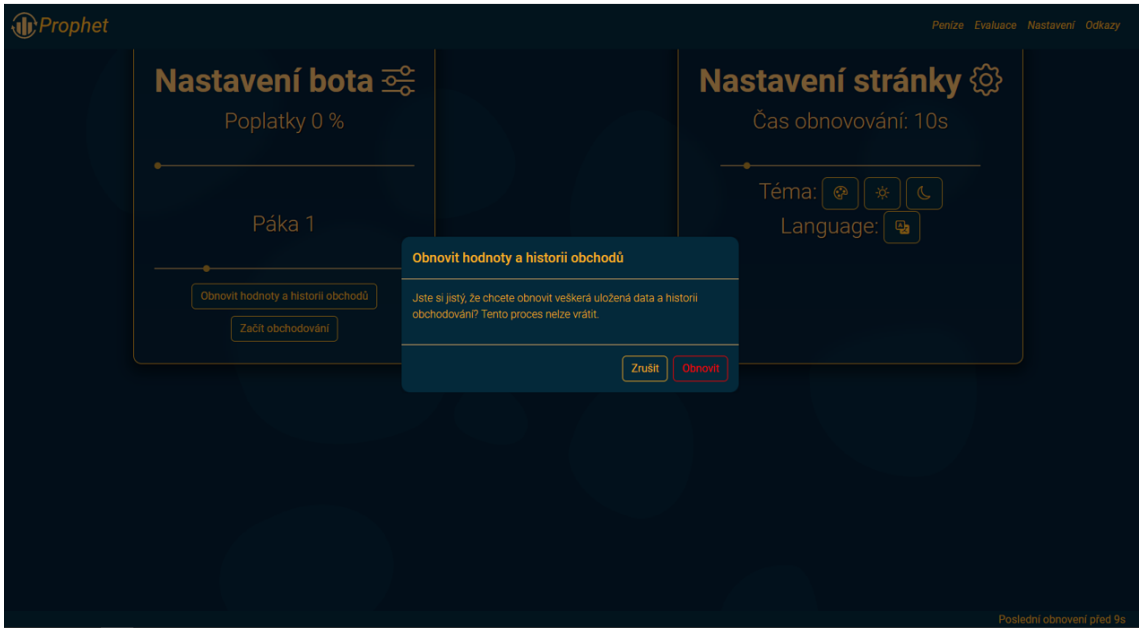
10.2.2 Bootstrap

Bootstrap je front-endový framework pro vývoj webových stránek a aplikací. Poskytuje nástroje pro rychlé a efektivní vytváření responzivního designu. Obsahuje velkou řadu již předdefinovaných komponentů, které se jednoduše používají díky dobré dokumentaci s vysvětlením a příklady na stránce Bootstrapu. My jsme jeho funkce hojně využívali, například pro tzv. modal, navbar, slider a další. [29] [30]

Níže je kód takzvaného modalu, ten slouží jako zpětná vazba pro uživatele; v našem případě byl využit jako potvrzení resetu dat a historie. Bootstrap má modal na své stránce dobře zdokumentovaný, tak pro jeho implementaci stačí zkopírovat kód a následně jej editovat podle vlastní potřeby. My jsme modal upravili následujícím způsobem: Samotný modal se zobrazí uprostřed stránky pomocí modal-dialog-centered. Veškerý text má atribut data-translation. Modal-header, content a footer upravuje css do našich barev, a v závěru je nastaveno tlačítko s funkcí resetSavedData. [31]

Ukázka 20: base.html, Bootstrap modal

```
<div class="modal␣fade" id="exampleModal" tabindex="-1"
 aria-labelledby="exampleModalLabel" aria-hidden="true" >
   <div class="modal-dialog␣modal-dialog-centered">
     <div class="modal-content">
       <div class="␣modal-header␣">
         <h1 class="modal-title␣fs-5␣" id="exampleModalLabel">
           <span data-translation="tReset"> Obnovit hodnoty a historii obchodu
           </span>
          </h1>
       </div>
       <div class="modal-body">
           <span data-translation="tModal">Jste si jisty, ... proces nelze vratit.
           </span>
       </div>
       <div class="modal-footer">
         <button type="button" class="btn␣btn-secondary" data-bs-dismiss="modal">
          <span data-translation="tCancel">Zrusit </span> </button>
           <button type="button" class="btn␣btn-primary"
            style="border-color:␣red;␣color:␣red;"
            onclick=resetSavedData() data-bs-dismiss="modal" >
         <span data-translation="tResetYes"> Obnovit</span> </button>
       </div>
     </div>
   </div>
 </div>
```



Obrázek 6: Screenshot Modal na stránce

10.3 Grafický design

Naše aplikace pro lepší uživatelská zkušenost používá kaskádové styly (CSS). Kromě těch také máme využíváme pro čitelnější stránku Bootstrap Icons a font Roboto od Googlu. [32, 33]

10.3.1 Dynamické pozadí

Pozadí našeho projektu je vytvořeno pomocí tzv. “blobů”. “Blobové” jsou kulaté tvary, které mění svůj tvar na základě animací. V kódu dole používáme několik atributů s různými hodnotami, které určují barvu blobů, výšku, šířku, pozici, rádius ohraničení, animaci pomocí pravidla @keyframes, o kterých je napsáno níže. Také zde máme vlastnost opacity, která dělá průhlednost tvaru a na konci je vlastnost z-index, která určuje ”úroveň vrstvy”prvku v rámci vrstvení na ose ”z”.

Ukázka 21: style.css, blob

```
48 .shape-blob { /* blobs attribute start */
49     background-color:var(--blobColor);
50     height: 60px;
51     width: 80px;
52     border-radius: 40% 50% 30% 40%;
53     animation:
54         transform 18s ease-in-out infinite both alternate,
55         movement_one 12s ease-in-out infinite both;
56     opacity:.7;
57     position: fixed;
58     z-index: -1;
59 }
```

K vytvoření různorodosti blobů používame deset téměř identických tříd, jenž mají podobné atributy. Mění se zde pouze lišící se hodnoty, až na barvu - ta je všude stejná. Na konci máme funkce transform a animation, které dělají samotný pohyb blobů. Zadává se zde velikost otáčení ve stupních a čas transformování.

Ukázka 22: style.css, blob one

```
61 .shape-blob.one{
62     background-color:var(--blobColor);
63     height: 200px;
64     width: 250px;
65     left: 1%;
66     top: 3%;
67     transform: rotate(-180deg);
68     animation:
69         transform 8s ease-in-out infinite both alternate,
70         movement_two 20s ease-in-out infinite both;
71 }
```

Dole najdeme třídy, ve kterých se děje celá animace. Animace transform definuje změny vlastnosti border-radius v různých časech během animace, což umožňuje přechod tvaru hran v čase. Animace začíná na 0% a končí na 100%, a v průběhu těchto hodnot se mění border-radius. Každá skupina čísel v border-radius definuje zaoblení rohů, a hodnoty před a za lomítkem odpovídají svislým a vodorovným osám zaoblení.

- 0%, 100%: Obě hodnoty jsou stejné, což znamená, že animace se uzavírá v cyklu. Tvar má poměrně nepravidelný vzhled s různým zaoblením.

- 20%: Tento stav má více asymetrické zaoblení než původní a konečný tvar. Zaoblení se mění, přidávají se a ubírají na různých stranách.

- 40%: Opět další změna ve tvaru. Objevuje se zde výraznější asymetrie s velmi odlišnými hodnotami.

- 60%: Tento tvar vrací část původního zaoblení, ale má jiné změny ve srovnání s 20% a 40%.

- 80%: Tento tvar má mírně vyváženější vzhled s výrazným změněním zaoblení v obou osách.

Na ukázce 23 pod animací transform najdeme dvě movement animace. Obě animace zahrnují změny pozice a rotace v různých časech během animace. První animace (movement_one) má jednoduchý posun a mírnou rotaci kolem osy Y, zatímco druhá animace (movement_two) má výraznější rotaci kolem osy Z a zvětšení velikosti prvku. Obě animace se vracejí do původního stavu na začátku a konci animace. [34]

Ukázka 23: style.css, Keyframes

```
@keyframes transform /* blobs transform */
{
    0%,
  100% { border-radius: 33% 67% 70% 30% / 30% 40% 70% 70%; }
   20% { border-radius: 37% 63% 51% 49% / 37% 35% 35% 63%; }
   40% { border-radius: 36% 64% 64% 36% / 64% 48% 52% 26%; }
   60% { border-radius: 37% 63% 51% 49% / 30% 30% 70% 73%; }
   80% { border-radius: 40% 60% 42% 58% / 51% 51% 49% 59%; }
}

@keyframes movement_one /* blobs movement */
{
    0%,
  100% { transform: none; }
   50% { transform: translate(50%, 20%) rotateY(10deg) scale(1); }
}

@keyframes movement_two
{
    0%,
  500% { transform: none; }
   50% { transform: translate(50%, 20%) rotate(-200deg) scale(1.3);}
}
```

10.3.2 Karty

Text hlavních sekcí našeho projektu, je v kartách; tyto karty mají několik vlastností. Ty esenciální jsou backdrop-filter/-webkit-backdrop-filter, který má atribut blur, který rozmazává pozadí, také background, kde je atribut linear-gradient, který přelévá dvě barvy od vršku pod spodek. Poté je zde vlastnost box-shadow, která dává kartě stín okolo a hodnoty px, které zadávají, popořadě, vertikální posun stínu, rozmazání stínu a šíření stínu neboli jak daleko má stín jít od okrajů, a hex kód, který zadává barvu a poslední dvě číslice zadávají průhlednost. [35, 36]

Ukázka 24: style.css, Karty

```
33 .card {
34     user-select:none;
35   margin: 5rem auto;
36   padding: 30px;
37   border: 1px solid var(--orange);
38   background-color: #021a3139;
39   background: linear-gradient(0deg, #0c273f3d 0%, #041c323d 100%);
40   box-shadow: 0 7px 20px 5px #00000088;
41   border-radius: .7rem;
42   backdrop-filter: blur(7px);
43   -webkit-backdrop-filter: blur(7px);
44   overflow: hidden;
45   transition: .5s all;
46 }
```

10.3.3 Animace podtrhnutí

Tato část CSS kódu vytváří efekt animovaného podtržení pro prvky v navbaru. Když uživatel najede myší na prvek s třídou underL, podtržení se zobrazí s hladkým přechodem a zvýrazní daný prvek. Prvky s třídou underL jsou stylizovány tak, aby byly relativně umístěné, což umožňuje použít element ::after pro přidání podtržení. Element ::after je umístěn absolutně, což mu umožňuje se přesně pozicovat pod textem. Tento element nemá žádný obsah, ale slouží jako vizuální prvek pro podtržení. Podtržení je oranžové, má výšku 3 pixely a je umístěno dole s počáteční šířkou 0, tím pádem je neviditelné. Když uživatel najede myší na prvek s třídou underL, barva textu zůstává oranžová, ale podtržení se hladce rozšiřuje na celou šířku prvku. Tento přechod trvá 0,3 sekundy, což vytváří plynulý efekt. Navíc změna velikosti zachovává výšku podtržení, zatímco jeho šířka se rozšiřuje na plnou délku. [37]

11 Závěr

Využití neurální sítě na rozklíčování chaotických finančních dat Bitcoinu se do jisté míry ukázalo jako možné. Ale realizace systému automatického obchodování, založeném čistě na principu vyhodnocování časové posloupnosti finančních dat, se kvůli poplatkům ukázalo jako nemožné.

Co se týče vizuální stránky, myslíme si že se nám velice vydařila a disponuje velkou škálou funkcí, které jsou nadrámcové. Jeden z mála problémů, které se nejspíše měli promyslet, bylo nepoužívat statickou HTML stránku pro zobrazování měnících se dat v čase, pro něž se spíše hodí dynamický framework jako vue.js, angular nebo react, které jsou pro podobné projekty vhodnější.

I přes nedosažení vysoce ambiciózního cíle považujeme projekt za vysoce úspěšný. Na projektu by se dále dalo pokračovat přidáváním dalších funkcí a optimalizováním.

12 Použité zdroje

[1] *ATC Market - Co je to Bitcoin?* URL: https://www.atcmarket.cz/articles/24733.

[2] Kriptomat. *Co je Bitcoin a jak funguje? Kdo vytvořil BTC?* URL: https://kriptomat.io/cs/kryptomeny/bitcoin/co-je-to-bitcoin/.

[3] *Flask*. 7. dub. 2024. URL: https://pypi.org/project/Flask/.

[4] *Welcome to Flask — Flask Documentation (3.0.X)*. URL: https://flask.palletsprojects.com/en/3.0.x/.

[5] *NumPy* -. URL: https://numpy.org/.

[6] *pandas - Python Data Analysis Library*. URL: https://pandas.pydata.org/.

[7] J. D. Hunter. „Matplotlib: A 2D graphics environment“. In: *Computing in Science & Engineering* 9.3 (2007), s. 90–95. DOI: 10.1109/MCSE.2007.55.

[8] *Matplotlib — Visualization with Python*. URL: https://matplotlib.org/.

[9] *PyTorch*. URL: https://pytorch.org/.

[10] Fred Oh. *What is CUDA — NVIDIA Official Blog*. 27. led. 2022. URL: https://blogs.nvidia.com/blog/what-is-cuda-2/.

[11] *Buy/Sell Bitcoin, Ether and Altcoins — Cryptocurrency Exchange — Binance*. URL: https://www.binance.com/en/binance-api.

[12] Greg Hogg. *Amazon Stock Forecasting in PyTorch with LSTM Neural Network (Time Series Forecasting) — Tutorial 3*. 8. dub. 2023. URL: https://www.youtube.com/watch?v=q_HS4s1L8UI.

[13] *LSTM — PyTorch 2.3 documentation*. URL: https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html.

[14] Christopher Olah. *Understanding LSTM Networks – colah's blog*. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[15] James Chen. *Backtesting: definition, how it works, and downsides*. 11. dub. 2024. URL: https://www.investopedia.com/terms/b/backtesting.asp.

[16] *How to update json file with python*. URL: https://stackoverflow.com/questions/13949637/how-to-update-json-file-with-python.

[17] *How to append a new row to an old CSV file in Python?* URL: https://stackoverflow.com/questions/2363731/how-to-append-a-new-row-to-an-old-csv-file-in-python.

[18] Adam Hayes. *Long Position: Definition, types, example, pros and cons.* 6. říj. 2021. URL: https://www.investopedia.com/terms/l/long.asp.

[19] Adam Hayes. *Short Selling: pros, cons, and examples.* 21. břez. 2024. URL: https://www.investopedia.com/terms/s/shortselling.asp.

[20] *StackOverflow - chart hiden datapoints.* URL: https://stackoverflow.com/questions/59972363/chartjs-how-to-set-a-data-point-to-be-hidden-by-default.

[21] *Chart JS.* URL: https://www.chartjs.org/docs/latest/axes/.

[22] *StackOverflow - hide points on chart.* URL: https://stackoverflow.com/questions/35073734/hide-points-in-chartjs-linegraph.

[23] *StackOverflow - multiple language.* URL: https://stackoverflow.com/questions/46008760/how-to-build-multiple-language-website-using-pure-html-js-jquery.

[24] *Color palette: 041C32 04293A 064663 ECB365 - Color Hunt.* URL: https://colorhunt.co/palette/041c3204293a064663ecb365.

[25] Alvaro Trigo. *How To Change CSS With JavaScript [With Examples] - Alvaro Trigo Blog.* 7. ún. 2024. URL: https://alvarotrigo.com/blog/change-css-javascript/#5.-overwrite-css-!important-style-with-javascript.

[26] Per Harald Borgen. *How to easily create themes with CSS Variables.* 24. zář. 2019. URL: https://www.freecodecamp.org/news/how-to-easily-create-themes-with-css-variables-2d0f4cfa5b9a/.

[27] *ChatGPT - Change image with JavaScript.* URL: https://chat.openai.com/share/005545c6-656d-4c3b-96e4-c5b130135dc8.

[28] *W3School HTML.* URL: https://www.w3schools.com/html.

[29] *Bootstrap.* URL: https://icons.getbootstrap.com/.

[30] *W3School slider.* URL: https://www.w3schools.com/howto/howto_js_rangeslider.asp.

[31] Craig Campbell. *Modal colors*. Květ. 2016. URL: https://webdesign.tutsplus.com/customizing-bootstrap-components--CRS-200513c/modal-colors.

[32] *Bootstrap Icons*. URL: https://icons.getbootstrap.com/.

[33] *Fonts Google*. URL: https://fonts.google.com/.

[34] *30 CSS blob effects*. Dub. 2022. URL: https://freefrontend.com/css-blob-effects/.

[35] *W3School shadow*. URL: https://www.w3schools.com/css/css_text_shadow.asp.

[36] *NFT Card component*. URL: https://codepen.io/kiberbash/pen/MWEpevg.

[37] Coding Artist. *Draw Underline Link Hover Effect — CSS Menu Hover Effect*. Led. 2021. URL: https://www.youtube.com/watch?v=aswRKAjjWuE.

Seznam obrázků

Seznam ukázek kódu