|  |  |
| --- | --- |
| A képen embléma, szimbólum, címerpajzs, jelvény látható  Automatikusan generált leírás | EÖTVÖS LORÁND TUDOMÁNYEGYETEM  Informatikai Kar  Média- és Oktatásinformatika Tanszék |

Kriptovaluták árfolyamának elemzése és korrelációk keresése SAP –

Capireban

|  |  |
| --- | --- |
| Témavezető: Dr Vincellér Zoltán  egyetemi mesteroktató | Szerző: Téglás Tamás  programtervező informatikus BSc. |

Budapest, 2023

Tartalomjegyzék

[Bevezetés 6](#_Toc136076296)

[Felhasználói dokumentáció 7](#_Toc136076297)

[Szoftver telepítése és futtatása 7](#_Toc136076298)

[Konzol használata 7](#_Toc136076299)

[Webes felület használata 8](#_Toc136076300)

[Home 8](#_Toc136076301)

[Charts 8](#_Toc136076302)

[Custom Charts 8](#_Toc136076303)

[Bookmarked Charts 9](#_Toc136076304)

[Hidden Charts 9](#_Toc136076305)

[Grafikonok kinézete, típusai 9](#_Toc136076306)

[Chart operations 10](#_Toc136076307)

[Create UI5 11](#_Toc136076308)

[Create Chart-js 12](#_Toc136076309)

[Update or Delete 12](#_Toc136076310)

[Run script 13](#_Toc136076311)

[Fejlesztői dokumentáció 16](#_Toc136076312)

[Architektúra 16](#_Toc136076313)

[Backend kialakítása, és működése 17](#_Toc136076314)

[Backend technológai oldala – Core Data Services 17](#_Toc136076315)

[Adatmodellek 19](#_Toc136076316)

[Szolgáltatások 24](#_Toc136076317)

[Szoftvert támogató szkriptek működése 27](#_Toc136076318)

[add\_charts.py 28](#_Toc136076319)

[add\_data.py 28](#_Toc136076320)

[analysis.py 30](#_Toc136076321)

[monthly\_charts.py 31](#_Toc136076322)

[Frontend kialakítása és működése 32](#_Toc136076323)

[Routes 33](#_Toc136076324)

[Layout 36](#_Toc136076325)

[Basic Components 37](#_Toc136076326)

**EÖTVÖS LORÁND TUDOMÁNYEGYETEM**

**INFORMATIKAI KAR**

**SZAKDOLGOZAT TÉMABEJELENTŐ**

**Hallgató adatai:**

**Név:** Téglás Tamás

**Neptun kód:** BQMR6I

**Képzési adatok:**

**Szak:** programtervező informatikus, alapképzés (BA/BSc/BProf)

**Tagozat** : Nappali

Belső témavezetővel rendelkezem

***Témavezető neve:*** *dr. Vincellér Zoltán*

*munkahelyének neve, tanszéke:* ***ELTE IK, Információs rendszerek Tanszék***

*munkahelyének címe:* ***1117, Budapest, Pázmány Péter sétány 1/C.***

*beosztás és iskolai végzettsége:* ***egyetemi mesteroktató***

**A szakdolgozat címe:** Kriptovaluták árfolyamának elemzése és korrelációk keresése SAP –

Capireban

**A szakdolgozat témája:**

*(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben szakdolgozat témájának leírását )*

**A legtöbb kriptovalutákban érdeklődő ember számára bonyolult lehet korszerű, és átlátható tudást szerezni, amely ingyenesen elérhető,**

**vagy nem kell hozzá bármiféle gazdasági háttérrel rendelkeznie. Sok esetben az oldalak melyek átfogó információt nyújtanak egy-egy**

**kriptovalutáról, nem térnek ki, vagy nem nyújtanak olyan funkciókat mellyel többet egyszerre össze lehetne hasonlítani. Gyakran a**

**zöldfülűeknek csak néhány lényegre törő grafikon és ábra kell mely kompakt módon mindent elárul kezdő szinten a kriptovaluta jelenlegi**

**helyzetéről.**

**Szakdolgozatom témája az efféle problémákra próbál megoldást nyújtani, melyben a legtöbb kriptovalutáról valós időben gyors átfogó,**

**egyszerűen olvasható, és személyre szabható grafikonokat és ábrákat biztosít minden felhasználó számára. A felhasználó válogathat az**

**elérhető kriptovaluták árfolyamának megtekintése között és az előre definiált ábrák közt, melyek egy komplexebb képet nyújtanak a piac**

**mélyebb megértéséhez. A felhasználók képesek lesznek a kedvenc árfolyamaikat és ábráikat könyvjelzőzni, mely gyorsabb elérést biztosít**

**azoknak, akik már türelmetlenül várják, hogy befektethessenek. A profiknak testreszabható grafikonokat és ábrákat nyújtunk melyeken**

**különböző valuták korrelációját vizsgálhatják a múltba tekintve. Akik pedig a jövőbe szeretnének látni, egyedi szolgáltatásunk képes**

**bizonyos esetekben megjósolni egy kriptovaluta árfolyamának változását különböző paraméterek figyelembevételével.**

**A fejlesztéshez SAP Capire és CDS technológiákat használok, mellyel egy full-stack webalkalmazást készítek, mely adattároláshoz SAP**

**Hana adatbázist fogok készíteni.**

Budapest, 2022. 11. 30.

# Bevezetés

A mai világban minden nap találkozhatunk kriptovalutákkal különböző szentimentalitású hírekben. Szemünkbe tűnhetnek dollármilliárdos csalások, átverések, de néha megjelennek interjúk olyan emberekkel, akik hatalmas vagyonra tettek szert, mert 10-15 éve vásároltak fillérekért egy teljesen ismeretlen kriptovalutát, ami mai napra többet ér, mint az arany. Az átlagember nem tudja nyomon követni, hogy melyik kriptovaluta mennyit ér a millió közül, a legtöbb pénzügyi weboldal pedig túl sok információt nyújt.

A képen szöveg, képernyőkép, diagram, Diagram látható

Automatikusan generált leírás

1. *ábra: A Yahoo! Finance oldalnak egy grafikonja*

Az első ábrán láthatjuk, hogy a kriptovaluta árfolyam értéke pontosan milyen módon változott napról napra, mekkora értékeket ért el, és hol nyitott/zárt a piac. A gyertyagrafikonok tudják a legjobban reprezentálni egy árfolyam változását, de a legtöbb ember számára, túlságosan bonyolultak. Ha pedig szeretnénk egy második kriptovalutát elhelyezni a grafikonon, sokszor csak hosszú keresés után tudunk megfelelő oldalt találni, mely ezt megengedi nekünk előfizetés nélkül. A célom ezek alapján egy alkalmazást nyújtani a felhasználók részére, mely egy letisztult webes felületen keresztül lehetővé teszi, hogy minden kriptovaluta iránt érdeklődő személy könnyen hozzáférhessen személyre szabott grafikonokhoz, melyek csak a felhasználó által kívánt információt jeleníti meg.

# Felhasználói dokumentáció

## Szoftver telepítése és futtatása

A szoftver telepítéséhez és használatához a további előfeltételek szükségesek:

* npm (Node Package manager, 9.6.1)
* node.js (^16.15)
* cds-dk (Core Data Services Development Kit 6.8.1)
* python (3.9.2)
* Windows operációs rendszer
* Naprakész böngésző

A szoftver telepítéséhez és futtatásához létrehoztam PowerShell szkripteket, melyek megkönnyítik ezeket a folyamatokat. Ezeket a projekt gyökérkönyvtárában megtalálhatóak.

* *deploy.ps1* – Telepíti a megfelelő csomagokat, frissíti a kezelőfelületet, és létrehozza az adatbázist alap adatokkal.
* *run.ps1* – Elindítja az alkalmazást.
* *deploy\_and\_run.ps1* – Telepíti a megfelelő csomagokat, frissíti a kezelőfelületet, és létrehozza az adatbázist alap adatokkal, majd elindítja az alkalmazást.

Indítás után az alkalmazást elérhetővé válik a [*https://localhost:4004*](https://localhost:4004)címen. Ezt a címet a böngésző URL sávjába írva, a felhasználó elérheti a webes kezelőfelületet.

## Konzol használata

Az alkalmazást egy terminálból, vagyis egy konzolból kell indítani, mely Windows operációs rendszeren a *Powershell* lehet, vagy újabb verziókon a *Windows Terminál,* mely ugyanúgy megfelel a célnak. Ezen a konzolon, indítás után láthatunk üzeneteket, logokat, melyet az alkalmazás futása során kiír. Láthatjuk az összes végpont kérést, ami érkezik, illetve a backend függvényeinek üzenetét is egyaránt. A *python* szkriptek melyeket a felhasználó indít, nem csak az adatbázisba töltik a futási eredményüket, hanem több informatív üzenetet is kiírnak ide futásuk során. *(2. ábra)*

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

*2. ábra*

## A képen szöveg, képernyőkép, Betűtípus, szám látható Automatikusan generált leírásWebes felület használata

Az alkalmazáson belüli navigálásra, a felhasználó számára elérhető az oldal bal oldalán egy sávban, a különböző menüpontok. *(3. ábra)*

A menü fejlécére kattintva minimalizálódik és csak az ikonok látszódnak, míg az almenük felugró ablakokban jelennek meg.

### Home

Az alkalmazás kezdőoldala a *Home* menüpontnak felel meg, és az alkalmazás címére navigálva, ezzel az oldallal találkozik a felhasználó először.

*3. ábra*

Itt találhatóak az alkalmazás által automatikusan generált grafikonok, az adatbázisban található kriptovaluták havi zárási értékeiről.

### Charts

Ebben a menüpontban találhatóak a felhasználó által készített grafikonok megjelenítési almenüi, ahol megtekinthetjük a könyvjelzőzött, az elrejtett, és az összes grafikont, egy bizonyos rendezés keretein belül.

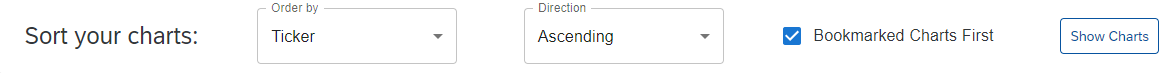
#### Custom Charts

Erre a menüpontra való kattintás után megjelennek a felhasználó készített grafikonok alap sorrend szerint: az elsődleges adathalmaz szimbóluma ábécé sorrendben, a könyvjelzőzött grafikonokkal kezdve.

Itt meg lehet adni, hogy mi más alapján rendezzük a grafikonokat, milyen irányba, illetve, hogy a könyvjelzőzött grafikonokat, soron kívül külön rendezve előrébb jelenjenek meg, mint a többi. *(4. ábra)*

Amikor a rendezési szempontok közül valamelyiket módosítjuk, a grafikonok eltűnnek, de a *Show Charts* gomb megnyomása után megjelennek.

A rejtett grafikonok itt nem jelennek meg.



*4. ábra*

*Order By – Rendezési szempont*

*Direction – Rendezési irány*

*Bookmarked Charts First – Könyvjelzőzött grafikonok előre vétele*

*Show Charts – Grafikonok megjelenítése a kiválasztott rendezés alapján*

#### Bookmarked Charts

Ebben a menüpontban találhatóak azok a grafikonok melyeket a felhasználó könyvjelzőzött a gyorsabb elérés kedvéért.

A rejtett grafikonok itt nem jelennek meg.

#### Hidden Charts

Ebben a menüpontban találhatóak a grafikonok melyek rejtettként vannak megjelölve, más menüpontokban nem jelennek meg az itt található grafikonok.

#### Grafikonok kinézete, típusai

A program kétfajta grafikont kezel, egy szimplát a *UI5* *(5. ábra)* csomagból, és egy komplexet a *Chart-js* *(6. ábra)* csomagból. Ezekre a típusokra mostantól a csomagjaik alapján fogunk hivatkozni. A *UI5* grafikonok csak egy adathalmazt képesek megjeleníteni, viszont rajtuk találhatóak különböző előrejelzési adatok. A *Chart-js* grafikonok általában kettő adathalmazt jelenítenek meg, különböző színekkel és különböző léptékekkel. Rajtuk nem lehet előrejelzéseket megjeleníteni.

A képen szöveg, sor, kézírás, Diagram látható

Automatikusan generált leírás

*5. ábra*

A képen szöveg, Diagram, sor, diagram látható

Automatikusan generált leírás

*6. ábra*

Mindkét grafikon alján található 2 gomb: *(7.ábra)*

*7. ábra*

* Hide Chart / Unhide Chart
* Bookmark / Remove Bookmark

Itt az előbbi gomb felel a grafikon elrejtéséért, rákattintva vagy elrejti a program a grafikont, és átnavigál a *Hidden Charts* menüpontra, vagy visszavonja a rejtett státuszát a grafikonnak, és átnavigál a *Custom Charts* menüpontra.

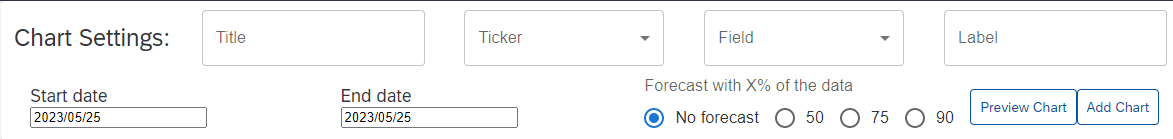
Hasonlóan, a másik gomb könyvjelzőzi a grafikont, és átnavigál a *Bookmarked Charts* menüpontra, vagy leveszi a könvjelzőt a grafikonról és átnavigál a *Custom Charts* menüpontra.

### Chart operations

Ez a menüpont almenüi tartalmazzák a grafikonok létrehozásához, törléséhez vagy módosításához használható felületeket.

#### Create UI5

Ezen a felületen találhatóak a *UI5* grafikonok létrehozásához szükséges beviteli mezők. *(8. ábra)*

**

*8. ábra*

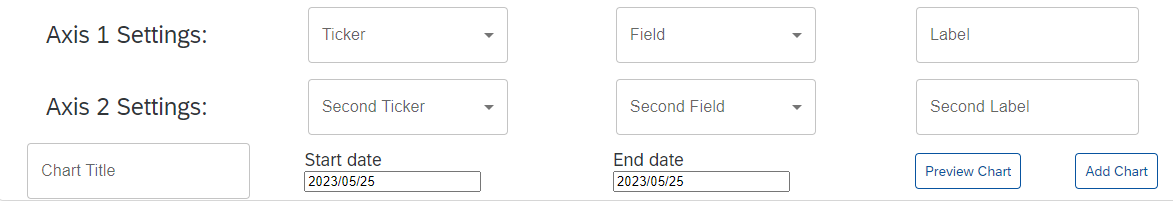
* *Title*: A grafikon címe
* *Ticker*: A kriptovaluta szimbóluma
* *Field*: Egy a kriptovaluta különböző értékei közül:
  + *Open*: A kriptovaluta árfolyama tőzsde nyitáskor
  + *High*: A kriptovaluta árfolyamának aznapi legmagasabb értéke
  + *Low*: A kriptovaluta árfolyamának aznapi legalacsonyabb értéke
  + *Close*: A kriptovaluta árfolyamának aznapi zárási eredménye
  + *Adjusted Close*: A kriptovalutáknál ritka, hogy ez az érték más legyen, mint a *Close* érték, ám lehetséges, ha *forkolják* a kriptovalutát, vagyis új verziót kezdenek használni belőle, ami nem kompatibilis az előző tranzakciókkal
  + *Volume*: Az aznapi összes tranzakciókban szerepelt kriptovaluta mennyisége
* *Label*: Az adathalmaz címkéje
* *Start date*: A grafikon kezdeti dátuma
* *End date*: A grafikon befejezési dátuma
* *Forecast with X% of the data*: Ebben a mezőben tudja a felhasználó kiválasztani, milyen típusú előrejelzést szeretne látni a grafikonjában. Minden előrejelzés extra 60 értéket biztosít a *Close* mező kiválasztásakor.
  + *No forecast*: Nincs előrejelzés
  + *50*: A rendelkezésre álló adatok második feléből generált előrejelzéseket mutatja.
  + *25*: A rendelkezésre álló adatok negyedik negyedéből generált előrejelzéseket mutatja.
  + *10*: A rendelkezésre álló adatok tizedik tizedéből generált előrejelzéseket mutatja.

A *Preview Chart* gombra kattintva megjelenik a grafikon előnézete. Amennyiben valamelyik beviteli mezőt módosítjuk, az előnézet eltűnik és újra erre a gombra kell kattintani, hogy megjelenjen.

Az *Add Chart* gomb megnyomására a beviteli adatok alapján létrejön egy *UI5* grafikon, és a program átnavigál minket a *Custom Charts* menübe.

#### Create Chart-js

Ezen a felületen találhatóak a *Chart-js* grafikonok létrehozásához szükséges beviteli mezők. *(9. ábra)*

**

*9. ábra*

A felület hasonlóan működik, mint a *Create UI5* menüpontban megtalálható beviteli mező. Ehhez a grafikontípushoz nem tudunk megadni előrejelzést, viszont kettő különböző adathalmaz értékeit tudjuk megadni.

Az *Axis 1 Settings* felirat melletti mezők az első adathalmaz adataira vonatkoznak, míg az *Axis 2 Settings* felirat melletti mezők a második adathalmaz adataira vonatkoznak.

Második adathalmazt nem kötelező megadni, egy adathalmazzal is létre tudja hozni a grafikont a felhasználó, viszont a megjelenítendő adatokat az első adathalmaz adataihoz kell beírni.

A *Preview Chart* és az *Add Chart* gomb is a *Create UI5* felületnek megfelelően működik.

#### Update or Delete

Ebben a menüpontban tudja a felhasználó módosítani, vagy kitörölni a grafikont az adatbázisból. *(10. ábra)*



*10. ábra*

A *Chart title* lenyíló menü tartalmazza az adatbázisban lévő grafikonok címeit. Ezek közül a felhasználónak ki kell választania egyet.

Az *Edit Chart* gombra kattintva, előjön a grafikon típusától függően a létrehozáshoz használt menü, egy extra mezővel kiegészítve.

Itt ugyanazokat a beállításokat lehet módosítani, megadni, amiket létrehozáskor is, viszont a beviteli mezők alapértelemzett értéke a grafikon értékeire vannak beállítva, így nem kell minden adatot a felhasználónak újra felvinni.

Két új jelölőnégyzetet típusú mező is található, melyekkel a grafikon elrejtettségét, és könvjelzőzöttségét lehet ki/be kapcsolni.

Az *Update Chart* gombra kattintva, a beviteli adatok alapján frissül a grafikon és a program átnavigál a *Custom Charts* menübe.

A *Delete Chart* gombra kattintva az adott grafikon törlődik és a program átnavigál a *Custom Charts* menübe.

### Run script

Ebben a menüben tud a felhasználó különböző szkripteket futtatni, mellyel az adatbázisban lévő adatokat lehet módosítani, illetve elemezni. *(11.ábra)*

A képen szöveg, Betűtípus, szám, sor látható

Automatikusan generált leírás

*11. ábra*

A *Script* mezőben öt dolgot tud a felhasználó kiválasztani:

* *Monthy Chart refresh*: Ezen szkript futtatása törli az összes generált grafikont és a rendelkezésre álló kriptovaluták adataiból generál egy-egy grafikont mely az ehavi *close* értékeit mutatja.
* *Refresh ticker data*: Ezen szkript kiválasztásakor megjelenik egy mező, ahol az adatbázisban lévő kriptovalutákból lehet választani egyet, és egy időpontot megadni. Ez a szkript törli a kiválasztott kriptovaluta értékeit és a megadott dátum kezdetétől frissíti az értékeket. *(12. ábra)*

A képen szöveg, képernyőkép, sor, Betűtípus látható

Automatikusan generált leírás

*12. ábra*

* *Add new ticker data*: Ezen szkript kiválasztásakor megjelenik egy mező, ahova a kriptovaluta szimbólumát kell beírni, és egy dátumot mely kezdetétől a kriptovaluta értékei bekerülnek az adatbázisba.
* *Remove ticker data*: Ez a szkript futtatása eredményeként, a kiválasztott kriptovaluta összes értéke törlődik az adatbázisból.
* *Analyze ticker data*: Ez a funkció elemzi a kriptovaluta értékeit a megadott időintervallumon belül, és megadja mellé, hogy a trend melyik irányba fog mozogni, Lineáris Regresszió alapján. *(13. ábra)*

A képen szöveg, képernyőkép, sor, Betűtípus látható

Automatikusan generált leírás

*13. ábra*

*A trend nem tükrözi a valóságot, és nem helyettesít semmilyen valódi elemzést.*

A kriptovaluták hozzáadása vagy törlése során automatikusan hozzá tartozó generált grafikon is megjelenik.

A *Run Script* gomb megnyomására a választott szkript a megadott paraméterekkel lefut. A futás során a szöveg *Selected script is running...* jelenik meg ami indikálja a szkript futását a háttérben.

Amint a szkript lefut, ez a szöveg lecserélődik vagy arra, hogy *Script ran successfully* amennyiben sikeresen futott le, vagy arra, hogy *Script ran with errors* amennyiben hibákkal futott le

Ez a szöveg alatt a szkriptnek futása alatt generálódott üzenetek jelennek meg.

# Fejlesztői dokumentáció

## Architektúra

A szoftvernek három fő része van, mely el van különítve három különböző mappába:

* frontend - */react-ui/*
* backend - */crypto-gaze/*
* szkriptek */python\_scripts/*

A szoftver áll egy *React*-ban készített webes felületből, egy *Node.JS* segítségével készített backend részből, melyben *CDS* technológiák kerültek felhasználásra, illetve a programhoz tartozik néhány *python-*ban írt szkript, mely könnyíti az alkalmazásban a nagyobb adatmanipulációs műveleteket.

A *React* frontend *REST* kérések segítségével kéri le az adatokat a backendtől, és ezeket jeleníti meg. Effajta kérések nem csak az adatok lekérése szolgálnak, hanem adatok feltöltésére és különböző műveletek indítására is. A backend kiszolgálja a frontendet adatokkal, és végpontokkal, felelős az adatok tárolásáért egy *SQLite* adatbázis felhasználásával, és a backend hívja meg a különböző szkripteket.

A képen szöveg, diagram, Tervrajz, Műszaki rajz látható

Automatikusan generált leírás

*14. ábra: A program különböző részeinek egymáshoz való viszonyulása*

## Fejlesztői környezet

A program fejlesztése során egy chromium alapú Brave nevű böngészőt használtam. A kód írásához Visual Studio Code javasolt. A program futtatásához, teszteléséhez egy terminál szükséges, futtatáshoz elegendő egy *Powershell* terminál*,* de fejlesztéshezszükséges lehet több terminál ablak, vagy a *Windows terminál* nevű program, mely alapértelmezetten megtalálható a *Windows 11* operációs rendszerben.

### Frontend fejlesztése

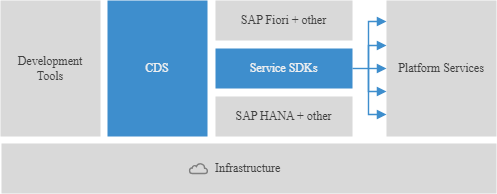
A frontend fejlesztéséhez elég megnyitni a *react-ui* mappát, a frontend összes eleme ott található. Mielőtt a fejlesztést megkezdnénk, ajánlott a terminálban, ugyanebben a mappában lefuttatni az *npm install* parancsot, ami az internetről leszedi az frontend fordításához és futtatásához szükséges összes csomagot. Amint ez megvolt, kiadhatjuk az *npm start* parancsot mely elindítja a *React* fejlesztői szerverét a *http://localhost:3000* címen. Itt meg fog jelenni a frontend jelenlegi kinézete, és minden alkalommal amikor frissítjük és elmentjük valamelyik forrásfájlt, az frontend újra lefordul, és frissül is a megadott címen, ahol böngészővel meg tudjuk tekinteni a fordítás eredményét. Amennyiben meg vagyunk elégedve a frontenddel, az *npm run build* parancs lefordítja, és becsomagolja nekünk a kódunkat, melyet az alkalmazás futáskor fel tud használni. Ahhoz, hogy teljes mértékben tudjuk tesztelni a frontend kódját, a backendet is el kell hozzá indítani, hiszen nélküle nem tudunk kommunikálni az adatbázissal, és nem tudunk adatokat vagy grafikonokat lekérni megjelenítésre.

### Backend fejlesztése

A backend fejlesztéséhez elég megnyitni a *crypto-gaze* nevű mappát, a backend összes eleme ott található. Hasonlóan ahogy a frontendnél, itt is ajánlott a terminálban, ugyanebben a könyvtárba futtatni az *npm install* parancsot, mely a backend által használt csomagokat letölti az internetről. A backendet fejlesztés során tudjuk perzisztens, és nem perzisztens módon futtatni. Amennyiben nem perzisztens módon szeretnénk, a *cds watch* parancsot kell kiadni a terminálban. Ez a parancs elindítja a backendet egy memóriába töltött *SQLite* adatbázissal, ami addig őrzi meg az adatokat amíg le nem állítjuk. Ide bekerülnek kezdőadatként a *\db\csv* mappában található *CSV* fájlok. Hasonlóan, mint a *React* fejlesztői szervernél, itt is minden forrásfájl módosítására az backend újraindul, de a backend mindig a *http://localhost:4004* címen fog elindulni és futni. Ha perzisztenciával szeretnénk fejleszteni, először is, amennyiben nincs még létrehozva adatbázisfájl, le kell futtatnunk a *cds deploy –to sqlite* parancsot, ami létrehozza az adatbázisfájlt. és beletölti a *\db\csv* mappában található *CSV* fájlokat. Ezután a *cds serve* vagy a *cds run* parancsot kell kiadni, és a backend el is indul. Itt viszont minden módosítás életbeléptetéséhez szükséges a backendet újraindítani.

## Backend kialakítása, és működése

### Backend technológai oldala – Core Data Services

A szoftver ezen része, az *SAP CAPire*, vagyis az *SAP* felhő alkalmazási programozási modelljének a gerince. Egy infrastruktúra mely segít létrehozni perzisztens adat modelleket, melyet a felhasználói felület számára szolgáltatásokkal, és végpontok elérhetővé tételével biztosít. A *CDS* segítségével lehetőségünk van különböző definiált modelleket adatbázisban táblákként, nézetekként tárolni, illetve meghatározni a többi alkalmazás a modellek mely részeihez hogyan képesek hozzáférni, azokat módosítani.

*15. ábra: CDS elhelyezkedése a Capire keretrendszeren belül*

A *CDS* modellek egyszerű *JavaScript* objektumok, melyek megfelelnek a *Core Schema Notation* specifikációinak. Ezek a modellek dinamikusan, futási idő alatt kerülnek feldolgozásra, és futási idő alatt is létrehozhatóak.

A *CDS* segítségével meg tudjuk oldani, hogy a *React* frontend egyszerű *HTTP* kérésekkel egy *REST* *api*-n keresztül kommunikáljon a backenddel, és a backenden keresztül az adatbázissal.

A rekordokat lekérni, és manipulálni a *GET POST PATCH* és *DELETE* kérésekkel lehetséges, melyek az alábbi módon felelnek meg a különböző adatbázis operációknak:

|  |  |
| --- | --- |
| **HTTP Query** | **SQL** |
| GET | Select |
| POST | Insert into |
| PATCH | Update |
| DELETE | Delete |

A *REST* hívásoknál, hogy melyik táblára hivatkozunk, azt a definiált végpont dönti el, ahol a modellünk helyezkedik el. Például egy egyszerű szolgáltatásban mely *Catalog Service* néven fut, definiálunk egy *Books* modellt, akkor az *SQL* lekérdezés mellyel lekérnénk az összes elemét a *Books* táblának az az alábbi módon nézne ki:

**SELECT \* FROM Books**

Míg a *CDS* segítségével, elég nekünk egy *GET* kérést elküldeni a *http://localhost:4004/catalog/Books* címre, melyet, ha a böngésző *URL* sávjába beírnánk, *JSON* formátumban vissza is kapnánk az összes rekordot a *Books* táblából. Különböző szűrési és aggregációs műveletekhez tudjuk használni az *OASIS* által szabványosított *OData* protokoll *URL* szabványait, mely a következőképpen épül fel:

A képen szöveg, képernyőkép, sor, Betűtípus látható

Automatikusan generált leírás

*16. ábra: URL lekérdezés felépítése*

Az alkalmazásban használt modelleket több ehhez hasonló szolgáltatásba, modellbe és végpontba szerveztem.

A definiált szolgáltatásokban nem csak entitásokat találhatunk, hanem leírhatunk cselekményeket (*actions*), és függvényeket (*functions*), melyekhez a háttérben személyre szabható kezelő függvényeket adhatunk hozzá, ezzel kibővítve a backend funkcionalitását, és hasonló módon saját kezű eseményeket is hozzáadhatunk.

A szoftverben több kötetlen cselekményt is definiáltam, a különböző adatmanipulációs funkciók biztosításáért.

*Kötetlen cselekmény, az olyan végpont, mely nem tartozik egy adatmodellhez sem, mint egy JavaScript függvény.*

### Adatmodellek

A backend 3-4 adatmodellel dolgozik összesen. Ezek a modellek szimplán az adatok tárolására, és megjelenítésére kellenek.

Ezekből az adatmodellekből, a backend csinál egy-egy táblát az *SQLite* adatbázisban, és a táblák elérhetőek a definiált szolgáltatásokon keresztül, a megfelelő végpontok szerint.

#### Crypto

A Crypto entitásban található az összes kriptovaluta napi rekordja. Egy táblában helyezkedik el minden érték valuta összes értéke, így kevesebb végpont hívással lehet nagyobb terjedelmű adatot lekérni.

Mezői:

* *date*: Rekord feljegyzésének dátuma, *Edm.Date* formátumban
* *open*: Kriptovaluta egy értéke *Edm.Double* formátumban, null mezővel ellátva
* *high*: Kriptovaluta egy értéke *Edm.Double* formátumban, null mezővel ellátva
* *low*: Kriptovaluta egy értéke *Edm.Double* formátumban, null mezővel ellátva
* *close*: Kriptovaluta egy értéke *Edm.Double* formátumban, null mezővel ellátva
* *adj\_close*: Kriptovaluta egy értéke *Edm.Double* formátumban, null mezővel ellátva
* *volume*: Kriptovaluta egy értéke *Edm.Double* formátumban, null mezővel ellátva
* *ticker*: Kriptovaluta szimbóluma *Edm.String* formátumban
* *type*: Kriptovaluta rekordjának típusa *Edm.String* formátumban

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírásA rekord bizonyos mennyiségű mezőjének lehetővé kellett tenni, hogy null értéket is felvehessenek, amennyiben feltöltéskor valamilyen probléma lenne, vagy maga a forrásállomány melyből az adatokat töltenénk fel, hibás értékkel rendelkezne, null értéket táplálhassunk be az adatbázisba, így a grafikonok melyek megjelenítik a lekért adathalmazt, igazsághoz hű képet adjanak.

*17. ábra: Egy kriptovaluta rekord JSON reprezentációja*

A kriptovaluta rekordjának típusa 4 különböző érték lehet:

* *real*
* *forecast\_05*
* *forecast\_075*
* *forecast\_09*

Ezek azt jelzik, hogy a rekord valóságos értéket tartalmaz, vagy valamelyiket a különböző előrejelzésekből. (Az előrejelzések mivoltijáról a szkriptek részletes leírásai közt olvashatunk majd). Amennyiben nem valóságos a rekord, hanem előrejelzés, a *close* érték lesz az egyetlen a kriptovaluta értékei közül mely szám lesz, a többi érték (dátum, típus, tickeren kívül) mind *null*, vagy *0*.

#### PredefinedCharts

Ebben a modellben találhatóak az automatikusan generált havi zárási értékeket megjelenítő grafikonok különböző tulajdonságai.

Mezői:

* *id*: a grafikon egyedi azonosítója *UUID* formában
* *ticker*: a kriptovaluta szimbóluma melyet a grafikon ábrázol *Edm.String* formátumban
* *start\_date*: a grafikon kezdődátuma *Edm.Date* formátumban
* *end\_date*: a grafikon végső dátuma *Edm.Date* formátumban
* *label*: a megjelenített adathalmaz címkéje *Edm.String* formátumban
* *title*: a grafikon neve / címe *Edm.String* formátumban

A *ticker*, *start\_date* és *end\_date* mezők mind azért szükségesek, hogy a *React* frontend, ezekből az adatokból konstruálhasson egy lekérdezést a kriptovalutákról. Ezekben a grafikonokban mindig csak valós értékek jelennek meg, és ezeknek is csak a zárási, vagyis a *close* mezőjük.

A *title* és *label* mezők szimplán a felhasználói élményhez, és a kinézethez szükségesek.

#### CustomCharts

Ebben a modellben találhatóak a felhasználó által létrehozott grafikonok. Ez a modell több tulajdonságot tartalmaz, mint a *PreDefinedCharts*, mivel több fajta grafikont tárolhat egy ilyen rekord, illetve a grafikonok megjelenítéséről is több információt tartalmaz, hiszen a *PreDefinedCharts* modell példányai csak a *Home* felületen jelennek meg, míg a *CustomCharts* példányai megjelennek számos oldalon.

Mezői:

* *id*: a grafikon egyedi azonosítója *UUID* formában
* *ticker*: a kriptovaluta szimbóluma melyet a grafikon ábrázol *Edm.String* formátumban
* *start\_date*: a grafikon kezdődátuma *Edm.Date* formátumban
* *end\_date*: a grafikon végső dátuma *Edm.Date* formátumban
* *label*: a megjelenített adathalmaz címkéje *Edm.String* formátumban
* *title*: a grafikon neve / címe *Edm.String* formátumban
* *field*: a megnevezett kriptovaluta mely értékét jelenítse meg a grafikon *Edm.String* formátumban
* *chart\_type*: a grafikon típusa *Edm.String* formátumban
* *ticker2*: amennyiben meg van adva, a grafikon második adathalmazának szimbóluma *Edm.String* formátumban
* *field2*: amennyiben meg van adva, a grafikon második adathalmazának megjelenített értéke *Edm.String* formátumban
* *label2*: amennyiben meg van adva, a grafikon második adathalmazának címkéje *Edm.String* formátumban
* *forecast*: milyen fajta előrejelzést jelenítsen meg a grafikon *Edm.String* formátumban
* *bookmarked*: a grafikon könyvjelzőzve van-e vagy sem *Edm.String* formátumban
* *hidden*: a grafikon el van-e rejtve *Edm.String* formátumban

A képen szöveg, képernyőkép, szoftver, Betűtípus látható

Automatikusan generált leírásNéhány mező egy az egyben megegyezik a *PreDefinedCharts* mezőivel, nem csak elnevezésben, hanem funkcionalitásban is. A *field*, az lehet bármelyik a *Crypto* modellben található szám típusú értékei közül.

*19. ábra: Egy felhasználó által készített grafikon rekordjának JSON reprezentációja*

A *chart\_type* határozza meg, hogy a frontend milyen fajta grafikonként kezelje a lekért rekordot, amennyiben az érték *simple* a frontend egy *UI5* grafikont fog kirajzolni, amennyiben az értéke *complex* egy *Chart-js* grafikont fog megjeleníteni.

A *simple* grafikonoknál nem kerül figyelembe a *ticker2, field2, label2* mezők, és létrehozáskor ezek a mezők *NULL* értékkel szerepelnek az adatbázisban. A *complex* grafikonoknál, viszont ezek az értékek felhasználásra kerülnek, amennyiben helyes értékkel rendelkeznek, és a frontend ezek alapján egy második lekérdezést indít, és megjeleníti ugyanazon a grafikonon a kettő adathalmazt. A *complex* grafikonoknál a *forecast* mező viszont nem kerül számításba, mivel összetett grafikon létrehozásánál nincs lehetőség előrejelzéseket hozzátenni a megjelenítendő adatokhoz, így létrehozáskor ez a mező mindig a *None* értéket veszi fel.

A *forecast* mező hasonló módon, mint a *Crypto* modellben a *type* mező, az előrejelzésekért felel, és felveheti az alábbi értékeket:

* None
* forecast\_05
* forecast\_075
* forecast\_09

A *None* érték mutatja, hogy a grafikonban nem kívánunk **nem** valós adatokat megjeleníteni, míg a többi érték mutatja, a **nem** valós adatok közül melyik típusút szeretnénk, hogy megjelenjen.

A *bookmarked*, és a *hidden* mezők egyaránt a *True* vagy *False* értéket vehetik fel szöveg formájában, melyek jelzik a grafikon könyvejlzőzöttségét, illetve rejtettségét a frontend felé.

#### CommandResult

Ebben az entitásban találhatóak a lefutott szkriptek eredményei, az itt lévő rekordokat a felhasználó csak akkor látja amikor egy bizonyos szkritpet lefuttat, és kivárja a végeredményt. Általában ez a tábla üres, mivel a frontend kiveszi a megjelenített értékeket, így itt csak a kulcs nélkül indított műveletek, és némelyik nem megjelenített műveletnek a lefutási eredményei vannak.

Ez a tábla automatikusan törlődik minden indítás során.

Mezői:

* *command*: A lefuttatott szkript neve *Edm.String* formátumban.
* *data*: A lefuttatott szkript kimeneti adatai *Edm.String* formátumban.
* *opKey*: A szkript futtatásának egyedi azonosítója *Edm.String* formátumban.

A *command* mezőbe a lefutott szkript neve kerül bele, hogy a konzolt figyelve egyértelműen meg lehessen határozni, hogy milyen típusú szkript indult.

A data mezőben találhatóak különböző hibaüzenetek, vagy szimplán lefutási eredmények, mint például az *analyst.py* szkript futtatásakor, ugyanis akkor nem csak a futás során előjött hibaüzenetek vannak benne, hanem értékes adatok megjelenítéshez, és adatok melyeket a frontend felhasznál ideiglenes grafikon létrehozásához is.

Az *opKey* mezőben egy *UUID*-nek 4-es verziójú karakterlistája található, melyet a frontend használ, hogy nyilván tudja tartani melyik szkriptnek a kimenetét szükséges megjelenítenie a felhasználó részére.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

*20. ábra: Egy meg nem jelenített parancs futási eredménye*

*(az alkalmazás újraindításakor a rekord törlődni fog)*

### Szolgáltatások

#### CryptoService

Ebben a szolgáltatásban található a *Crypto* entitás és három *action* mely a kriptovaluták adatainak módosításához fűződik.

* *DeleteTicker*: Egy kriptovaluta adatait törli az adatbázisból, paraméterei:
  + *ticker*: a kriptovaluta szimbóluma *Edm.String* formátumban
  + *opKey*: a szkript futtatási kulcsa *Edm.String* formátumban
* *AddTicker*: Egy kriptovaluta hozzáadása az adatbázishoz, paraméterei:
  + *ticker*: a kriptovaluta szimbóluma *Edm.String* formátumban
  + *date*: a dátum melytől az adatokat letöltjük *Edm.String* formátumban
  + *opKey*: a szkript futtatási kulcsa *Edm.String* formátumban
* *RefreshTicker*: egy kriptovaluta adatainak frissítése, paraméterei:
  + *ticker*: a kriptovaluta szimbóluma *Edm.String* formátumban
  + *date*: a dátum melytől az adatokat letöltjük *Edm.String* formátumban
  + *opKey*: a szkript futtatási kulcsa *Edm.String* formátumban

Ennek a szolgáltatásnak a három *action* végpontján definiáltunk három különböző függvényt, mely kezeli az ide érkezett hívásokat, és tartalmazza az üzleti logikát.

##### DeleteTicker

A kérésben szerepel, a kriptovaluta szimbóluma, és egy operációs kulcs, amennyiben a kriptovaluta szimbóluma mégsem szerepel, a függvény terminál.

Helyes működéssel, a függvény törli a *Crypto* táblának összes rekordját, ahol a szimbólum megegyezik a kérésben szereplővel, illetve kitörli az ehhez a kriptovalutához generált grafikont is. Ezeket helyben JavaScript lekérdezések segítségével teszi meg. Ezután a sikeres futás eredményét betölti a *CommandResult* táblába.

##### AddTicker

A kérésben szerepel a kriptovaluta szimbóluma, egy kezdődátum, és egy operációs kulcs.

Itt is, hasonlóan, a függvény rögtön terminál, ha nincs érték megadva a *ticker* részének a kérésnek.

A függvény elindítja az *add\_data.py* *python* függvényt, melynek paraméterei a kérésben szereplő argumentumok. Ezután a függvény folyamatosan figyeli, hogy az elindított szkript milyen adatokat ír ki az *stdout*-ra, és ezeket kiíratja a konzolra, melyben elindítottuk az alkalmazást. Amikor az indított szkript terminál, a konzolra ez is kikerül, illetve amennyiben a terminálási kód 0, vagyis futást megakadályozó hiba nélkül terminált a szkript, a függvény létrehoz egy előre generált grafikont, a kriptovaluta e-havi értékeivel, és ezt belehelyezi az adatbázisba, miután törölte az ugyanehhez a kriptovalutához tartozó grafikont, amennyiben létezett volna.

##### RefreshTicker

A *RefreshTicker*, a *DeleteTicker*, és az *AddTicker* funkciójának egybevonása.

Először törli a kriptovaluta adatait, amennyiben a kérés tartalmazza a szimbólumot. Törlés végbevitele előtt, lekéri az első dátumot mellyel a kriptovaluta szerepel, amennyiben a kérés nem tartalmaz érvényes dátumot. Ezután a függvény, hasonlóan, mint az *AddTicker*, elindítja a megadott szkriptet, a megfelelő paraméterekkel, és sikeres futás esetén, törli, és létrehozza a megfelelő grafikont.

#### ChartService

Ez a szolgáltatás tartalmazza a kettő grafikonokhoz kapcsolódó entitást, a *PreDefinedCharts*-ot, illetve a *CustomCharts*-ot is. Mellettük *action*-ként megtalálható a *RefreshCharts* is, paramétere *opKey* – szkript operációs kulcsa, *Edm.String* formátumban.

Ez a cselekmény felel a generált grafikonok frissítésének manuális futtatásáért.

##### RefreshCharts

A kérésben adatként csak egy operációs kulcs szerepel.

A függvény nem csinál mást, csak elindítja a *monthly\_charts.py* nevű szkriptet az operációs kulccsal, mint argumentum, és kiíratja a konzolra, amit a python szkript írna ki, illetve jelzi a szkript terminálásának eredményét.

#### EndpointService

Ez a szolgáltatás tartalmazza a *CommandResult* entitást, illetve kettő *action*-t mely összefüggésben van az entitás mivoltijával.

* *DeleteResult*: A megfelelő szkript eredményének törlése, paramétere *opKey*, a szkript operációs kulcsa*, Edm.String* formátumban.
* *Analyst*: Az *analyst.py* szkript futtatásának meghívásához használja a program, paraméterei:
  + *ticker*: a kriptovaluta szimbóluma *Edm.String* formátumban
  + *start\_date*: elemzés kezdeti dátuma *Edm.String* formátumban
  + *end\_date*: elemzés végső dátuma *Edm.String* formátumban
  + *opKey*: a szkript futásának operációs kulcsa *Edm.String* formátumban

##### DeleteResult

Ez a függvény nem csinál mást, csak helyben kitörli a *CommandResult* táblában lévő rekordokat, melynek megegyezik az operációs kulcsa a kérésben szereplőjével.

##### Analyst

Hasonlóan, mint a legtöbb szkript indításához használt függvénynél, itt is amennyiben nincs megadva a kérésben a kriptovaluta szimbóluma, a függvény terminál. Amennyiben tartalmazza, a függvény meghívja az *analyst.py* szkriptet, a kérésben szereplő szimbólummal, kezdődátummal, végdátummal, és operációs kulccsal.

Itt is a szkript futásának eredményét, illetve a futása során kiírt üzeneteket jeleníti meg a konzolra.

#### Szerver

A szolgálgatások, entitások, végpontok, és ezeket kezelő függvények mellett található még egy szkript mely a szerver különböző eseményeit figyeli, és ezeknek megfelelően cselekszik.

##### Listening

Ez az esemény akkor triggerelődik, amikor a szerver elkezdett a megadott porton figyelni a bejövő kapcsolatokra, és képes azokat fogadni, és kiszolgálni, vagyis amikor a program használatra készen fut. Ilyenkor a függvény először is, kiüríti a *CommandResult* tábla rekordjait, majd kigyűjti egy változóba az elérhető kriptovaluták szimbólumait. Ezután párhuzamosan, minden szimbólummal, amit lekérdezett, lefuttatja az *add\_data.py* szkriptet, azzal a dátummal, amellyel az első adott szimbólumú rekord található az adatbázisban, illetve az *add\_chart.py* nevű szkriptet, mely egy olyan szkript, amelyhez a felhasználó nem fér hozzá a webes felületen keresztül. Ez a szkript felelős a grafikon létrehozásáért, és azért nem egy JavaScript lekérdezés, mert a *for* ciklus aszinkron módon fut, és több műveletet is tartalmaz, melynek futására várnia kell a programnak, így egy külön szkriptbe szervezve, nem kerülnek bizonyos változók átírásra kiolvasásuk előtt/alatt.

##### Bootstrap

Ez az esemény a backend indításának elején történik. Itt a függvény, csak instruálja az alkalmazást, hogy a beépített automatikusan generált felület helyett, a frontend felületet használja, és szolgáltassa a felhasználónak, mikor a böngészőbe beírja az alkalmazás elérési pontját.

Itt található még egy instrukció, mely megmondja az alkalmazásnak, hogy használja a *CORS* nevű csomagot, melynek fő funkciója, hogy különböző címekről is lehessen adatmanipulációs műveletek végezni. Ez kifejezetten hasznos, ha egy felhőben lévő adatbázishoz csatlakoztatnánk az alkalmazást, vagy elindítanánk a *React* fejlesztői szervert, mely a *localhost*-on fut ugyebár, de a *3000*-es porton, mely megszegné a *CORS* irányelveket.

A konfigurációs fájlban egy harmadik függvény is található, mely egy *CronJob*-ot ír le, melyen belül ugyanazon parancsok futnak le, mint a *listening* eseménynek kezelője során. Ez a *CronJob* minden alkalommal amikor éjfélt üt az óra lefut, és frissíti a kriptovaluták adatait, illetve a hozzájuk tartozó grafikonokat, amennyiben éjfélkor, az alkalmazás fut.

## Szoftvert támogató szkriptek működése

A szoftvernek egyes funkcionalitása kiszervezésre került *python* szkriptekbe, mely a könnyebb adatmanipuláció, és különböző előrejelzési, elemzési funkciók ellátása miatt történt.

Ezeknek a szkripteknek nem javasoljuk az elindítását manuálisan, csak a webes felületen keresztül, a legjobb működés elősegítéséhez.

A *python* szkriptek futtatásához *python-*ra szüksége van a rendszernek, és azon belül több különböző telepített csomagra is.

A csomagok melyek szükségesek a szkriptek használatához:

* *datetime*
* *requests*
* *pandas*
* *numpy*
* *darts*
* *time*
* *wget*

### add\_charts.py

Ez a szkript, csak a szoftver számára létezik, a felhasználó nem tudja külön elindítani ezt a szkriptet a webes felületen keresztül.

A szkriptre a szoftvernek azért van szüksége, hogy párhuzamos módon képes legyen több grafikont létrehozni indításkor, vagy éjfélkor.

A szkript argumentumként csak egy kriptovaluta szimbólumot kap, és a *CustomChart* végpontba küld egy grafikont, melyben ez a kriptovaluta szerepel, és a szkript futtatásakori hónap.

### add\_data.py

Ez a szkript a legfontosabb az egész szoftveren belül, hisz ez tartalmazza a megfelelő funkcionalitást, hogy új adatokat lehessen a szoftverbe betölteni.

A szkript először csekkolja, hogy argumentumként érkezett-e operációs kulcs, amennyiben nem, vagy az operációs kulcs *null* vagy *undefined* a szkript egy hibaüzenetként kiírja az *stdout*-ra, illetve egy változóba gyűjti, melyet a *CommandResult* táblájának egyik mezőjébe be fog tölteni.

A szkript ellenőrzi, hogy kriptovaluta szimbólum érkezett-e, mint első argumentum, amennyiben nem, vagy az értéke *null* vagy *undefined* a szkript terminál egyes kóddal, és a hibaüzenetet kiírja a *stdout*-ra, illetve elküldi a *CommandResult* táblájába, mint egy rekordot, melyben a parancs az *add\_data*, az adat a hibaüzenet, és az operációs kulcs az, amit megadtunk.

Ehhez hasonló módon ellenőrzi, hogy a dátum megadásra került-e és átkonvertálható -e *UNIX* időbélyeggé. Amennyiben nem, a szkript kiírja a *stdout-*ra, és a változóba mely a végeredményt közvetíti, a végére hozzáfűzi. Ilyenkor a szkript kiszámol egy öt évvel ezelőtti dátumnak a *UNIX* időbélyegjét, és azzal folytatja a futást.

Ezután a szkript ellenőrzi, hogy az adatbázisban szerepel-e az adott kriptovaluta, amennyiben igen, a szkript terminál egyes hibakóddal, és a hibát kiírja az *stdout*-ra, és elküldi a *CommandResult* táblába is.

A szkript ezután megkonstruálja a letöltési *URL*-t mellyel a *yahoo.com*-ról lekéri egy *CSV* formátumban a kriptovaluta adatait. Amennyiben hiba történik a letöltés alatt, a szkript itt is hasonló módon, terminál egyes kóddal, és a hibát kiírja az *stdout-*ra, és elküldi a *CommandResult* táblába is.

Letöltés után beolvassa a letöltött *CSV* állományt, törli a fájlt, majd átnevezi az oszlopokat, és hozzáadja a *ticker* és a *type* nevű oszlopokat. A *ticker* oszlop minden értéke a megadott szimbólum lesz, és a *type* oszlop értékei mind a *real* érték lesz.

Ezek után az adathalmaznak a dátum és *close* értékeiből készít egy sorozatot, melynek veszi a második felét, a negyedik negyedét, és a tizedik tizedét, mind külön-külön adathalmazokba. Ezeknek a halmazoknak a hiányzó értékeit interpolálással lecseréljük, majd három példányát létrehozzuk az *XGBModelnek*.

Mielőtt a predikciókat lefuttatjuk, létrehozunk egy listát, melyben a prediktált értékekhez tartozó dátumokat legeneráljuk, méghozzá mához a következő 60 nap dátumát.

Ezután mindhárom adatsorozattal melyet létrehoztunk, a következő dolgokat tesszük:

* az eredeti adatszerkezethez hasonlóan, legeneráljuk a *ticker* mezőt, és a *type* mezőt, mely adathalmaztól függően lehet *forecast\_05, forecast\_075, forecast\_09*
* az egyik modellt betaníttatjuk az adathalmazzal, majd 60 értéket előre generálunk, melyeket áttöltünk egy másik formába.
* az így létrejött adatstruktúrának átnevezzük és átrendezzük az oszlopait, hogy megegyezzen a *Crypto* modellben lévő sémával.

Amennyiben a tanítás, vagy a predikció hibába ütközik, kiírjuk az *stdout*-ra, és hozzáfűzzük a változóhoz mely a szkript lefutási állapotát őrzi.

Tapasztalatink szerint, a predikció, és a tanítás, csak abban az esetben ütközik hibába, amennyiben túl kis mértékű adathalmazt adtunk meg a szkriptnek, így nagyobb adathalmaz elemzése esetén mindegyik predikció sikeresen fog lefutni.

Miután a predikciókat lefuttattuk, a szkript meghatározza a kerekítési értéket. Alapesetben három tizedesjegyre kerekít, amennyiben a kriptovaluta *close* értékei közül a legnagyobb 10-nél kisebb, 5 tizedesjegyre kerekít, viszont, ha 1000-nél nagyobb akkor csak 2 tizedesjegyre kerekít.

Miután az adatmanipulációs lépések megvoltak, a szkript az eredeti, és a három előre jelzett adathalmazt, ugyanazon módon próbálja feltölteni az adatbázisba.

Először megpróbálja a rekordot feltölteni, amennyiben hibába ütközik feltöltés közben, megpróbálja a rekordot *null* adatokkal feltölteni, ha így is hibába ütközik a szkript terminál egyes hibakóddal, és a hibát kiírja az *stdout*-ra, és elküldi a *CommandResult* táblába is.

Amennyiben a szkript hibába ütközött volna olyan helyen, melyen nem lett volna lekezelve, a szkript ugyanúgy a hibát kiírja az *stdout*-ra, és elküldi a *CommandResult* táblába is.

### analysis.py

A szkript egy kriptovaluta időintervallum alatti elemzésért felelős, és ezt jeleníti meg a *Run Script* felületen, miután a meghívásra került a webes felületen.

Megadja a legkisebb, legnagyobb értéket és az átlagot az időintervallumon belül, illetve az utolsó 10 napot nézve, ugyanezeket az értékeket. Ezután *LinearRegression* modell segítségével előre jelez egy kialakuló trendet.

Ezeket az értékeket mind a *CommandResult* táblába tölti be, és ezek alapján jeleníti meg a frontend.

A szkript hasonlóan az *add\_data.py*-hoz először ellenőrzi az operációs kulcs létezését, és értékét, majd a megadott kriptovaluta szimbólum létezését, és értékét.

Ezután a kezdő és végdátumot ellenőrzi, amennyiben nincs kezdődátum 1970-01-01-et veszi kezdődátumnak, amennyiben nincs végdátum, *None*-nak veszi az értéket és kihagyja a lekérdezésből.

A változóhoz hozzáköti pontosvessző után a szimbólumot, a kezdődátumot és a végdátumot.

Ezután lekéri a kiválasztott kriptovaluta értékeit, majd beletölti egy adatszerkezetbe. Kiszámítja a kerekítési számot, ugyanazzal a módszerrel, mint az *add\_data.py* szkriptben, majd az alábbi értékeket hozzáteszi a változóhoz:

* *Range*: a legalacsonyabb érték és a legmagasabb érték egy kötőjellel elválasztva
* *Average*: az átlaga a *close* értékeknek

Amennyiben 10 napnál nagyobb a megadott időintervallum, a következő kettő érték is belekerül:

* *10 day range*: ugynaz mint a *Range* csak az utolsó 10 napra nézve
* *10 day average*: ugynaz mint a *Average* csak az utolsó 10 napra nézve

Ezután egy lineáris regressziót használó modell segítségével előre jelzünk 60 napi értéket, majd az első és utolsó előre jelzett nap értékéből megállapítjuk, hogy a trend szerint a kriptovaluta értéke felfelé, vagy lefelé fog mozogni, és ezt is belerakjuk a változóba. Amennyiben itt problémába ütköznénk, azt jelenítjük meg a változóban.

Ezután egy pontosvesszővel elválasztva a változó végére a különböző generált adatokat dátummal együtt belerakjuk, hogy a frontend egy grafikon keretein belül kirajzolhassa a generált értékeket.

Itt is, hasonlóan az *add\_data.py*-hoz, ha bárhol hibába ütközne a szkript, ami nem lenne lekezelve, a szkript terminál egyes hibakóddal, és a hibát kiírja az *stdout*-ra, és elküldi a *CommandResult* táblába is.

### monthly\_charts.py

Ennek a szkriptnek a feladata, hogy törölje, és újra generálja a létrehozott grafikonokat, a felhasználó kérésére. Először kitörli a táblában lévő grafikonokat, majd az adatbázisban lévő kriptovaluták szimbólumaiból, havi grafikonokat generál, és feltölti a *PreDefinedCharts* táblába.

A szkriptnek csak egy argumentuma van, az *opKey*, így először az *add\_data.py*-hez hasonlóan, ellenőrzi létezik-e az argumentum, és amennyiben nem, vagy *undefined*, vagy *null*, ezt kiírja az *stdout*-ra, illetve a változóba, és folytatja a futást.

Ezután a szkript lekéri a kriptovalutákat az adatbázisból és eltárolja.

Majd, a szkript lekéri a generált grafikonokat, és egyesével kitörli őket az adatbázisból.

Ezután, amennyiben talált kriptovaluta adatokat, mindegyikhez konstruál egy grafikont, mely az ehavi értékeit fogja mutatni a kriptovalutának, majd ezeket feltölti az adatbázisba.

Ezután elküldi a *CommandResult* táblába, hogy sikeresen frissítette a grafikonokat.

Hasonlóan, mint a többi szkriptnél, amennyiben itt is egy lekezeletlen hiba történne, a szkript terminál egyes hibakóddal, és a hibát kiírja az *stdout*-ra, és elküldi a *CommandResult* táblába is.

## Frontend kialakítása és működése

A frontend teljes egésze a *react-ui* mappában található, a hozzá felhasznált technológia pedig a *React*.

A *React* egy *JavaScript* alapú keretrendszer, melynek segítségévek egyoldalas webalkalmazásokat tudunk létrehozni egymásba ágyazható komponensek segítségével. Egyoldalas, mivel a teljes frontend egy elérési ponton, *URL*-en található, egy darab *HTML* fájl, és bármilyen változás történik az oldalon, vagy az *URL* sávban, az mind kliensoldali logika alapján történik.

A komponensek *jsx* fájlokba külön mappákba vannak szervezve, és ezekből az *npm run build* parancs létrehoz egy *index.html*-t melyben az összes komponens, amit felhasználtunk, *JavaScript* fájlokba kerül fordításra, mely a generált *index.html* fájlba bele van linkelve.

A komponensek, melyeket a frontendhez készítettem, három mappába vannak rendezve. A *Basic Components* mappában vannak a komponensek, melyeket egy oldalon többször használtam fel, vagy olyan komponensek, melyeket meg lehet jeleníteni több helyen is. A *Layout* mappában találhatóak a frontend elrendezéséért felelős komponensek, melyek biztosítják a webes felület kialakítását. A *Routes* mappában vannak a különböző menüpontokat leíró fájlok. Az frontend belépési pontja a *MyApp.jsx* nevű fájl. Ebben a fájlban vannak egymásba ágyazva a főbb komponensek, mint az Oldalsó navigációs sáv, a *React* navigációért felelős komponens, a *Layout* komponens, melyre később kitérünk, és a különböző útvonalak, melyek segítségével különböző menüpontokat jelenítünk meg, látszólag más és más *URL*-eken, min például a grafikonok listázási komponense a */charts* úton érhető el.

Az alkalmazás felületén az oldalsó sáv, sosem változik, csak lekicsinyíteni lehet. A menüpontokra kattintás során a mellette lévő tartalom változik meg a menüpontokban meghatározott komponensekre.

### Routes

Ebben a mappában találhatóak meg, a különböző menüpontok elemei.

#### Home

Ez a komponens maga a főoldal, amit a felhasználó lát amint az alkalmazás címét beírja az *URL* sávba. Nem tartalmaz semmi különleges dolgot, csak egy *API* kérést a *PreDefinedCharts* táblára, melyből a generált grafikonokat lekéri, és sorban megjeleníti őket a *MyChart* komponens segítségével.

Fontos megjegyezni, hogy a program kialakítása miatt, mivel a *React* kliens oldali útvonalakat használ, bármikor a felhasználó az újra töltés gombot nyomja meg a böngészőben, vagy néha az előre vagy a vissza gombot, az alkalmazás erre az oldalra irányítja át a felhasználót.

#### Charts

Ez a komponens felel, a felhasználó által készített grafikonok megjelenítéséért. Menüpontban megjelenő neve *Custom Charts*

Egy fejléc szerű *Card* komponensben található kettő beviteli mező, mely különböző kiválasztási lehetőségeket ad a grafikonok sorrendbe helyezéséhez. Az első a mezőt, a második az irányt határozza meg. Mellettük van egy jelölőnégyzetet mely megmondja a komponensnek, hogy speciális rendezést alkalmazzon-e vagy sem, mellyel a könvyjelzőzött grafikonok, külön sorba rendezésre kerülnek, és a nem könyvjelzőzött grafikonok előtt kerülnek megjelenítésre. Mellette pedig egy gomb, mellyel a kívánt rendezést lehet megjeleníteni. Az alapértelmezett rendezés, melyet a felhasználó lát amint rákattint a menüpontra, az az elsődleges kriptovaluta szimbólumok alapján történik, növekvő sorrendben, mely ábécé sorrendnek megfelelő rendezést jelent a dátum mezők kivételével, és a könyvjelzőzött grafikonok előre kerülnek. Amint a felhasználó változtatna valamelyik beviteli mező értékén, a rendezett grafikonok eltűnnek, és a *Show Charts* gomb megnyomására jelennek meg újra. A rendezést, ugyanaz a függvény teszi meg, mely a grafikonok lekérdezéséért felelős. Az elrejtett grafikonok itt nem jelennek meg.

A fejléc szerű konstrukció alatt pedig egy egyszerű *ChartList* komponens található, mely miután megadtuk neki a grafikonok rekordjait sorrendben, azért felelős, hogy ezeket megfelelő formában, adatokkal együtt megjelenítse nekünk.

Legtöbb grafikonnal való művelet után, a szoftver, erre az oldalra irányít át minket.

#### BookmarkedCharts

Ez a komponens felel a könyvjelzőzött grafikonok megjelenítésért.

A komponens kifejezetten hasonlít a *Home* komponenshez működés szempontjából, hiszen semmi más feladata sincs, csak lekéri az adatbázisból azokat a grafikonokat melyek nem rejtettek, és könyvjelzőzött értékkel rendelkeznek, majd ezeket átadja egy *ChartList* komponensnek, mely megjeleníti a grafikonokat.

Ha egy grafikont könyvjelzőzünk, az alkalmazás erre az oldalra fog minket átirányítani, amennyiben viszont a könyvjelzőt levesszük egy grafikonról, az alkalmazást a *Custom Charts* menüpontra fog minket átirányítani.

#### HiddenCharts

Ez a komponens felel az elrejtett grafikonok megjelenítéséért, az elrejtett grafikonok itt jelennek meg, és sehol máshol az alkalmazáson belül.

Működésében, kialakításában szinte teljesen megegyezik a *BookmarkedCharts* komponenssel, annyi kivétellel, hogy a lekérdezés, melyet az adatbázisnak küldünk, itt tartalmazza, hogy a rejtett mezője a grafikonnak *True* legyen, mely szűkítés szinte sehol máshol nem található meg a programban.

A lekért grafikonokat, itt is egy *ChartList* komponens jeleníti meg sorban.

Amint egy grafikont elrejtünk, az alkalmazás erre a menüpontra irányít minket, illetve, ha egy grafikonnak megszüntetjük az elrejtettségét, a *Custom Charts* menübe irányít innen minket az alkalmazás.

#### AddSimpleCharts

Ez a komponens felel az egyszerűbb *UI5* grafikonok létrehozásáért.

Nem tartalmaz semmit, csak egy *MyChartSettings* nevű komponenst, melyet később fogunk tárgyalni.

#### AddComplexChart

Ez a komponens felel az komplexebb *Chart-js* grafikonok létrehozásáért.

Nem tartalmaz semmit, csak egy *MyChartSettings2* nevű komponenst, melyet később fogunk tárgyalni.

#### ChartModifier

Ez a komponens felel a grafikonokon végzendő műveletek megengedéséért.

Hasonlóan, mint a *Charts* komponensben, itt is találhatunk egy fejléc szerű elemet, melyen egy kiválasztó menü, és három gomb található meg.

A lenyíló menüben választhatjuk ki, hogy melyik grafikont szeretnénk törölni vagy módosítani. Az *Edit Chart* gomb megnyomására megjelenik egy *ChartEditor* komponens, melyben a grafikon adatai vannak. Az *Update Chart* gomb megnyomására a grafikon rekordjára a frontend elküld egy *PATCH* kérést melyben benne vannak a beírt adatai a grafikonnak. Amennyiben nem változtatjuk az adatokat, a kérés az eredeti adatokkal fut le. A *Delete Chart* gomb megnyomása esetén, pedig egy *DELETE* kérés indul az adatbázis felé, mellyel a grafikon törlődik. Amennyiben nem választottunk ki grafikont, a gombok nem csinálnak semmit. Amennyiben igen, a művelet befejezése után, a szoftver átnavigál minket a *Custom Charts* menüpontra.

#### Scripts

Ez a komponens felel a *python* szkriptek böngészőből való futtatásáért, illetve a szkriptek eredményének kijelzéséért is egyaránt. A menüben ezt a komponens-t a *Run script* néven érjük el.

Ahogy az előző komponensben, itt is találunk egy fejléchez hasonló elemet, melyen különböző beviteli mezők találhatóak, egy lenyíló menü, és egy *Run Script* nevű gomb.

Ez a kettő mező, végig jelen van, bármilyen szkriptet választanánk futtatásra. Mivel néhány szkriptnek egy vagy kettő felhasználó által bevitt paraméter szükséges, így a különböző szkriptek kiválasztásakor, különböző beviteli mezők jelennek meg:

* *Monthly Chart refresh*: semmilyen mező nem jelenik meg
* *Refresh ticker data*: egy lenyíló menü megjelenik, melyből kiválaszthatjuk melyik kriptovalutának szeretnénk az adatait frissíteni, illetve egy dátumválasztó menü is, mellyel az adathalmaz kezdődátumát tudjuk megadni
* *Add new ticker data*: itt egy szövegbeviteli mező jelenik meg, melybe a hozzáadni kívánt kriptovaluta szimbólumát kell beírni, és az előhoz hasonlóan egy dátumválasztó menü is, mellyel az adathalmaz kezdődátumát tudjuk megadni
* *Remove ticker data*: itt egy lenyíló menü jelenik meg, melyben az adatbázisban található kriptovaluták szimbólumai találhatóak, és ezek közül kell kiválasztani az egyet melyet szeretnénk eltávolítani
* *Analyze ticker data*: itt egy lenyíló menü jelenik meg melyben az adatbázisban található kriptovaluták szimbólumai van, illetve kettő dátumválasztó menü, melyben meg lehet adni, hogy a kriptovaluta elemzése, milyen időintervallumon történjen

Miután kiválasztottuk a lefuttatni kívánt szkriptet, és megadtuk a megfelelő adatokat, a *Run script* gomb megnyomására, a frontend elküldi a megadott adatokat a backend egy végpontjára, mely a megfelelő műveletek futtatásáért felelős.

Míg a szkript fut, egy *Selected script is running…* szöveg megjelenik a fejléc alatt. A szkript futása során a frontend, folyamatosan (1/2 másodpercenként) ellenőrzi a *CommandResult* táblát, hogy érkezett-e oda adat a szkript indítása során generált operációs kulccsal. Folyamatosan *API* kéréseket küld egészen addig, ameddig nem jön be rekord, vagy a felhasználó el nem kapcsol a menüpontból. Amennyiben a felhasználó elkapcsol, a szkript futtatási eredményének lekérése, és így megjelenítése is abortál. Miközben a kérések mennek, a *Run Script* gomb is kiszürkül, így nem lehet rögtön új szkriptet indítani, hasak a felhasználó el nem kapcsol egy másik menüpontra majd vissza.

Amennyiben felhasználó megvárja, hogy a szkript termináljon, a lekért értékeket átadja egy *CommandResponse* nevű komponensnek, elérhetővé teszi a *Run Script* gombot, majd a lekért adatot törli az adatbázisból.

### Layout

Ebben a mappában az elrendezésért felelős komponensek találhatóak.

#### Layout

A *Layout* komponens arra van, hogy szimplán elkülönítse a teret, kódban és böngészőben is egyaránt, ahol a változó tartalom fog megjelenni.

#### SidebarComponent

Ez a komponens felel a navigációért az alkalmazásban, benne találhatóak a különböző menüpontok elrendezése, illetve a pszeudo elérési útvonalak, melyeket a frontend kattintásra lekezel, és megjeleníti a kiválasztott tartalmat.

### Basic Components

Ebben a mappában találhatóak a legkisebb egységei a frontendnek, azok a komponensek, melyeket többször, több formában, vagy több helyen is megjelenít a webes felület.

#### ChartEditor

Ez a komponens a grafikon módosításakor kerül elő, és egyetlen feladata, hogy a paraméterek alapján vagy *MyChartSettingsUpdate*, vagy *MyChart2SettingsUpdate* komponens-t jelenítsen meg, illetve a belső változók állapotát közvetítse a szülő és a gyerek komponensei közt.

#### ChartList

Ennek a komponensnek az egyetlen feladata, hogy a bejövő grafikon rekordokon végig iterál, és a grafikon típusa alapján kirajzoljon egy *MyChart* vagy egy *MyChart2* komponenst.

#### CommandResponse

Ez a komponens felel a futtatott szkriptek eredményének megjelenítéséért.

A komponens egyfejléc szerű objektumban kiírja, hogy a szkript sikeresen futott, vagy hibákba ütközött futása során, illetve, az eredmény tartalma alapján több dolgot is kijelezhet.

Amikor az eredmény típusa *simple* akkor csak egy egyszerű szövegmezőt jelez ki, hiszen a szkript futása során csak egyszerű szöveges üzenetek keletkeztek, melyeket szimplán meg tud jeleníteni.

Amennyiben az eredmény típusa *complex*, vagyis az eredményszövegben található egy pontosvessző, mely azt indikálja, hogy az *analyst.py* szkript futott le, mely az eredményben nem csak szöveget helyez, hanem adatokat, mellyel grafikont lehet kirajzolni. A válaszban ilyenkor benne lévő adatok:

* Futás közbeni hibák
* Kalkulált értékek
* Grafikont kijelzéséhez szükséges adatok
* Extra generált adatok a vizualizációhoz

A grafikonhoz tartozó adatokat, és a vizualizációhoz tartozó adatokat, a komponens kiveszi a megjelenítendő adatok közül, és egy *MyChartCommandResponse* komponensnek adja át, mely megjeleníti a grafikont, az extra adatokkal együtt.

#### MyChart

Ez a komponens felelős a *UI5* grafikonok megjelenítéséért. Egy *LineChart* típusú grafikon helyezkedik el rajta, illetve egy fejléchez hasonló elem mely tartalmazza a grafikon címét.

Az alján 2 gomb helyezkedik el, melyek a könyvjelzőzést, és a grafikon elrejtését teszik lehetővé. Ezeknek felirata a grafikon állapotától függ. Az elrejtés, vagy könyvjelzőzés utáni átirányításokat ennek a komponensnek egy függvénye indítja.

A komponens minden adatot a szülő komponens által megadott argumentumokból szed ki, és az itt megadott adatokból rakja össze a grafikont és a hozzá tartozó lekérdezést. Ugyanezen komponens kerül felhasználásra a *Home* komponensen belül, de mivel az ottani grafikonok értékei közt nincs se *bookmarked*, se *hidden*, így az aljukon nem jelennek meg semmilyen gombok.

#### MyChartUpdate

Funkcióban és kinézetben szinte megegyezik a *MyChart* komponenssel, viszont nem található benne semmilyen gomb, hiszen a komponenst csak a különböző grafikonok előnézetére használja a program.

#### MyChartCommandResponse

Egyedi változata a *MyChartUpdate* komponensnek, kinézete megegyezik vele, viszont a benne lévő adatlekérési függvény, a mellékelt extra adatokat is belefűzi a lekért adatokba, így a grafikon a kriptovaluta elemzése során gyártott értékeket is meg tudja jeleníteni gond nélkül ugyanazon a grafikonon.

#### MyChartSettings

Ez a komponens felelős a *UI5* grafikonok létrehozásáért. Hasonlóan ahogy néhány komponensnél, itt is egy fejléc szerű részen találhatóak meg a különböző beviteli mezők:

* Egy szövegbeviteli mező a grafikon címéhez
* Egy lenyíló menü a grafikon szimbólumához
* Egy lenyíló menü a grafikon értekének kiválasztásához
* Egy szövegbeviteli mező a jelmagyarázathoz a grafikonon
* Egy dátumbeviteli mező a kezdődátumhoz
* Egy dátumbeviteli mező a végső dátumhoz
* Egy rádiógombcsoport melyben kiválaszthatjuk, hogy milyen fajta előrejelzés jelenjen meg a grafikonon
* Egy gomb mellyel megtekinthetjük a grafikon előnézetét
* Egy gomb mellyel a grafikont elmenthetjük

Amint a beviteli mezők közül bármelyiket megváltoztatjuk, az előnézet eltűnik és ismét meg kell nyomnunk a *Preview Chart* gombot, hogy megjelenítsük.

Amennyiben az *Add Chart* gombra rányomunk a grafikon hozzáadódik az adatbázishoz, és az alkalmazás átnavigál minket a *Custom Charts* menübe, ahol megtekinthetjük a létrehozott grafikonunkat.

#### MyChartSettingsUpdate

Ez a komponens nagyon hasonlít az előző komponensre, de míg a *MyChartSettings* komponens arra szolgál, hogy grafikonokat létrehozzunk, ez a komponens arra szolgál, hogy létrehozott grafikonok beállítását módosítsuk, így bizonyos dolgokban eltér.

Ezen a komponensen nem található meg az *Add Chart* gomb, mivel ezt a funkciót nem látja el, viszont egy extra beviteli mező tartozik hozzá, kettő jelőlőnégyzet mely a grafikonnak a rejtettségét és könyvjelzőzöttségét irányítja, melyeket itt is be tudunk állítani.

Nagy különbség az előző komponenssel szemben, hogy az itteni beviteli mezők alapértelmezett értéke megegyezik a grafikonok beállításaival.

#### MyChart2

Ez a komponens felel a *Chart-js* típusú grafikonok megjelenítéséért.

Funkcióban, és felépítésben nagyon hasonló a *MyChart* komponenshez, mivel egy fejléc szerű elemben található a grafikon címe, és a grafikon alatt ugyanaz a kettő gomb, mely a könvyjelzőzésért, és az elrejtésért felel.

Maga a grafikon más, mivel kettő adathalmazt jelenít meg egyszerre, így amennyiben megadtunk egy második kriptovaluta szimbólumot a grafikon készítésekor, kettő lekérdezést fog indítani a komponens.

#### MyChart2Update

Ahogy a *MyChart* komponensnek van egy változata melyben nem találhatóak meg a gombok, analóg módon a *MyChart2* komponensnek is van egy ilyen megfelelője. A *MyChart2Update* komponens annyiban különbözik a *MyChart2* komponenstől, hogy az alján a könyvjelzőzést, és a rejtettséget irányító gombok nem jelennek meg. Ezt a komponenst a grafikonok előnézetéhez használja a szoftver.

#### MyChart2Settings

Ez a komponens felel meg a *Chart-js* grafikonok létrehozásáért. Kinézetében, felépítésében nagyon hasonló, mint a *MyChartSettings* komponens. A fejlécben az alábbi beviteli mezők találhatóak:

* Egy lenyíló menü a grafikon első adathalmazának szimbólumához
* Egy lenyíló menü a grafikon első adathalmazának értékének kiválasztásához
* Egy szövegbeviteli mező a jelmagyarázathoz a grafikon első adathalmazának
* Egy lenyíló menü a grafikon második adathalmazának szimbólumához
* Egy lenyíló menü a grafikon második adathalmazának értékének kiválasztásához
* Egy szövegbeviteli mező a jelmagyarázathoz a grafikon második adathalmazának
* Egy dátumbeviteli mező a kezdődátumhoz
* Egy dátumbeviteli mező a végső dátumhoz
* Egy szövegbeviteli mező a grafikon címéhez
* Egy gomb mellyel megtekinthetjük a grafikon előnézetét
* Egy gomb mellyel a grafikont elmenthetjük

Itt is hasonlóan a beviteli mezők bármelyikének módosítására a grafikon előnézete eltűnik, illetve az *Add Chart* gombra való kattintás során a szoftver elnavigál a *Custom Charts* menüpontba.

#### MyChart2SettingsUpdate

Ez a komponens hasonlóan, mint a *MyChart* komponensek esetében, megegyezik lényegében a *MyChart2Settings* komponenssel, viszont az *Add Chart* gomb itt is hiányzik, láthatóak a bookmarked, és a hidden jelölőnégyzetek, illetve a grafikon adatai melyet módosítani kívánunk itt is alapértelmezett adatként megtalálhatóak a beviteli mezőkben.

## Továbbfejlesztési lehetőségek

A projekt moduláris mivoltija miatt több különböző fejlesztési irány áll rendelkezésünkre.

A *CDS* segítségével könnyen adhatunk modelleket a backendbe, szkriptek használatával extra adatmanipulációs funkciókat tudunk implementálni párhuzamosan. A *React* frontend komponensekből való összeállása miatt, minimális változtatásokkal tudunk hozzáadni új oldalakat, menüpontokat.

### Grafikonok

Gyertyagrafikonok segítségével összetetteb képet lehet adni a különböző valutákról. A jelenlegi *React* csomagok között nem nagyon található gyertyagrafikon, melyen több különböző valuta összehasonlítását lehet kijelezni.

### Adatok

A *CDS* lehetőséget ad számunkra, hogy statikus értékek helyett, élőben webszolgáltatásokon keresztül tudjunk különböző szolgáltatóktól adatokat lekérni.

### Adatbázis

A *Capire* szolgáltat nekünk különböző eszközöket, melyekkel könnyen tudjuk az alkalmazást csatlakoztatni egy *Hana* adatbázishoz, mely során a backend funkciókat fel is tudjuk tölteni a *SAP* felhőbe, és ott *hostolni.*

### Kitelepítés (deploymemt)

Az alkalmazást lehetőségünk van konténerizálni, és kitelepíteni különböző felhőszolgáltató cégek rendszerébe, hogy folyamatosan, a világ bármely pontjáról elérhető legyen.

### Felhasználók – Autentikáció

Amennyiben szeretnénk, hogy az alkalmazást több felhasználó használja, lehetőségünk van a *SAP* által készített *XSUAA* rendszert használni, mellyel könnyen hozzá tudunk az alkalmazáshoz felhasználókat adni, és a végpontokat felhasználókra, és jogkörökre korlátozni.

### Funkcionalitás

Az *URL* melyet a letöltési szkript konstruál, kriptovalutákra van állítva, de minimális változtatással rá lehet állítani, hogy részvényeket, vagy *fiat* valutákat töltsön le. A grafikonokat is be lehet állítani, hogy más modellek értékeit mutassák.

### Elemzés – Predikció

A *python* szkriptek fejlesztése során, több elemzési kalkulációt is hozzá lehet adni, mely akkurátusabb képet nyújt a kriptovaluta állapotáról. A különböző modellek használatával, pedig pontosabb predikciót is hozzá lehet adni. Ha a backend pedig egy szervergépen futna, nagyobb erőforrás igényű gépi tanulásos algoritmusokat is lehet az előrejelzésekhez alkalmazni.

## Tesztelés

### Frontend tesztelése

A frontend működését manuális tesztelés során biztosítottam, itt főleg a felhasználói felület elemeinek működése került tesztelésre. Itt tesztelésre került néhány végpont funkcionalitás is a különböző funkciók tesztelése során.

| Műveletsorozat | Környezet | Elvárt eredmény | Valódi eredmény |
| --- | --- | --- | --- |
| Oldalsávon *Menu* feliratra kattintás | Az oldalsáv normál mérettel jelenik meg | Oldalsáv minimalizálódik |  |
| Oldalsávon *Menu* feliratra kattintás | Az oldalsáv minimalizált mérettel jelenik meg | Oldalsáv normál méretre vált |  |
| Oldalsávon menüpontra kattintunk | Nem azon a menüponton vagyunk amire kattintunk | Az oldalsáv mellett a menüpontnak megfelelő tartalom jelenik meg |  |
| A *Custom Charts* menüben átírjuk a rendezési szempontot | - | A megjenelített grafikonok eltűnnek |  |
| A *Custom Charts* menüben átírjuk a rendezési szempontot, majd megnyomjuk a *Show Charts* gombot | A rendezési szempont melyet beállítottunk más, mint ami eredetileg volt, illetve vannak **nem** rejtett grafikonok az adatbázisban | A grafikonok megjelennek az adott rendezésben |  |
| Egy grafikonon a *Hide Chart* gombot megnyomjuk | A grafikonon megtalálható a *Hide Chart* gomb | A grafikonon megjelenik az *Unhide Chart* gomb, a grafikont csak a *Hidden Charts* menüben lehet megtekinteni, a program átnavigál minket a *Hidden Charts* menübe |  |
| Egy grafikonon az Unh*ide Chart* gombot megnyomjuk | A grafikonon megtalálható az Unh*ide Chart* gomb | A grafikonon megjelenik a H*ide Chart* gomb, a grafikont meg lehet találni a *Custom Charts*, a program átnavigál minket a *Custom Charts* menübe |  |
| Egy grafikonon a *Bookmark* gombot megnyomjuk | A grafikonon megtalálható a *Bookmark* gomb | A grafikonon megjelenik az *Remove Bookmark* gomb, a grafikon megjelenik a *Bookmarked Charts* menüben, a program átnavigál minket a *Bookmarked Charts* menübe |  |
| Egy grafikonon a Remove *Bookmark* gombot megnyomjuk | A grafikonon megtalálható a *Remove* *Bookmark* gomb | A grafikonon megjelenik az *Bookmark* gomb, a grafikont eltűnik a *Bookmarked Charts* menüből, a program átnavigál minket a *Custom Charts* menübe |  |
| A *Create UI5* menüben, a beviteli mezők kitöltése után rákattintunk a *Preview Chart* gombra | A beviteli mezők olyan szimbólumot, és dátumokat tartalmaznak, melyekhez megfelelő adatok megtalálhatóak az adatbázisban | A beviteli mezők alatt megjelenik a grafikon előnézete, és legalább egy adatpont megjelenik rajta |  |

| Műveletsorozat | Környezet | Elvárt eredmény | Valódi eredmény |
| --- | --- | --- | --- |
| A *Create UI5* menüben, a beviteli mezők kitöltése után rákattintunk a *Add Chart* gombra | A beviteli mezők olyan szimbólumot, és dátumokat tartalmaznak, melyekhez megfelelő adatok megtalálhatóak az adatbázisban | A program átnavigál a *Custom Chart* menüpontra, és a készített grafikon megjelenik a kijelzett grafikonok közt |  |
| A *Create Chart-js* menüben, a beviteli mezők kitöltése után rákattintunk a *Preview Chart* gombra | A beviteli mezők olyan szimbólumot, és dátumokat tartalmaznak, melyekhez megfelelő adatok megtalálhatóak az adatbázisban | A beviteli mezők alatt megjelenik a grafikon előnézete, és legalább egy adatpont megjelenik rajta |  |
| A *Create Chart-js* menüben, a beviteli mezők kitöltése után rákattintunk a *Add Chart* gombra | A beviteli mezők olyan szimbólumot, és dátumokat tartalmaznak, melyekhez megfelelő adatok megtalálhatóak az adatbázisban | A program átnavigál a *Custom Chart* menüpontra, és a készített grafikon megjelenik a kijelzett grafikonok közt |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Műveletsorozat | Környezet | Elvárt eredmény | Valódi eredmény |
| Az *Update or Delete* menüpontban egy grafikont kiválasztunk, és megnyomjuk a *Delete Chart* gombot | Legalább egy saját készítésű grafikon megtalálható az adatbázisban, és nincs rejtettre állítva | A program átnavigál a *Custom Charts* menüpontra, és a grafikont, melyet kiválasztottunk, nem jelen meg sehol |  |
| Az *Update or Delete* menüpontban egy grafikont kiválasztunk, és megnyomjuk a *Edit Chart* gombot | Legalább egy saját készítésű grafikon megtalálható az adatbázisban | A program, a grafikon típusának megfelelő beviteli mezőket jeleníti meg |  |
| Az *Update or Delete* menüpontban egy grafikont kiválasztunk, megnyomtuk az *Edit Chart* gombot, majd átírunk egy értéket, és megnyomjuk az *Update Chart* gombot | Legalább egy saját készítésű grafikon megtalálható az adatbázisban, és nincs rejtettre állítva, és a változtatás során **nem** állítottuk rejtettre | A program átnavigál a *Custom Charts* menüpontra, és a grafikont, melyet kiválasztottunk, megjelenik a kirajzolt grafikonok között változott konfigurációval |  |

Az itt található teszt főleg a backend kezelő függvényeinek, vépgpontjainak, és a szkriptek futásának eredményeit ellenőrzi, így az ide tartozó funkciókat, egy tesztkeretrendszer segítségével ellenőriztem.

|  |  |  |  |
| --- | --- | --- | --- |
| A *Run Script* menüben az *Add new ticker data* szkriptet lefuttatjuk a *Run Script* gomb megnyomásával | A Ticker mezőbe olyan kriptovaluta szimbólumát írtuk, mely az adatbázisban nem található, és a dátum melyet beírtuk mai nap vagy korábbi dátum | Az adatbázisban megjelenik a kiválasztott kriptovaluta az adott kezdődátumtól kezdődően, és a Home menüpontban megtaláljuk a kriptovaluta ehavi értékének változását mutató grafikont |  |

### Backend tesztelése

A backendnek a végpontjait automatikusan teszteltem, a *Jest* tesztrendszer segítségével *Chia* asszerciókkal. A teszteket az *npm run test* parancs futtatásával indítottam, a *crypto-gaze* mappában, és az eredmény a konzolon jelent meg.

|  |  |  |  |
| --- | --- | --- | --- |
| Műveletsorozat | Környezet | Elvárt eredmény | Valódi eredmény |
| POST *Crypto* | - | *201*-es kód |  |
| GET *Crypto* | A POST *Crypto* tesztesetben használt adatokat kérjük le, és hasonlítjuk a kézzel megadottakhoz | A kézzel megadott érték a lekért rekordok között van |  |
| DELETE *Crypto* | A POST *Crypto* tesztesetben használt szimbólum alapján meghívásra kerül a *DeleteTicker* endpoint | *204*-es kód |  |
| POST *PreDefinedCharts* | A feltöltött rekord egyedi azonosítóját elmentjük egy külön változóba | *201*-es kód |  |
| GET *PreDefinedCharts* | A POST *PreDefinedCharts* tesztesetben feltöltött grafikon kerül lekérésre és összehasonlításra | A kézzel megadott érték a lekért rekord adataival megegyezik |  |
| DELETE *PreDefinedCharts* | A POST *PreDefinedCharts* tesztesetben feltöltött grafikon kerül törlésre | *204*-es kód |  |
| POST *CustomCharts* | A feltöltött rekord egyedi azonosítóját elmentjük egy külön változóba | *201*-es kód |  |
| GET *CustomCharts* | A POST *CustomCharts* tesztesetben feltöltött grafikon kerül lekérésre és összehasonlításra | A kézzel megadott érték a lekért rekord adataival megegyezik |  |
| DELETE *CustomCharts* | A POST *PreDefinedCharts* tesztesetben feltöltött grafikon kerül törlésre | *204*-es kód |  |
| POST *CommandResult* | - | *201*-es kód |  |
| GET *CommandResult* | A POST *CommandResult* tesztesetben használt adatokat kérjük le, és hasonlítjuk a kézzel megadottakhoz | A kézzel megadott érték a lekért rekordok között van |  |
| DELETE *CommandResult* | A POST *CommandResult* tesztesetben feltöltött grafikon kerül törlésre | *204*-es kód |  |

## Felhasznált technológiák

A szoftver fejlesztése során több különböző harmadik féltől származó technológiát használtam fel. Ezek között találhatóak csomagok melyeket egy komponens megjelenítéséhez használtam, és vannak komplett programok melyek esszenciálisak voltak a szoftver fejlesztéséhez.

* Visual Studio Code
* Brave
* React
* NodeJS
* HTML
* CSS
* JavaScript
* CDS
* axios
* cors
* express
* jest
* node-schedule
* react-pro-sidebar
* supertest
* chart-js
* moment
* react-datepicker
* react-router-dom
* react-scripts
* react-uuid
* uuid

# Irodalomjegyzék