

SZAKDOLGOZAT



MISKOLCI EGYETEM

Interaktív megjelenítő eszköz pénzügyi adatok elemzéséhez

Készítette:

Bencze Zsombor

Programtervező informatikus

Témavezető:

Dr. Karácsony Zsolt

MISKOLC, 2023

MISKOLCI EGYETEM

Gépészmérnöki és Informatikai Kar

Alkalmazott Matematikai Intézeti Tanszék

Szám:

SZAKDOLGOZAT FELADAT

Bencze Zsombor (LP5J4B) Programtervező informatikus jelölt részére.

A szakdolgozat tárgyköre: Webfejlesztés, pénzügy

A szakdolgozat címe: Interaktív megjelenítő eszköz pénzügyi adatok elemzéséhez

A feladat részletezése:

A szakdolgozat célja olyan interaktív, dinamikus megjelenítési módok tervezésének, működésének és használatának a bemutatása, amelyekkel egyazon idősor különböző részei, különböző forrásból származó idősorok, az azokból származtatott értékek összehasonlíthatók, az aggregáláshoz használt paraméterek rugalmasan változtathatók.

A pénzügyi adatok (például tőzsdei árfolyamok) elemzéséhez elengedhetetlen, hogy a rendelkezésre álló információk a szakértők számára könnyen áttekinthető formában rendelkezésre álljanak

Az alkalmazásnak webes környezetben, szerver-kliens architektúrának megfelelően kell elkészülnie. Ehhez szerver oldalon Node.JS programnyelvet fogok használni. Míg a kliens weboldal megvalósításához HTML5, CSS, JavaScript programnyelveket kell használni.

Témavezető: Dr. Karácsony Zsolt (egyetemi docens)

A feladat kiadásának ideje: 2023. február 16.

.....
szakfelelős

EREDETISÉGI NYILATKOZAT

Alulírott **Bencze Zsombor**; Neptun-kód: LP5J4B a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős Programtervező informatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírással igazolom, hogy *Interaktív megjelenítő eszköz pénzügyi adatok elemzéséhez* című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szó szerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, év hó nap

.....

Hallgató

1.

szükséges (módosítás külön lapon)

A szakdolgozat feladat módosítása

nem szükséges

.....

dátum

.....

témavezető(k)

2. A feladat kidolgozását ellenőriztem:

témavezető (dátum, aláírás):

konzulens (dátum, aláírás):

.....

.....

.....

.....

.....

.....

3. A szakdolgozat beadható:

.....

dátum

.....

témavezető(k)

4. A szakdolgozat szövegoldalt

..... program protokollt (listát, felhasználói leírást)

..... elektronikus adathordozót (részletezve)

.....

..... egyéb mellékletet (részletezve)

.....

tartalmaz.

.....

dátum

.....

témavezető(k)

5.

bocsátható

A szakdolgozat bírálatra

nem bocsátható

A bíráló neve:

.....

dátum

.....

szakfelelős

6. A szakdolgozat osztályzata

a témavezető javaslata:

a bíráló javaslata:

a szakdolgozat végleges eredménye:

Miskolc,

.....

a Záróvizsga Bizottság Elnöke

Tartalomjegyzék

1. Bevezetés	1
2. Elméleti háttér	2
2.1. Grafikonok története és típusai	2
2.1.1. Grafikon típusok [4]	3
2.2. Tőzsde	5
2.2.1. Részvény	5
2.3. Befektetési alapok	6
2.3.1. Tőzsdei árfolyam	8
3. Specifikáció	9
3.1. Tervezés	9
3.2. Felhasznált technológiák	10
3.2.1. HTML5	10
3.2.2. CSS	11
3.2.3. Node.js	13
3.2.4. JavaScript	14
3.3. Chart.js bemutatása	15
3.3.1. Chart.js összehasonlítása D3.JS keretrendszerrel	16
4. Implementáció	18
4.1. Áttekintés	18
4.1.1. Problémakör	18
4.1.2. Program működésének elve	19
4.2. Fejlesztési eszközök	19
4.2.1. Részvények adataihoz való hozzáférés	19
4.2.2. Postman	20
4.2.3. Fejlesztői környezet	21
4.2.4. Git	21
4.3. Kliens	22
4.4. Szerver	26
5. Alkalmazás bemutatása	30
5.1. Alkalmazás felépítése	30
5.2. Alkalmazás panelei	31
5.2.1. Kezdőlap	31
5.2.2. Grafikonrajzoló	32
5.2.3. Befektetések	33
5.2.4. Kapcsolat	34

5.2.5.	Design megtervezése	34
5.2.6.	Navigáció	35
5.2.7.	Lábléc	35
5.2.8.	Logó	36
5.3.	Alkalmazás áttekintése	36
5.3.1.	Más oldalakkal összehasonlítás	36
5.3.2.	Miért fontosak a tőzsdét elemző weboldalak?	36
5.3.3.	Hogyan lehet felhasználni adatok elemzéséhez?	37
5.3.4.	Részvényelemzés	37
6.	Összefoglalás	38
6.1.	Továbbfejlesztési lehetőségek	39
6.2.	Summary	39
	Irodalomjegyzék	41

1. fejezet

Bevezetés

A szakdolgozatom célja egy weboldal megtervezése és implementálása, amely alkalmas többféle pénzügyi adat elemzéséhez. Ehhez különféle befektetési alapokat használok, továbbá egy grafikonrajzoló. Az oldalnak bemutatom a szerkezeti felépítését, elkészítését és az alkalmazás működését. A pénzügyi adatok, vagy tőzsdei árfolyamok megismerése lehet nagyon egyszerű, de gyakran nehézséget jelent azon személyek számára, akik először próbálkoznak meg vele a hétköznapi életben. Ezért a dolgozatom olyan lehetőségeket mutat be, amelyekkel átláthatóbban lehet megérteni és kezelni őket.

Ezeket a lehetőségeket kétféle alternatívára osztottam szét. Az egyik opció a részletes leírása és ismertetése a befektetési alapoknak, kiegészítve egy táblázattal, amely ábrázolja a hozam-kockázat profilt, továbbá egy alap által elért éves nettó hozam sávot. Másik opcióként külön oldalon grafikonrajzoló használható. Lehetőség van kezdési és zárópontot megadni, így különböző intervallumokat kiválasztva tudjuk vizsgálni az adott befektetési alapot. A dolgozat az utóbbira fektet nagyobb hangsúlyt több okból kifolyólag is. Ezt elsősorban az indokolja, hogy az így nyert adatokat rugalmasabban tudja kezelni a felhasználó, mivel egyszerre több kiválasztási lehetőség áll rendelkezésre, amivel részletesebb adathalmaz nyerhető ki. A teljesség igénye nélkül felsorolok pár lehetőséget. Például, napi adatokat lehet vizsgálni olyan kategóriák szerint, mint napi legmagasabb, vagy éppen napi legalacsonyabb árfolyam. Továbbá a legelterjedtebb ábrázolási forma ha vonaldiagrammon ábrázoljuk a kívánt adatokat. Az alkalmazás elkészítése során számomra szempont volt, hogy ettől függetlenül több Választást kínáljak a felhasználónak, így lehetősége nyílik több ábrázolási módszer közül is választani, úgy mint például oszlopdiagram, vagy kördiagram.

A megjelenítéshez és az algoritmusok fejlesztéséhez több technológia is kiválasztásra került, többek között HTML5 a weboldal vázát, CSS a weboldal megjelenítését írja le, Node.JS amellyel a webhely szervere működik és végezetül JavaScript nyelven válik elérhetővé a grafikonrajzoló teljes egésze. Ezekon felül még használtam további JavaScript könyvtárat, mint a JQuery, illetve az oldal reszponzivitásáért felel a nyílt forráskódú kliens oldali keretrendszer a Bootstrap5.

Több grafikonrajzoló és eszközkezelő weboldal elérhető a különböző befektetési vállalatoknak az Interneten. Ebből kifolyólag felmerülhet a kérdés, miért volt szükség még egy weboldal elkészítésére? Többek között erre a kérdésre is választ kaphatunk a szakdolgozat elolvasása után.

2. fejezet

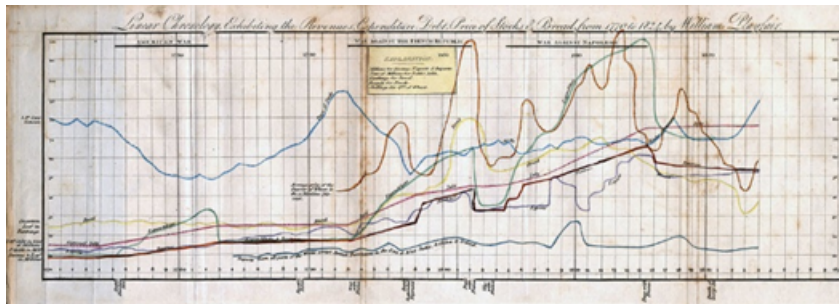
Elméleti háttér

Szakdolgozatom célja létrehozni egy olyan kliens-szerver architektúrájú weboldalt, amely bárki számára könnyen használható és látványos kimutatásokat tud ezáltal készíteni. Már a tervezési fázisban elhatároztam, hogy a weboldal felépítése hasonlítani fog a kutatásom során megismert tőzsdei témájú webhelyekhez, mind felépítésében, mind megvalósításában, hogy a felhasználó azt érezze egy *élő és lélegző weboldalon* böngészik. Ehhez viszont előtte szeretném ismertetni a szakdolgozatomhoz kapcsolódó fő tématerületeket.

2.1. Grafikonok története és típusai

Egy látványos grafikonnal könnyebb bemutatni egy elemzést, vagy történetet, mint szóban elmesélni. Ugyan ma alapkellékként értelmezzük a vonaldiagrammokat és kördiagrammokat, viszont egykor forradalmi újításnak számítottak. Volt rá példa, hogy használatuk szó szerint életbevágónak bizonyult, amikor egy angol orvosnő, diagrammok segítségével győzte meg a férfiak által vezetett orvosközösséget, hogy jobban oda kell figyelni a kórházi higiénára. A kifogástalan és figyelem felkeltő grafikonjának hála, katonák tömegei éltek túl a krími háborút. [1]

A grafikon a diagram egyik fajtája, amely két adat kapcsolatát egy derékszögű koordináta-rendszerben ábrázolja. A két tengely jelképezi a két adatot, és a grafikon jelzi, hogy az egyik adatnak az egyes értékeihez a másik mely értékei tartoznak. Általában két mennyiség közötti kapcsolatot, vagy egy mennyiség időbeli változásának bemutatására használják. Matematikai szempontból a grafikon egy függvényt ábrázol. [2]



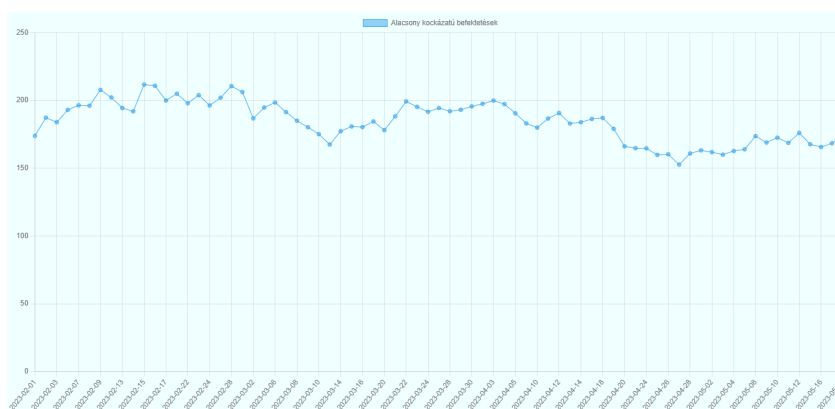
2.1. ábra. Az egyik első statisztikai gráf (forrás: [3])

2.1.1. Grafikon típusok [4]

Vonaldiagram

A vonaldiagram olyan grafikon, amely vonalakat használ az egyes adatpontok összekapcsolására. A vonaldiagram kvantitatív értékeket jelenít meg egy meghatározott intervallumban. A pénzügyekben a vonaldiagrammokat általában egy eszköz vagy értékpapír történelmi árfolyamműveletének ábrázolására használják (a 2.2).

Működését tekintve egy vonal köti össze az egyes adatpontokat, amelyek jellemzően mennyiségi értékeket jelenítenek meg. Befektetések terén a technikai elemzés területén a vonalgrafikonok meglehetősen informatívak, lehetővé téve a felhasználó számára a trendek megjelenítését. Míg a vonaldiagrammokat sok különböző mezőben használják különböző célokra, leggyakoribb funkciójuk az értékek időbeli változásainak grafikus ábrázolása.

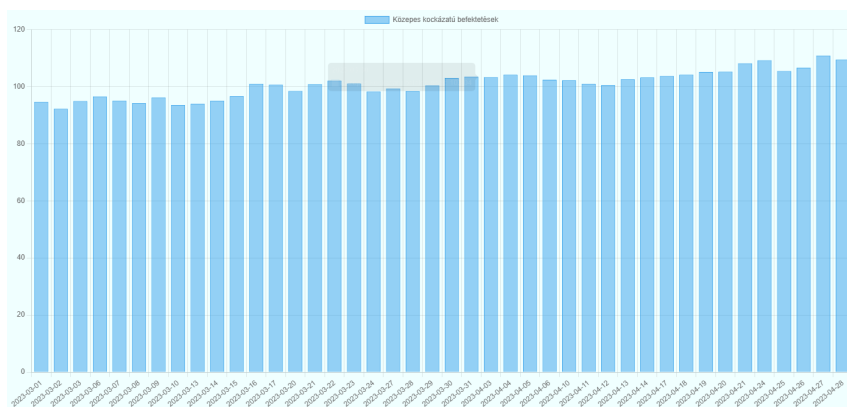


2.2. ábra. Grafikonrajzoló oldalon megjelenített vonaldiagram

Oszlopdiaagram

Az oszlopdiaagram az információ grafikus ábrázolására szolgál. Különböző magasságú sávokat használ az értékek megjelenítésére (a 2.3).

Működését tekintve oszlopdiaagrammokat létre lehet hozni függőleges sávokkal, vízszintes sávokkal, csoportos sávokkal (több sáv, amelyek egy kategória értékeit hasonlítják össze) vagy halmozott sávokkal (több típusú információt tartalmazó oszlopok). A sávok egy adott adatkategória értékét jelenítik meg, az hatázzorra meg annak méretét. Az oszlopdiaagramokat általában az üzleti és pénzügyi elemzésekben használják a gyakran bonyolult adatok megjelenítésére, mivel gyorsan és hatékonyan tudnak információt közvetíteni.

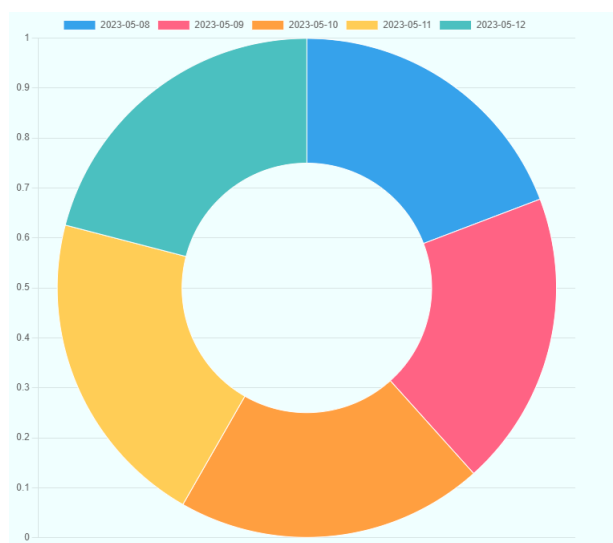


2.3. ábra. Grafikonrajzoló oldalon megjelenített oszlopdiagram

Kördiagram

A kördiagram egy névleges adathalmaz összegzésének vagy egy adott változó különböző értékeinek megjelenítésének módja (pl. százalékos eloszlás). A kördiagrammok használata meglehetősen népszerű, mivel a kör vizuális koncepciót ad az egészről. Egyik alfaja a poláris diagram (a 2.4).

Működését tekintve az ilyen típusú diagram egy szegmenssorozatra osztott kör. Minden szegmens egy adott kategóriát képvisel. Az egyes szegmensek területe a körnek ugyanolyan arányban van, mint a kategória a teljes adatkészletben.



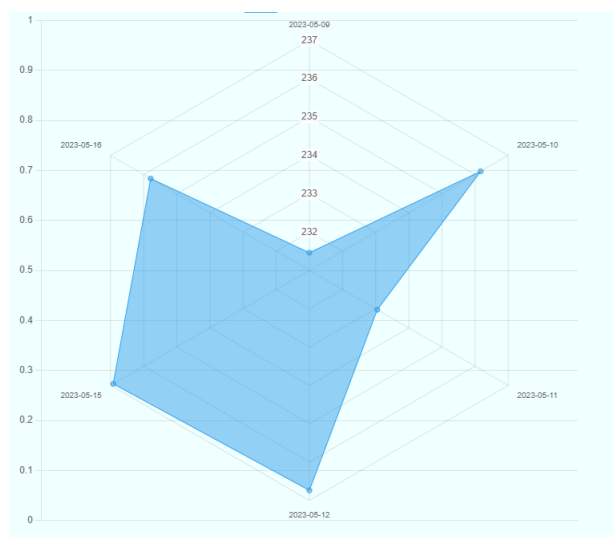
2.4. ábra. Grafikonrajzoló oldalon megjelenített kördiagram

Radardiagram

A radardiagram segít szemléltetni a különböző jellemzőkkel rendelkező adatcsoportok összehasonlítását. Segítségével könnyen össze lehet mérni másfajta paraméterekkel rendelkező elemeket (a 2.5).

Működését tekintve az adatokat ugyanarra a középső pontra helyezi, és átlátszó árnyalatokkal és mintákkal mutatja be a kontrasztot az Olvasó számára. Egy adott

pont értékének nagyságát az jelzi, minél távolabbra nyúlik az origótól a pókháló szerű diagrammon.



2.5. ábra. Grafikonrajzoló oldalon megjelenített radardiagram

2.2. Tőzsde

A tőzsde talán a modern világ egyik legjobban félreértelmezett kifejezése. A hétköznapi ember lelki szemei előtt számok végtelen sokasága lebeg, amikor meghallja és csak legyint rá, hogy ez számára túl komplikált, ahhoz, hogy átlássa és megértse. Nos, ez az állítás részben igaz is, hiszen lehet nagyon bonyolult a tőzsde, viszont lehet nagyon egyszerű is.

A tőzsde egy olyan nyilvános, központosított és szervezett piac, ahol meghatározott árukat, meghatározott időben, azon belül meghatározott személyek adhatnak, vagy vehetnek szigorú eljárási szabályok szerint. Ez mit is jelent a gyakorlatban? A tőzsde is egy olyan piac, mint amelyet akármelyik városban, vagy faluban találhatunk. A különbség az, hogy vásárlás során maga az áru nem materiális formában jelenik meg, illetve nem csak az adott környezettel, hanem az egész világgal lehetőségünk van kereskedni. Viszont, hogy mibe érdemes fektetni az bonyolultabb, mint elsőre képzelnénk. [5]

A tőzsdék fajtái lehetnek:

- Árutőzsde,
- Értéktőzsde, ezen belül:
 - Devizatőke,
 - Értékpapírtőzsde ,
 - Hírpiac.

2.2.1. Részvény

A részvény tulajdonjogot és egyéb gazdasági jogokat megtestesítő értékpapír, nincsen lejárat ideje. Ezeket az értékpapírokat a részvénytársaságok a részvény birtokosainak,

annak részesedése arányában osztják szét jövedelem és szavazati jogok arányos formájában.

A részvények nagyjából névre szóló értékpapírok, amelyeket leginkább elektronikus úton, dematerializált formában állítanak elő, hogy kereskedelmi forgalomba hozhatóak legyenek. A fizikai értékpapírok tárolása leginkább bankokban, vagy takarékszövetkezetekben történik. A részvényekkel tőzsdén kívüli piacokon is kereskednek, amelyekre általában lazább szabályok vonatkoznak, mint a központosított piacokon.

A részvénytulajdonlás jövedelem növelése érdekében történik. Egy vállalat termelő, vagy szolgáltató tevékenységének célja a profitszerzés, hogy ezt elérhetővé tegye részvénytársasággá kell vállalnia. A vállalkozásban lévő részesedést az alapító tagok között szokás meghatározni, hogy mely tag mekkora összeggel járult hozzá az adott vállalkozás elindításához. Abban az esetben, ha csak az alapítók rendelkeznek részesedéssel, akkor zárt részvénytársaságról beszélünk. Amennyiben az alapítók úgy határoznak, hogy a céget a nyilvánosság elé tárják, akkor elkezd részvényeket kibocsájtani. Ezen részvények összessége arányos azzal a tulajdonjoggal, amiről az alapítók lemondtak.

Többfajta részvénytípusok elérhetőek, a teljesség igénye nélkül felsorolok párat, például: befektetési életbiztosítások, **befektetési alapok**, nyugdíj célú megtakarítások. [6]

2.3. Befektetési alapok

Azon személyek, akik pénzüket be szeretnék volna fektetni valamilyen szolgáltatásba, már nagy eséllyel hallottak a befektetési alapokról. Ezen termékek kedvezőek és meggyőzőek lehetnek bárki számára. A tőzsde egyik „alapszabálya”, hogy befektetéskor nem ajánlott egyetlen terméket vásárolni. A diverzifikáció ugyanis nagyon fontos, hiszen nem csak ezzel csökkenthetjük a kockázatot, hanem sokkal több tapasztalatot is tudunk gyűjteni és minél több csapot nyit meg a befektető annál nagyobb eséllyel lesz sikeres. [7]

A befektetési alap olyan vagyontömeg, amely állhat értékpapírból, ingatlanokból, bankbetétekből és részvényekből. Befektetési alapot csakis befektetési alapkezelő hozhat létre és kezelhet. Egy befektetési alapkezelő több befektetési alapot is kezelhet. A befektetési alapok futamideje változó, beszélhetünk akár pár hónapos futamidőről, de akár évtizedekig tartó időintervallumról is. A rövid futamidejű alapokat likvidálási alapoknak hívjuk és mindig nyitott végű alapok. Mit jelent az a kifejezés, hogy *nyitott végű alapok*?

A befektetési alapokat két fajta típus alapján különböztetjük meg, ezen típusok jelzik, hogyan lehet csatlakozni egy alaphoz.

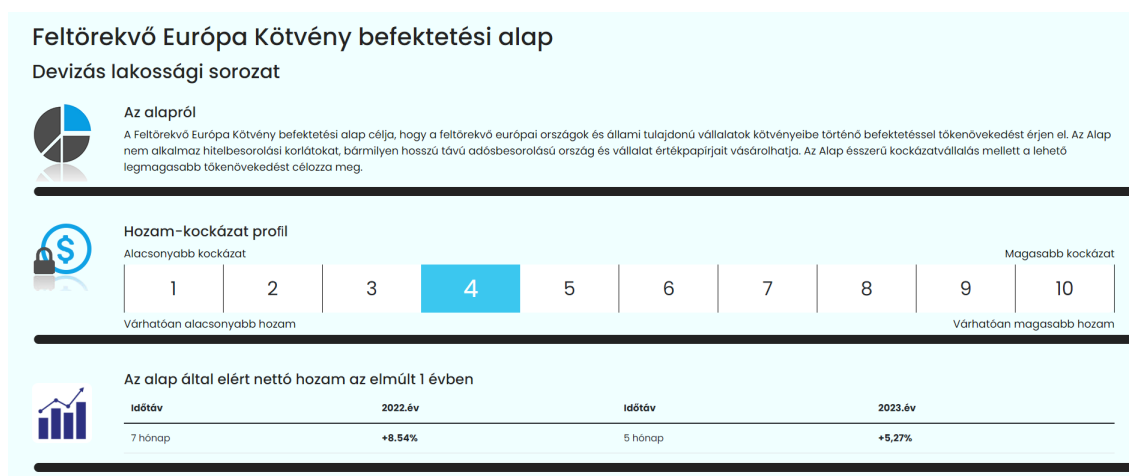
- **Nyíltvégű alapok:** bármelyik forgalmazási napon lehetőségünk nyílik csatlakozni, illetve vissza is válthatjuk belőle a megtakarításainkat. Ezek az alapok általában határozatlan futamidőre jönnek létre és mi dönthetjük el, meddig tartjuk bent a megtakarított pénzüket. Mindig érdemes figyelni a javasolt befektetési időtávokat, hogy a befektetésünk jó eséllyel az elvárt teljesítményt tudja nekünk szolgáltatni.
- **Zártvégű alapok:** a zártvégű alapokhoz csak az alap indulása előtti jegyzési időszakban van lehetőségünk csatlakozni. Ezen alapok általában határozott futamidőre jönnek létre, tehát lejáráttal rendelkeznek, ezen időszakig a megtakarításainkat az alapban kell tartanunk. Természetesen, ha a futamidő alatt mégis

szükségünk lenne a befektetett összegre, a zártvégű alapokat tőzsdei megbízás útján értékesíteni tudjuk, ilyenkor a pillanatnyi kereslet fogja meghatározni a befektetésünk ellenértékét, nem pedig annak az értéke. Ezért, fontos jól megfontolni, ha zártvégű alapot választunk, célszerű, és javasolt a befektetésünket a lejáratig megtartani.

Az alapokat az alapján is csoportosíthatjuk, hogy azok milyen típusú eszközbe fektetik a befektetők megtakarításait.

- **Pénzpiaci befektetési alap:** pénzpiaci alapok, állampapírok és különböző banki betétek.
- **Kötvényalapok:** különböző kötvények vásárolhatók.
- **Résztvényalapok:** vállalatok által kibocsátott részvények.
- **Vegyes alapok:** többféle portfólióban csoportosított részvények és kötvények.
- **Ingatlan befektetési alap:** már megépült vagy építésben levő ingatlanokba való befektetés.
- **Speciális befektetési alapok:** abszolút hozamú, tőkevédett és származtatott alapok.

Mivel ilyen sokrétű lehet egy befektetési alap, így felmerülhet a vásárlóban, hogy miért érdemes befektetési alapba fektetnie a pénzét? Erre a kérdésre az egyik legjobb érv a kockázatmegosztás. Ugyanis már egyetlen befektetési alap megvásárlásával is több egyedi értékpapírból álló, szakszerűen összeállított portfólióhoz juthatunk hozzá, amit egyéni befektetőként rengeteg idő és erőfeszítés árán érhetnénk el. Amennyiben befektetési alapot választunk megtakarításaink elhelyezésére nincsen más dolgunk, mint vásárolni az alap befektetési jegyeiből. Ezzel az egyszerű tranzakcióval gyakorlatilag megbízunk egy szakértői gárdát, akik az összes a befektetésünkkel kapcsolatos terhet levesznek a vállunkról, és különféle értékpapírokba helyezik el megtakarításaikat (a 2.6).



2.6. ábra. Az alkalmazás egyik befektetési alapja

2.3.1. Tőzsdei árfolyam

Tőzsdei árfolyamoknak nevezett árfolyamokat a globális pénzügyi piacon határozzák meg, ahol a bankok és más pénzügyi intézmények éjjel-nappal kereskednek valutákkal ezen tényezők alapján. Az árfolyamot általában az általa képviselt nemzeti valuta betűszóval jegyzik, mint például a leggyakrabban használt valuta az *USD*, ami az amerikai dollárt jelenti. Mérésének számos módja van, a leggyakoribb módszer a kétoldalú árfolyam mérése. A bilaterális árfolyam az egyik valuta másikhoz viszonyított értékére vonatkozik. A tőzsdén található árfolyamoknak több típusát különböztetjük meg, ezek az alábbiak lehetnek:

- Nyitó árfolyam,
- Napi maximum érték,
- Napi minimum érték,
- Napi záró érték,
- Napi átlag érték. [8]

3. fejezet

Specifikáció

Az előző fejezetben taglaltam milyen fogalmakkal találkozhat a felhasználó az alkalmazásban. Ebben a fejezetben ismertetem a szoftver tervezésének lépéseit, a felhasznált technológiák technikai hátterét, továbbá bemutatom az általam választott grafikus könyvtárat, a Chart.js-t, és hogy miért erre esett a választásom.

3.1. Tervezés

A programom tervezése során több aspektust vettem figyelembe. Elsődleges szempont volt, hogy webalkalmazásként működjön legjobban. Ezt az érvet azzal tudom alátámasztani, hogy szem előtt tartva az előnyeit (könnyű hozzáférés, naprakész adatok, jól szemléltethető, skálázható), illetve személyes szakmai tudásomat is a frontend fejlesztés során szereztem. Másodlagos érveimet pedig azzal tudom alátámasztani, hogy az egyetemi tanulmányaim során is webfejlesztésre szakosodtam.

Az alkalmazás tervezésekor az egyik általam ismert agilis módszertan szerint dolgoztam, amelynek neve *Extreme Programming* (XP). Az Extreme Programming egy specifikus keretrendszer, amelynek célja nem csak a kiváló minőségű szoftver(ek) előállítása, hanem az egész folyamat megkönnyítése is a fejlesztő számára. A fejlesztési folyamatot 5 fázisra lehet bontani. [9]

Tervezés (Planning)

- Piackutatás
- Funkciók meghatározása
- Szerver-kliens architektúra megtervezése
- Felhasználandó technológiák

Elemzések (Analysis)

- Alkalmazás struktúrára felosztása
- Elkészüléshez szükséges idő meghatározása
- Szakmai tudás felmérése
- Erőforrás tervezés fejlesztői oldalon

Design

- A feladatok lebontása
- Összpontosított megjelenés létrehozása és végrehajtása

Kivitelezés (Execution)

- Kódolás
- Kész egységek tesztelése
- Hibajelentés generálása
- Iteráció közbeni és végi áttengkítés
- Folyamatfejlesztések

Zárás (Closure)

- Az elkészült termék üzembe helyezése
- Felhasználói kézikönyv
- Végso tesztelés

3.2. Felhasznált technológiák

A webfejlesztés számos szabályt és technikát tartalmaz, amelyeket minden webhelyfejlesztőnek tudnia kell. Mivel a számítógépek nem tudnak egymással úgy kommunikálni, mint az emberek, helyettük kódokra van szükségük. Egy progresszív webalkalmazás rendelkezik mind a webhely, mind a helyileg telepített program jellemzőivel, továbbá a modern alkalmazások új lehetőségeivel.

3.2.1. HTML5

Mivel a programom online felületre lett tervezve, annak szerkezeti felépítése HTML-ben íródott. A HTML (HyperText Markup Language), magyarul hipertext jelölő nyelv, egy leíró nyelv, amely egy weblap felépítését specifikálja az internetes böngészők felé. Ahogy nevében is láthatjuk, nem egy programozási nyelv, amely azt jelenti, hogy nem programozási logikák megírására, vagy adatok kezelésére használjuk, hanem különböző utasításokkal meghatározzuk az oldalunk struktúráját. A webböngészők HTML dokumentumokat fogadnak egy webszerverről vagy helyi tárolóról, és a dokumentumokat multimédiás weboldalakká alakítják. A HTML szemantikailag írja le a weboldal szerkezetét, és a megjelenésére utaló jeleket (a 3.1)


```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

3.1. ábra. HTML felépítés és elemek megjelenítése

HTML Canvas

A Canvas elem a HTML5 része, és lehetővé teszi két dimenziós alakzatok és dinamikus bittérképek megjelenítését. Több elterjedt 2D API-khoz hasonló rajzfunkciók teljes készletén keresztül érheti el a felhasználni kívánt területet, ezzel elérhetővé téve a dinamikus generált grafikákat. Főbb felhasználási területe: grafikonok, animációk, játékok és képalkotás.

Mivel a grafikonok megvalósításához elengedhetetlen, ezért az én applikációm részét is képezi. A Canvas elemre láthatunk egy példát a 3.2 ábrán:

```
<!DOCTYPE html>
<html>
<body>

<canvas id="myCanvas" width="200" height="100" style="border:1px solid #000000;">
</canvas>

</body>
</html>
```

3.2. ábra. Egy üres négyzet, megvalósítva Canvas elem által

3.2.2. CSS

Ahhoz, hogy a HTML által definiált oldalunkon megjelenjenek a kívánt elemek és ezeket jól olvashatóan és igényesen tudjuk megformázni, szükségünk van egy olyan nyelvre, ami a HTML elemekre tud hivatkozni, és különböző nézeti tulajdonságokat tud neki átadni. Ezen célra alkalmas a CSS (Cascading Style Sheets), magyarul lépcsőzetes stíluslapok, egy leíró nyelv, amely a HTML elemek megjelenítését, azok stílusát (például: elhelyezkedés, betűszín, karakter formátumok stb.) írja le a böngésző számára.

Különböző választókkal jelölhetünk ki elemeket, azok nevére, osztályára vagy típusára hivatkozva. A lépcsőzetesség arra utal, hogy mely választók vannak prioritizálva a weblap megjelenítésének szempontjából. Ez fontos, hisz összetett weblapok esetén többszáz CSS szabály is használatban lehet.

CSS stílusokat megadhatunk különféle eljárások szerint (a 3.3).

- Közvetlenül egy HTML elemen keresztül pontosvesszőkkel elválasztva. Ezt a változatot *sorközi stílus*nak (inline CSS) nevezzük.
- Használhatjuk *belső stílus*ként (internal CSS), amely egyedi kinézetet biztosít egyetlen dokumentumhoz. A HTML <head> részében kell meghatározni <style> címkén belül.
- Szakmailag legelterjedtebb felhasználása a *külső stílus* (external CSS). A külső stíluslapot általában akkor használjuk, ha több oldalon szeretnénk változtatni. Ideális erre az állapotra, mert megkönnyíti a teljes webhely megjelenésének megváltoztatását egyetlen fájl módosításával. Fejrészen belül elhelyezzük <link> címkék közé a külső fájl forrását, amiben a kódunk szerepel.

Inline CSS

```
<p style="color: blue;">This is a paragraph.</p>
```

Internal CSS

```
<head>  
  <style type = text/css>  
    body {background-color: blue;}  
    p { color: yellow;}  
  </style>  
</head>
```

External CSS

```
<head>  
  <link rel="stylesheet" type="text/css" href="style.css">  
</head>
```

3.3. ábra. Különböző CSS elérési útvonalak. (forrás: [10])

Bootstrap5

A Bootstrap egy ingyenes, nyílt forráskódú frontend fejlesztői keretrendszer webhelyek és webes alkalmazások létrehozásához. Úgy tervezték, hogy lehetővé tegye a webhelyek reszponzív fejlesztését, és a Bootstrap szintaxis gyűjteményt biztosít a sablontervekhez, így a fejlesztőknek csak be kell illeszteni a kódot egy előre meghatározott CSS-keretrendszerbe (grid). Ez a rendszer 12 oszlopos rácsrendszert használ, ezáltal egy reszponzív weboldalt több módon egyenletesen lehet felosztani, ahogy az eszköz nézete megváltozik különféle felbontásban (a 3.4).

```

<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css"
  </head>
  <body>
    <div class="container text-center">
      <div class="row">
        <div class="col">
          Column
        </div>
        <div class="col">
          Column
        </div>
        <div class="col">
          Column
        </div>
      </div>
    </div>
  </body>
</html>

```

3.4. ábra. Bootstrap beillesztése HTML dokumentumban és rácsrendszer felosztása (Forrás: [11])

Rácsrendszer: A rácsrendszerek olyan segédeszközök, amelyeket a tervezők a nehezen átlátható weboldalak megoldásaként valósították meg, hogy az információk rendezettek legyenek és következetes felhasználói élményt biztosítsanak a felhasználók számára. Logikáját tekintve szakított az informatikában gyakran használt decimális számokkal, mivel ezen számokat nem lehet felezni, harmadolni, úgy, hogy egész számokat kapjunk. Így merült fel az a meglátás, miszerint a tároló konténert 12 egyenlő részre osztja fel. Szabadon beállíthatjuk az adott sor(ok), vagy oszlop(ok) milyen szélességgel rendelkezzenek, esetleg a tároló kontéren mekkora részének feleljen meg a képernyőn (a 3.5).

span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1
span 4				span 4				span 4			
span 4				span 8							
span 6						span 6					
span 12											

3.5. ábra. 12 oszlopos rácsrendszer felosztása (forrás: [12])

Font Awesome

A legegyszerűbb módja annak, hogy figyelemfelkeltő ikonokat helyezzünk el weboldalunkon, ha ikon készletet használunk. Én a Font Awesome internetes ikonkönyvtárát és eszközkészletét használtam fel.

3.2.3. Node.js

A Node.js egy nyílt forráskódú, többplatformos futási környezet, amiben a gépünkön JavaScript kódot tudunk végrehajtani. Széles körben alkalmazzák szerveroldali programozáshoz, így a fejlesztők használhatják a JavaScriptet kliensoldali és szerveroldali kódokhoz anélkül, hogy további nyelvet kellene megtanulniuk. Általában egy bizonyos

címen és porton figyel a szerver, és mikor kérést kap, elvégzi a leprogramozott funkciókat, majd tétlen állapotban várakozik a következő hívásig. Mivel a Node.js nem folytat közvetlenül kimeneti, vagy bementi műveleteket, így erőforrások blokkolása nem merül fel, ezáltal nem kell félni, hogy a program elakad. Tehát a Node.js ideális interaktív megjelenítő alkalmazások fejlesztésére. [13]

Node Package Manager

A Node.js futtatásához Node Package Manager (NPM), azaz Node Csomagkezelőt alkalmazunk. Az NPM egy nyilvántartott adatbázis, amelyben szoftverek és a hozzájuk található metaadatok szerepelnek. Három részre lehet felosztani, úgy mint weboldal kezelés, parancssori felület és nyilvántartás.

Express.JS

Az Express.js egy Node.js backend keretrendszer, amelyet arra terveztek, hogy az API webalkalmazásait gyorsan és platformokon átívelő alkalmazásokat készítsen, és megkönnyítse a node.js-t dolgát. Webes alkalmazások és RESTful API-k építésére tervezték anélkül, hogy bármiben is korlátozná a szerver funkcióit.

3.2.4. JavaScript

A JavaScript, amelyet gyakran JS-ként is szoktak említeni, egy olyan többparadigmás, dinamikus programozási nyelv, ami lehetővé teszi, hogy a weblapokon komplex funkciókat valósíthassunk meg HTML és CSS használatával. A webhelyek majdnem 100%-a JavaScriptet használ az elsőrendű függvényeivel, továbbá az alacsony erőforrásigényével. Támogatja a prototípus-alapú objektumorientációval és első osztályú funkciókkal rendelkező kódolást, továbbá az eseményvezérelt, funkcionális és kötelező programozási stílusokat.

Működését tekintve, nem típusos nyelv, így nem szükséges a változókat deklarálni használat előtt, a megfelelő típust futás közben veszi fel. A JavaScriptet támogató böngésző betölti a HTML oldalt, amennyiben kód található benne a JavaScript elemző motor kerül előtérbe és a script kód betöltődik. Alkalmazásprogramozási felületekkel (API-kkal) rendelkezik a szöveggel, dátumokkal, reguláris kifejezésekkel való munkához. A legnépszerűbb futásidejű rendszer erre a felhasználásra a Node.js. (Bár a Java és a JavaScript nevüket és szintaxisukat tekintve hasonlóak, a két nyelv különbözik, és nagymértékben eltér egymástól.) [14]

JQuery

A jQuery egy nyílt forráskódú, gazdag JavaScript-keretrendszer, amely leegyszerűsíti a HTML/CSS-dokumentumok, pontosabban a Dokumentum Objektum Model (DOM) és a JavaScript közötti interakciókat. Szintaxisát úgy tervezték, hogy megkönnyítse a dokumentumban való navigálást, kezelést, a DOM- elemek kiválasztását, az animációk létrehozását, az események kezelését és az Aszinkron JavaScript XML (Ajax)- alkalmazások fejlesztését.

A jQuery lehetőséget biztosít, a fejlesztők számára, hogy absztrakciókat hozzanak létre alacsony szintű interakcióhoz és animációhoz, fejlett effektusokhoz és magas szintű,

témára alkalmas widgethez. A könyvtár moduláris megközelítése lehetővé teszi hatékony dinamikus weboldalak és webalkalmazások létrehozását.

JSON

A JavaScript Objektum Jelölés (JSON) egy nyílt szabványos, nyelvtől független fájlformátum és adatsere-formátum, amely ember által olvasható szöveget használ az adatobjektumok tárolására és továbbítására. Nagyon elterjedt adatformátum, amelyet sokrétűen alkalmaznak az elektronikus adatcserében, beleértve az adatcserét webes alkalmazások és szerverek között. A JSON szerkezete a következőképpen épül fel:

- Név - érték párok gyűjteménye. Különböző nyelveken ez objektumként, rekordként, struktúraként, szótárként, hash-táblaként, kulcsos listaként vagy asszociatív tömbként valósul meg.
- Az értékek listája rendezett.
- Ezen értékek lehetnek, sztringek, számok, másik JSON, tömbök, vagy boolean értékek (a 3.6).

A .json fájlnevet ezen objektumok kiterjesztésére használják, nem csak JavaScript programozási nyelven íródott kódokra is.

```
{
  "name": "John",
  "age": 30,
  "employee": { "name": "John",
                 "age": 30,
                 "city": "New York"
               },
  "employees": ["John", "Anna", "Peter"],
  "sale": true
}
```

3.6. ábra. Egy alkalmazott adatai JSON-ként tárolva

3.3. Chart.js bemutatása

A Chart.js egy ingyenes, nyílt forráskódú JavaScript-könyvtár adatvizualizációhoz, amely nyolc diagramtípust támogat: vonal, oszlop, terület, torta (fánk), buborék, radar, poláris és szórt diagram. JavaScript nyelven használt beépülő modulokat, diagramtípusokat és testreszabási lehetőségeket kínál (a 3.7). A számtalan beépített modul mellett, a közösség által karbantartott diagramtípusokat is elérhetővé tesz. Ezen felül több diagramtípus kombinálható vegyes diagrammá ugyan azon a Canvas elemen belül. [15]

```

<div>
  <canvas id="myChart"></canvas>
</div>

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<script>
  const ctx = document.getElementById('myChart');

  new Chart(ctx, {
    type: 'bar',
    data: {
      labels: ['Red', 'Blue', 'Yellow', 'Green', 'Purple', 'Orange'],
      datasets: [{
        label: '# of Votes',
        data: [12, 19, 3, 5, 2, 3],
        borderWidth: 1
      }]
    },
    options: {
      scales: {
        y: {
          beginAtZero: true
        }
      }
    }
  });
</script>

```

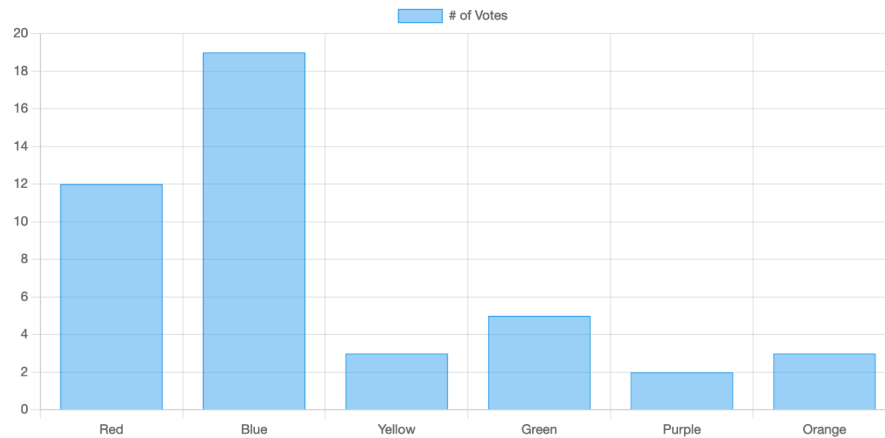
3.7. ábra. diagram létrehozása JSON formátumban (forrás: [15])

3.3.1. Chart.js összehasonlítása D3.JS keretrendszerrel

A csillagok száma alapján a GitHubon a második legnépszerűbb JavaScript diagramkönyvtár a D3.js után, amelyet ugyan lényegesen könnyebben kezelhetőnek tartanak, de kevésbé egyéni igényeket kiszolgáló, mint az utóbbi. A Chart.js HTML5 Canvas elemen belül jelenik meg, és széles körben elismert, mint az egyik legjobb adatvizualizációs könyvtár. A Chart.js nagymértékben testreszabható, hogy néhányat említsek: egyéni beépülő modulokkal, amelyek segítségével megjegyzéseket, nagyítást vagy húzással hozhat létre.

A Chart.js a diagramelemeket HTML5 Canvas elemek jelenítik meg, ellentétben számos más, többnyire D3.js-alapú diagramkönyvtárral, amelyek Skálázható Vektor Grafika (SVG)-ként jelennek meg. A Canvas megjelenítés nagyon hatékonyvá teszi a Chart.js-t, különösen nagy adatkészletek és összetett vizualizációk esetén, amelyek egyébként több ezer SVG-csomópontot igényelnének a DOM-fában. Ugyanakkor a Canvas renderelés nem engedélyezi a CSS stílust, ezért ehhez beépített opciókat kell használni, vagy különböző diagramtípusokat kell létrehozni, hogy a kívánt gráfok jelenjenek meg.

A Chart.js nagyon jól alkalmazható nagy adathalmazokhoz. Az ilyen adatkészletek hatékonyan feldolgozhatóak belső formátum működtetésével, így eredményes az adatok elemzéséhez és normalizálásához. Végül a Chart.js által kezelt Canvas megjelenítés csökkenti a DOM-fájának terhelését, míg az SVG-megjelenítéshez nagyon erőforrás igényes tud lenni. [16]



3.8. ábra. Oszlopdiagram ami szavazatok számát mutatja meg (forrás: [15])

Chart.js előnyei

- Mivel a Chart.js egy JavaScript könyvtár, így lehetővé teszi, hogy bármilyen választott JS keretrendszerrel dolgozzunk, mint például az Angular.js vagy a React.js.
- Használata könnyű és minimális fejlesztői ismeretet igényel.
- Nyílt forráskódú, így csak a meglévő könyvtárat kell igénybe venni és nem a fejlesztőnek kell mindent felépíteni.
- World Wide Web Consortium (W3C) szabványt követi, így nincs szükség más böngészőhöz tartozó technológiára, vagy bővítményre.
- Több megjelenítő eszközzel ellentétben 8 féle diagram megjelenítését engedélyezi.
- Adatok hatékony manipulálását teszi lehetővé. Rendkívül gyorsan jelenik meg és saját animációkkal rendelkezik.
- Kiváló dokumentációval van ellátva.

4. fejezet

Implementáció

A Harmadik fejezet alapján a felhasználó naprakész háttérinformációval rendelkezik a fejlesztési folyamattal és a szükséges technológiák ismeretével kapcsolatban. Ezen ismeretek megszerzésével ebben a fejezetben prezentálni fogom az elkészült alkalmazás látható és háttérmunkaként szolgáló programrészeit.

4.1. Áttekintés

A szakdolgozatom témája interaktív megjelenítő eszköz (ez esetben weboldal) pénzügyi adatok elemzéséhez. Ilyen szoftverekből különféle megvalósítást lehet találni az Interneten, amelyeket nagyobb befektetői háttérrel rendelkező cégek működtetnek általában valamilyen pénzügyi szolgáltatás ellenében. Számomra elsődleges szempontot képviselt a szabad és pénzmentes felhasználás, a könnyű kezelhetőség, átláthatóság és sokszínűség, különösképpen a grafikonok területén. A már bemutatott Chart.js technológia nagyszabású könyvtárának ismeretével megalkotott gráfok több lehetőséget is a felhasználó elé tárnak, mind megjelenés, mind adathalmazok terén.

4.1.1. Problémakör

Ahogy az Áttekintésben felvázoltam több grafikonrajzoló és eszközkezelő weboldal létezik napjainkban. Ezen oldalakat vizsgálva feltűnt, hogy több megkötés árán tudtam hozzájutni a kívánt adatokhoz. Az egyik ilyen megkötéssel akkor szembesültem, amikor a grafikonrajzolóhoz nem engedett hozzáférni felhasználó nélkül. Mivel teljes mértékben piackutatás céljából vizsgálódtam, nem tartottam fontos szempontnak felhasználó regisztrálását, így az oldal használatát a továbbiakban nem tudtam folytatni.

Következő korlátozás amivel találkoztam, mely szerint a biztosítási cégek által használt webhelyek részvények és portfóliók széles körű tárházával rendelkeznek, de jogi és piaci okokból kifolyólag korlátozott az adatbázisuk, hiszen más vállalatok termékeit nem híresztelhetik. Kutatásaim során meggyőződtem, hogy ezen adatok nyilvánosak az ügyfelek számára, tehát illusztrálásuk engedélyezett bármely kliens részére.

Továbbiakban széleskörű adatforrással rendelkező portálokat után keresgéltem és sikeresen találtam pár ilyen paraméterekkel rendelkező weboldalakat. Azonban túlnyomórészt idegen nyelven íródtak, ami globális szinten érthető megközelítés, viszont mint magyar anyanyelvű felhasználó ezt problémának éreztem.

4.1.2. Program működésének elve

Ebben az alfejezetben pár szóval ismertetem, hogy az alkalmazást milyen felhasználói körnek hoztam létre első sorban. Alapvetően adatbányász, tőzsdei jelenléttel rendelkező vállalatoknak ajánlanám, hiszen korlátozás nélkül bővíthető az adathalmaz, ezáltal optimalizálható bármely piaci részre. Struktúrájában kulcsszerepet képviseltek az alábbi szempontok.

- **Megjelenés:** Felépítése, megjelenése letisztult és modern, jelképezve egy innovatív oldal benyomását.
- **Reszponzivitás:** Weboldalam célja, hogy optimális megjelenést biztosítson (könnyű olvashatóság, egyszerű navigáció) a legkülönbözőbb eszközökön át az asztali számítógépektől egészen a mobiltelefonokig. Az oldal tökéletesen igazodik a megjelenítő eszközhöz a rugalmas felépítésével és flexibilis képeivel.
- **Egységes színvilág:** A színvilág megvalósításánál elengedhetetlen tényező volt, hogy a felhasználónak kellemes színeket használjak, amelyek harmonizálnak egymással. Szubjektív kikötésem volt a fehér háttérszín elvetése, mert számomra szemvakító.
- **Egységes formavilág:** Az egész weboldalnak egységes a formavilága, könnyen felismerhető panelekkel felépítve. A gombok, képek és szöveg megjelenése nem tér el panelenként szemben sok más alkalmazással.
- **Navigálhatóság:** A weblap összes panele könnyen elérhető akár a navigációs sáv használatával, akár az oldalak közti kommunikációval.
- **Funkciók:** Implementált funkciók szerepüket működésük szerint betöltik, bármely böngészőn futtatjuk az alkalmazást.
- **Tartalom:** Releváns és minőségi értéket ad a látogató számára.
- **Honlapszerkezet:** Az oldalak hierarchikusan vannak felépítve, tehát logikusan következnek egymásból.


4.2. Fejlesztési eszközök

Webfejlesztőként arra törekedtem, hogy modern weboldalt hozzak létre. Ezt a szempontot nem csak kódolás, hanem a kijelölt feladatok megvalósításánál is szem előtt tartottam, mint például a kliens és a szerver kezelése. Az alkalmazás során több webfejlesztő eszköz segítette a fejlesztés teljes folyamatát. A következő modulban ezeket az eszközöket fogom bemutatni.

4.2.1. Részvények adataihoz való hozzáférés

A részvények adatainak az igénybe vételét külsős oldal segítségével oldottam meg. Az oldal amit használok az *EOD Historical Data*, amely egyszerű hozzáférést biztosít az alapvető tőzsdei adatokhoz és az országokból származó részvényekhez és befektetési alapokhoz. Az oldalon található adatok védettek, ezért egy biztonsági eljárás útján tudom elérni őket. Ezen eljárás alapján a regisztrált felhasználómhoz az oldal generál úgy

nevezett "API KEY"-t, ami minden felhasználónak egyedi és elengedhetetlen része az adatokat lekérő URL-nek. Miután megszereztem az kulcsot a lekérdezésben feltétlenül szerepelnie kell, különben nincs jogosultságunk az adatok eléréséhez. [17]



```
https://eodhistoricaldata.com/api/fundamentals/AAPL.US?api_token=demo
```

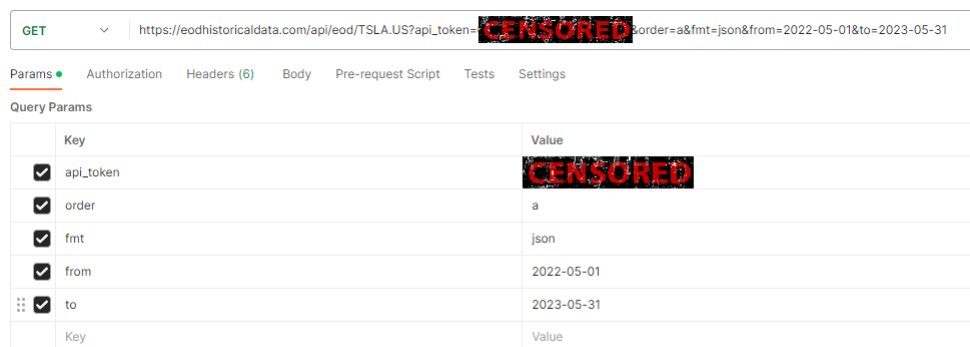
4.1. ábra. API hívás demo ID-val (forrás:[17])

Elérhető fizetős és ingyenes változata is, én az utóbbit használom, aminél némi megkötéssel kellett szembesülnöm, mint például a maximum időintervallum, amire az API-hívások szólhatnak az 1 év, tehát a grafikon ábrázolásának van határa. Ezen felül naponta csak 20 hívást tudok intézni, így erre a problémára egy megoldással kellett előállnom, amit a **Szerver** szekcióban fogok ismertetni.

4.2.2. Postman

A fenti részben említett API hívásokat külső applikációval valósítom meg, ez pedig a *Postman*. A Postman az egyik legnépszerűbb szoftvertesztelő eszköz, amelyet API tesztelésére, létrehozására, megosztására és dokumentálásra alkalmaznak a fejlesztők.

Működését tekintve igen egyszerű a használata. Elsőként ki kell választani milyen típusú hívást akarunk intézni, mivel adatokat szeretnénk elérni, ezért "GET" kéréssel hivatkozok. Továbbiakban szükségem van az elérni kívánt weboldal URL címére. Következő feltételhez tőzsdei ismeretek szükségesek. Minden nyilvánosan működő részvénytársaság regisztrálva van a globális piacon, ezáltal regisztrációs címmel, úgy nevezett "TOKEN"-nel rendelkeznek, például a legismertebb tőzsdei token a TSLA, nevéből kifolyólag a TESLA részvényeire vonatkozik (a 4.2). Ezen teendők után meg kell adni a kéréshez tartozó időtartam kezdeti és záró dátumát, ha nem adjuk meg a záró dátumot, akkor a lekérés automatikusan az aktuális napot állítja be.



4.2. ábra. Adatok lekérése Postman használatával (adatvédelmi okokból cenzúráztam a kulcsot)

4.2.3. Fejlesztői környezet

A **Visual Studio Code** nyílt forráskódú kódszerkesztő, mely ingyenesen elérhető a fejlesztők számára. Az egyik legnépszerűbb alkalmazás, hiszen számos kiegészítővel rendelkezik, amely leveszi a terhet a felhasználó válláról, továbbá majdnem az összes ismert programozási nyelvet támogatja, például az általam is használt JavaScriptet, HTML-t, CSS-t és Node.js-t.

A VS Code beépített támogatást tartalmaz az intelligens kódkezelésre az IntelliSense segítségével, mely nem csak a kód kiegészítésében és refaktorálásában, de a navigációban is hasznos lehet, főleg egy összetettebb projekt kapcsán. Ezenkívül beépített támogatást tartalmaz Node.js hibakereséshez és JavaScript fejlesztéséhez. Ezeken felül lehetőségünkben áll bővítményeket hozzáadni, mint például *Live Server*, illetve rendelkezik beépített parancssorral, melyből így futtathattam a Node.js kódomat. [18]

Live Server

A Visual Studio Code Elő Szerver (Live Server) bővítménye hihetetlenül hasznos a web-fejlesztés folyamat során. Lehetővé teszi, hogy egy kattintással elindítsunk egy HTML dokumentumot és lefuttat egy dinamikusan generált szerveret, amely során megnyit egy böngésző ablakot és feltölti az általunk kódolt elemekkel. A fájlön végrehajtott bármilyen gyakorlati módosítást követően a böngésző ablakát újratölti és azonnal végrehajtja a változtatásokat.

4.2.4. Git

A Git ingyenes, nyílt forráskódú verziókezelő rendszer, amelyet a kis és közepes projektek hatékony kezelésére használnak. A Git a forráskód változásainak nyomon követésére szolgál, lehetővé téve több fejlesztő számára, hogy együtt dolgozzon a fejlesztésen. Lényeges funkciója a verziókezelésen felül a Git folyamat elágazási stratégia (flow branching strategy), amely hierarchikus kapcsolatot hoz létre a különböző könyvtárak és fájlok között. Ezáltal a fejlesztési folyamatokat külön ágakon lehet végezni, így elkerülhető az esedleges hibás kódolás terjesztése többi fejlesztő számára. Néhány fontosabb információ a Git-tel kapcsolatban:

- Az elosztott verzióvezérlő eszköz a forráskód-kezelésre szolgál
- Lehetővé teszi több fejlesztő együttműködését
- Több ezer párhuzamos ágán keresztül támogatja a nemlineáris fejlesztést
- Nyomon követi az előzményeket, illetve biztonsági mentéseket készít [19]

Az általam készített alkalmazást regisztráltam a GitHub felhő alapú szolgáltatásán, amely tárolja a szükséges kódokat és fájlokat. Fejlesztésem során több ágon (branch) dolgoztam, így a forráskód csakis abban az esetben került fel a fő (main) ágamra, ha a mellékágakon tesztelési folyamatok után sem találtam hibát.

4.3. Kliens

Először a Kezdőlap és Grafikonrajzoló oldalak funkcióival kezdtem a tervezést. Az *index.html* fájlban létrehoztam a navigációhoz tartozó paneleket, majd CSS felhasználásával megalkottam a kinézetét és azt a kritikus funkciót, hogy a navigáció mindig a képernyő tetején legyen, és kövesse annak a mozgását, így a felhasználó könnyedén tud lapozni az oldalak között. Mivel az oldalamat több felbontásra terveztem a reszponzivitás szemléletében, következésképpen itt is kulcs szerepet tölt be. Mivel a betűk eléggé nagyok, ezáltal bizonyos felbontás mellett egymásra halmozódnának, ezt elkerülve, ha az képernyő mérete kisebb mint, 1099 px, akkor a navigáció összezsugorodik egy kis gombbá és a soros megjelenítést felváltja az oszlopos megvalósítás (a 4.3).

```
const bar = document.getElementById('bar');
const nav = document.getElementById('navbar');
const close = document.getElementById('close');

if (bar) {
  bar.addEventListener('click', () => {
    nav.classList.add('active');
  })
}

if (close) {
  close.addEventListener('click', () => {
    nav.classList.remove('active');
  })
}
```

4.3. ábra. A navigációs sáv összezsugorítása és kinyitása

További fejlesztés volt még az oldalak HTML vázának megvalósítása. Szükséges szempont volt, hogy a különböző oldalak más elemekkel töltsen fel, de alapjában véve egységes weboldal látszatát keltsék. A különböző részeket `<section>` címkével (tag) választom el, így átláthatóbb számomra, illetve néhány szekciót egyedi azonosítóval (id) láttam el, hogy működésüket, vagy külsejüket tudjam applikálni. A gombok és választható elemeknél a **Bootstrap** könyvtár beépített függvényeit használtam. A dátumokhoz tartozó gombokat jQuery függvény segítségével kiszelektálom és megjelenítek hozzájuk egy naptárat is. A naptáron lehetőség van kattintással kiválasztani a napot, vagy váltani a hónapok és évek között, illetve ha mégsem a megfelelő dátum lett kiválasztva, akkor a törlés gomb megnyomásával hatálytalanítjuk a bevitt adatot (a 4.4).

```

$(function () {
  $("#min-datepicker").datepicker({
    autoclose: true,
    clearBtn: true,
    format: 'yyyy-mm-dd',
    startDate: "2022-05-31",
    endDate: '2023-05-31'
  });
});

let startDateGraph = new Date();

$('#btn').on('click', function(){
  let minDate = new Date($('#min-date').val());
  let sday = minDate.getDate();
  let smonth = minDate.getMonth() + 1;
  let syear = minDate.getFullYear();

  startDateGraph = [syear, smonth, sday].join('-');
})

```

4.4. ábra. A dátumválasztó működése kezdő dátumra

A kliens egy API-n keresztül (a 4.15), HTTP kérésekkel kommunikál a szerverrel. A kommunikáció úgy néz ki, hogy a **Grafikonrajzoló** oldalon található gomb megnyomásával inicializáljuk a rajzoló egy aszinkron függvény használatával. A függvény csak akkor hívódik meg, ha a felhasználó beállította a programhoz elengedhetetlen adatokat, ilyen adatok például a grafikon típusa, valamelyik befektetési alap és az árfolyamok közül az egyik kiválasztása, illetve a kezdeti és záró dátumok. Hogy ne kelljen minden egyes alkalommal minden adatot újra beállítani, így alapértelmezetten kiválasztottam az első három opcióból egyet.

Az aszinkron függvény megvizsgálja, hogy a gomb felett található kettő dátum input fel van töltve adatokkal, tehát a felhasználó kiválasztotta a kezdő és a záró dátumot (a 4.5).

```

function missingDates(){
  let emptydate = 0;
  for (const sdate of startDateGraph) {
    if(sdate === "N"){
      emptydate = 1
      break;
    } else {
      emptydate = 0
    }
  }

  for (const edate of endDateGraph) {
    if(edate === "N"){
      emptydate = 1
      break;
    }
  }

  return emptydate
}

```

4.5. ábra. A missingDates függvény. megvizsgálja, hogy nincs-e üres dátum mező

Abban az esetben, ha üresek, akkor egy figyelmeztető üzenet jelenik meg, ami felhívja a figyelmet a dátumok kitöltésének követelményére. További feltételek még, hogy a kezdeti és záró dátum nem eshet ugyan arra a napra, illetve a záró dátum nem lehet kisebb, mint a kezdeti dátum, hiszen az API hívásban problémát okozna a hibásan megadott időintervallum.

```
startDateGraph = [syear, smonth, sday].join('-');
endDateGraph = [eyear, emonth, eday].join('-');

if ((Date.parse(endDateGraph) <= Date.parse(startDateGraph))) {
  document.getElementById("max-date").value = "";
  endDateGraph = "Nan";
  textType = 1;
}
```

4.6. ábra. A kód felülvizsgálja a záró dátumot

Amennyiben minden mező sikeresen ki lett töltve, a program "GET" kéréssel lekéri az adatokat a szerverről és átadja a weboldal címét, a felhasználó által kiválasztott token-t és a kezdeti, illetve a végső dátumokat.

```
const url = 'http://localhost:8080/api?ticker=${selectedInvest}&
            &start_date=${startDateGraph}&end_date=${endDateGraph}';
```

Ha sikerült a kérés és nem merült fel semmilyen hiba, akkor a függvény Promise-szal (ígéret) jelzi, hogy minden adat elérte a weboldalt (a 4.8). Ezen adatokat a program JSON formátumban küldi el, csakúgy, mint ahogy backenden tárolva vannak, azzal a különbséggel, hogy csak azok az adattagok jöttek fel, amit a felhasználó kiválasztott.

```
async function fetchStockValues(url){
  return fetch(url)
    .then(response => {
      if (!response.ok) {
        throw Error(response.statusText);
      }
      return response.json();
    }).catch(error => {
      console.log(error);
    })
}
```

4.7. ábra. Backendről felküldött adatokat adja át a frontendnek JSON formátumban

Most, hogy minden kért adat sikeresen elérte a klienst, következhet a diagram struktúrájának összetétele. Elsőként egy függvény felel azért, hogy meg tudjuk milyen típusú adatok lettek kiválasztva. Ez a függvény mindig meghívódik gombnyomás előtt, hogy inicializálja a gárfot, ha esetleg az alapértelmezett opciók helyett más lehetőséget választott volna a felhasználó, majd utána gombnyomásra aktiválódik a folyamat újra.

```

const radioButton = document.querySelectorAll('input[name="type"]');
const checkedButtons = document.querySelectorAll('input[name="invests"]');
const checkedValues = document.querySelectorAll('input[name="values"]');

function getTypes(){
  for (const radioButton of radioButton) {
    radioButton.addEventListener('change', showSelectedType);
  }
  for (const checkedButton of checkedButtons) {
    checkedButton.addEventListener('change', showSelectedInvest);
  }
  for (const checkedValue of checkedValues) {
    checkedValue.addEventListener('change', showSelectedValues);
  }
}

function showSelectedType() {
  if (this.checked) {
    selectedGraph = this.value;
  }
}
function showSelectedInvest() {
  if (this.checked) {
    selectedInvest = this.value;
  }
}
function showSelectedValues() {
  if (this.checked) {
    selectedValue = this.value;
  }
}

```

4.8. ábra. A getTypes függvény a diagram típusainak változását figyeli

A grafikon megvalósításához a Chart.js definiált könyvtárát használom fel. Ehhez HTML oldalon szükséges <canvas> címkét létrehozni, hiszen két dimenziós ábrát szeretnék megvalósítani. Egyedi azonosítóval láttam el a canvas elemet, amire a *script.js* fájlban hivatkozok. A grafikonnak három fontos paraméterrel rendelkezik, úgy mint a típusa (types), adathalmaza (data) és opciói (options).

A típusát "switch" választási mechanizmussal érem el aképpen, hogy a felhasználó melyik befektetést választotta ki az oldalon, úgy az ahhoz tartozó megnevezést adom át a grafikonnak. Az adathalmazt egy függvény segítségével töltöm fel. Az API hívás során megkapott JSON fájlban végigiterál a függvény és egy tömbben adja vissza az adatokat amivel a grafikon tud dolgozni (a 4.9). Az opcióit JSON formátumban használom. A grafikon felett megjelenített időszávot függvény során valósítom meg, ami mindig a felhasználó által kiválasztott aktuális dátumot jeleníti meg, feltéve ha a dátumok megfelelnek a hibaellenőrzésnek. Animációval teszem látványosabbá a diagrammok megjelenítését.

```

function getData() {
  return data = {
    labels: stockValues.map(stockValue => stockValue.date),
    datasets: [{
      label: getLabel(),
      data: stockValues.map(stockValue => stockValue[selectedValue]),
      borderWidth: 1,
    }]
  }
}

```

4.9. ábra. Az API hívás során megkapott adathalmazon iterál végig a getData függvény

Az alkalmazás struktúrája úgy működik, hogy minden alkalommal amikor a "Mehet" gombot megnyomja a felhasználó az addig használt gráf úgymond megsemmisül, és helyette a program új gráfot készít a megadott paraméterekkel (a 4.10). Természetesen a feltételvizsgálat itt is fontos szerepet játszik, tehát ha nincs megadva dátum, akkor hiba üzenet fog megjelenni és a gráf törlődik.

```
function makeNewChart() {  
  let chartStatus = Chart.getChart("myChart");  
  if (chartStatus !== undefined) {  
    chartStatus.destroy();  
  }  
}
```

4.10. ábra. A függvény megvizsgálja, hogy létezik gráf, ha igen, akkor törli azt

4.4. Szerver

Ahhoz, hogy a szoftver működjön telepítenünk kell a megfelelő technológiákat. Ahogy már a **Bevezetőben** ismertettem, a kliens-szerver architektúra megfelelő az alkalmazásra. A kliens egy böngészőben futtatott frontend, ami API hívásokon keresztül tud kommunikálni a backend szerverrel. Hogy az architektúra működjön a számítógépünkön először telepíteni kell a Node.js-t, és az Express-t. A Node.js-t egyszerűen le lehet tölteni a hivatalos honlapról [13]. Windows-os felhasználóként az ehhez az operációs rendszerhez készített 64-bites setup segítségével telepítettem fel az alkalmazást. Hogy megbizonyosodjunk a *Node.js* sikeres letöltésről (a 4.11) bemutatok egy próba tesztet.

```
C:\Users\Bencze Zsombor>x = 5  
'x' is not recognized as an internal or external command,  
operable program or batch file.  
  
C:\Users\Bencze Zsombor>node  
Welcome to Node.js v18.14.2.  
type .help for more information.  
> x = 5  
5  
> x + 5  
10
```

4.11. ábra. Sikeresen telepített node demó

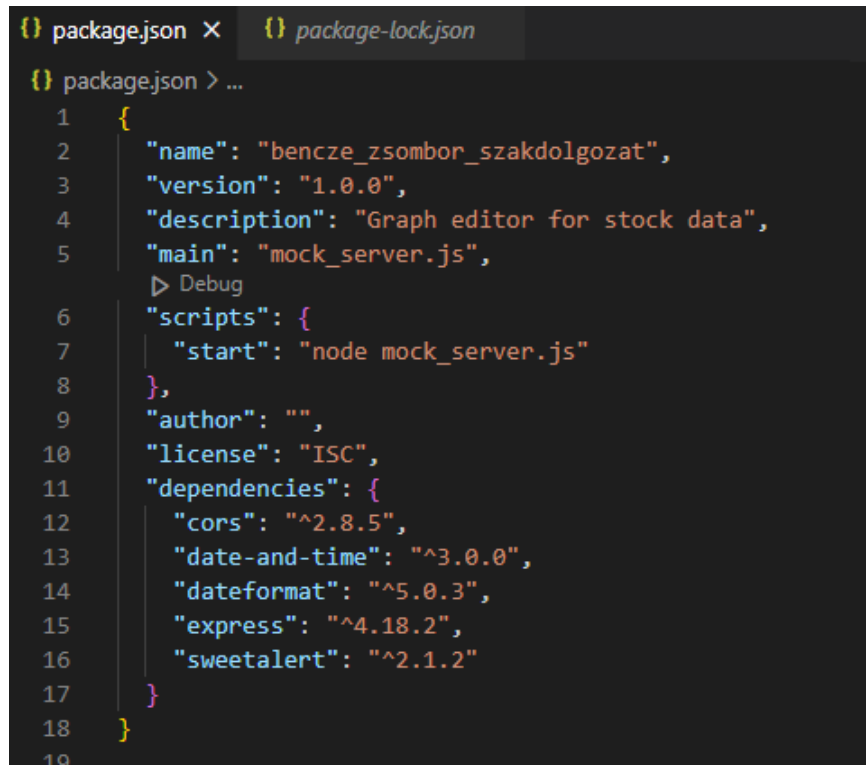
Miután megbizonyosodtunk, hogy sikeresen telepítettük a Node.js aktuális verzióját, használjuk a Visual Studio Code, vagy a Windows beépített terminálját és írjuk be a következő parancsot:

```
npm init
```

A kód lefutása után lehetőségünkben áll csomag nevet (package name), verziót (version), leírást (description), belépési pontot (entry point), teszt parancsot (test command), git adattárat (git repository), kulcsszavakat (keywords), szerzőt (author) és licenszt (license) megadni, de én azt javaslom az enter megnyomásával mindent hagyjunk

meg a rendszernek, hogy automatikusan töltsse ki helyettünk. Ha mindent jól csináltunk akkor létre kellett jönnie egy *package.json* fájlnek (a 4.12). A telepítésünk következő lépése a az *Express.js* installálása, amit szintén a terminálon keresztül tudunk letölteni az alábbi paranccsal:

```
npm install express —save
```



```
{
  "name": "bencze_zsombor_szakdolgozat",
  "version": "1.0.0",
  "description": "Graph editor for stock data",
  "main": "mock_server.js",
  "scripts": {
    "start": "node mock_server.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "date-and-time": "^3.0.0",
    "dateformat": "^5.0.3",
    "express": "^4.18.2",
    "sweetalert": "^2.1.2"
  }
}
```

4.12. ábra. Az alkalmazáshoz készített package.json

Észrevehetjük, hogy egy új mappa *node_modules* ékelődött be a fájlaink közé. Ez egy könyvtár, amelyet az npm hozott létre, és egy módja annak, hogy nyomon kövesse a helyileg telepített csomagokat. A felhasznált csomagok nevei és verziószámai, mind a *package.json* nevű fájlban találhatóak így, ha a GitHubra akarom menteni a projekteket, akkor időt sprólok, hiszen nem nekem kell egyesével több ezer fájlt feltöltenem a felhőbe, mert mindegyik megtalálható lesz a JSON-ben. Az ábrán (a 4.12) látható dependenciák (*dependencies*) tartalmazza mindazokat a telepítéseket amiket az alkalmazáshoz használtam.

A telepítést követően létre kell hoznunk a webszervert portját, amit az Express csomag segítségével létrehoztam. A szerver alapját úgy állítottam be, hogy a 8080-as porton figyeljen, azaz a böngészőben a <http://localhost:8080> címen legyen elérhető (a 4.13).

```
const express = require('express');
const app = express();
const PORT = 8080;

app.listen(PORT, function(err){
  if (err) console.log(err);
  console.log("Server listening on PORT", PORT);
});
```

4.13. ábra. Ezen a porton lehet elérni a szervert

A szerver működtetéséhez már csak egy lépés van hátra, mégpedig létrehozni magát a tényleges HTTP szervert (a 4.14). A kódot úgy írtam meg, hogyha a szerver 200-as státuszkódot kap, tehát nem történt hiba a szerver elérésében, akkor a "req" argumentumot felhasználva a kód lekéri a kienstől az URL-t, továbbá meghív egy "getData" nevű függvényt, amit később fogok ismertetni. [20]

```
const http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.write(getData());
  res.end();
}).listen(8080);
```

4.14. ábra. Szerver létrehozása

A szerver REST API-n keresztül hallgat a kérésekre (a 4.15). Ezen az API-n szolgálja fel a statikus tőzsdei adatokat, amiket backend/data mappába mentettem el és Postman (a 4.2) segítségével "GET" kéréssel értem el az adatokat. Mivel az alkalmazás hat különböző befektetést ismerteti, így hat API hívást intéztem, amelyeket JSON fájlban tároltam, hogy könnyen feltudjam küldeni őket a kliensre.

Mivel a szerver és a kliens nem egy URL címen futnak, ezért globálisan össze kell őket hangolni, amihez szükség van egy keresztező erőforrás megosztásra (Cross-Origin Resource Sharing) röviden: CORS.

```
let cors = require('cors')

app.use(cors({
  origin: '*'
}));
```

```

app.get('/api', function(req, res){
  console.log('name: ' + req.query.ticker);
  const ticker = req.query.ticker;
  const startDate = req.query.start_date;
  let endDate = req.query.end_date;
  if (!endDate) {
    endDate = date.format(new Date(), "YYYY-MM-DD");
  }
  let response = 'name: ' + req.query.ticker + '\r\n' +
    'start date: ' + req.query.start_date + '\r\n' +
    'end date: ' + req.query.end_date + '\r\n';
  const dataPart = getData(ticker, startDate, endDate);

  res.send(JSON.stringify(dataPart));
});

```

4.15. ábra. Szerverről intézett API hívás

Most, hogy a szerver és a kliens is sikeresen kommunikál egymással, már csak egy lépés van hátra, mégpedig a felhasználó által igényelt adatok elérése. A weboldalon felhasználónak lehetősége adódik kiválasztani, hogy melyik befektetést szeretné megjeleníteni, illetve kitud választani két dátumot, miszerint mettől (kezdeti dátum input) meddig (záró dátum input) szeretné megjeleníteni a grafikonrajzolóval a kívánt gráfot. Majd ezeket a dátumokat a kliens leküldi a szervernek, amely megkapva feldolgozza és a kért paraméterek alapján megszűri az adott JSON-t, és úgy küldi vissza a kliensnek (a 4.16).

```

function getData(tickerName, startDate, endDate) {
  let obj = JSON.parse(fs.readFileSync('data/' + tickerName + '.json', 'utf8'));
  console.log(JSON.stringify(obj));
  const result = obj.filter(x => (x.date >= startDate) && (x.date <= endDate));
  return result;
}

```

4.16. ábra. Frontend-ről megkapott adatok alapján kiválasztja a kezdeti és záró dátumon belüli értékeket

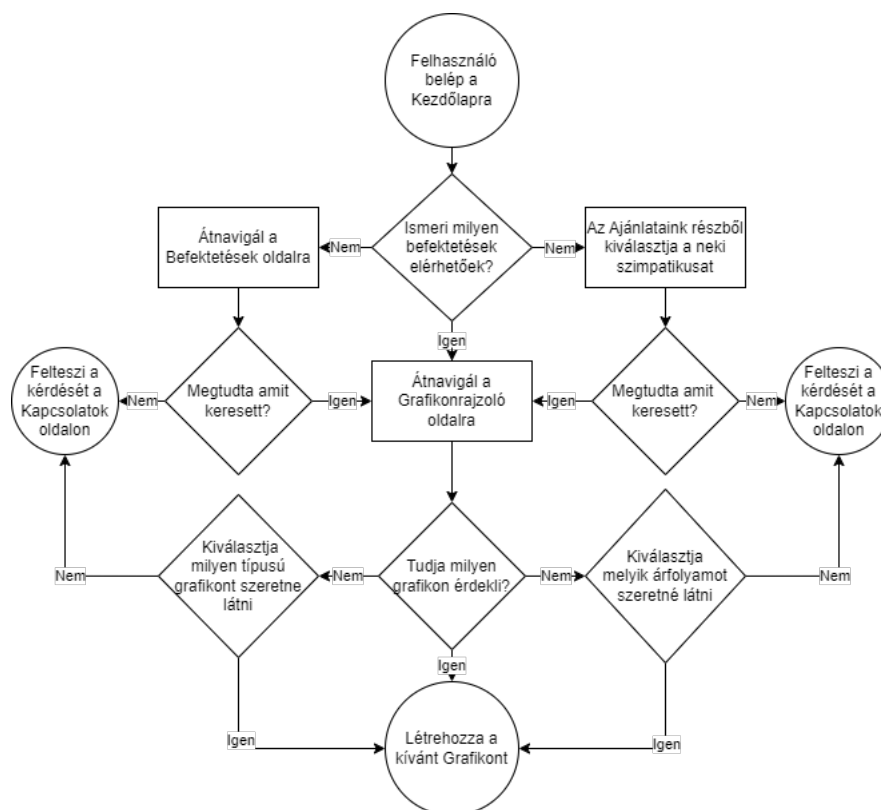
5. fejezet

Alkalmazás bemutatása

Az alábbi fejezetben az alkalmazást fogom részletesen ismertetni a felhasználóval. Először a szerkezeti felépítését fogom bemutatni, majd kitérek minden fontosabb oldalra és a kiegészítő elemekre. Miután prezentáltam a weboldalt áttekintem miért lettek implementálva a különféle funkciók, illetve kitérek arra, hogy milyen fejlesztési lehetőségek elérhetőek a jövőben.

5.1. Alkalmazás felépítése

Az alkalmazást úgy építettem fel, hogy a látogató elsőként a Kezdőlapra találja magát. Innen lehetősége van a további három másik főoldal bármelyikére kalauzolni magát, amennyiben szeretné (az 5.1).



5.1. ábra. Alkalmazás állapotdiagramja (Forrás: [21])

5.2. Alkalmazás panelei

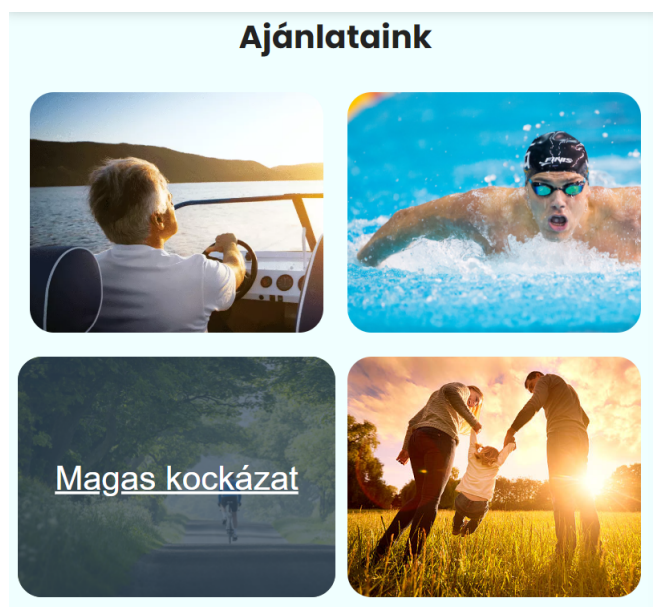
Az alkalmazás megalkotásánál az alapvető szemléletem az volt, hogy szaktudásomat megjelenítéssel és funkcionalitással szeretném prezentálni. Piackutatásaim során sokat inspirálódtam különböző grafikonrajzoló weboldalakról, főként abból a szempontból, hogy milyen paneleket és oldalakat érdemes megalkotni. Következésképpen a szoftver struktúráját négy fő oldalra és hat aloldalra terveztem meg.[22] A főbb oldalak a következők:

- Kezdőlap,
- Grafikonrajzoló,
- Befektetések,
- Kapcsolat.

5.2.1. Kezdőlap

Minden oldalnak szüksége van egy főoldalra, nálam ezt a szerepet a Kezdőoldal tölti be. Ugyan a felhasználó a forráskód birtokában az összes többi oldalt eltudja indítani a **Live Server** alkalmazásával, viszont a weboldal úgy lett felépítve, hogy az *index.html* elindításával kezdjen. Ez volt az első HTML fájl amit létrehoztam és a Kezdőlapról lehet elérni egy kattintással a másik három főoldalt is.

Minden oldalon egy általam "banner" (zászló) megnevezésű borítókép fogadja a látogatót, ezen borítókra több esetben is figyelem felkeltő szöveget írtam. A Kezdőlapon megjeleníték minden lényeges információt amit az alkalmazás kínál, továbbá lehetőség van egy kattintással átnavigálni a többi oldalra. Amit külön kiemelnék az animációk az Ajánlataink résznél, aminek a funkciója szintén a figyelemfelkeltés.



5.2. ábra. Kezdőlapon feltüntetett Ajánlataink részlet

5.2.2. Grafikonrajzoló

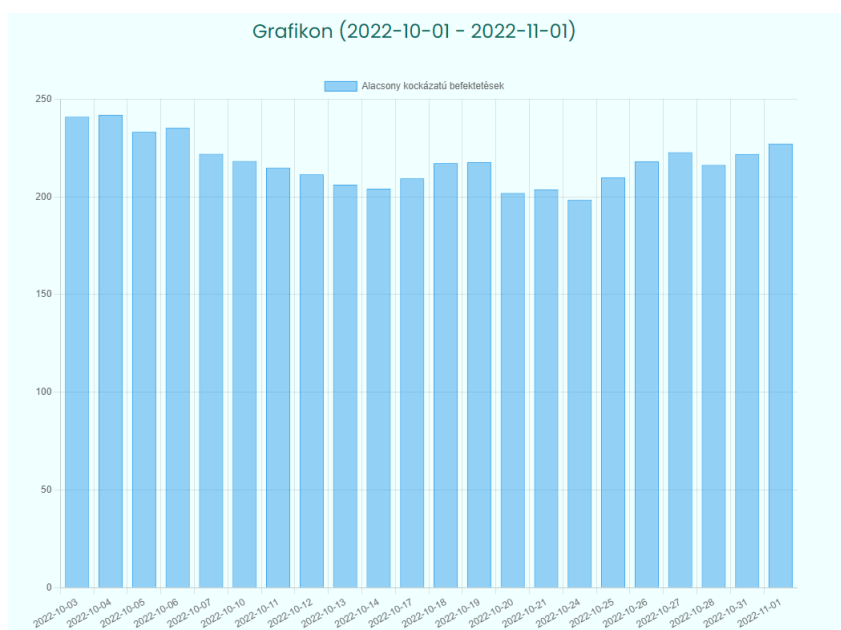
Ugyan a **Kezdőlap** hivatalosan a főoldal amivel elsőnek találkozik a látogató, mégis a szakdolgozatom fő eleme a Grafikonrajzoló oldal. A teljes projekt eköré a lehetőség köré épült, mivel a tőzsdei adatok sokasága emberi szemnek nem túl kedvező és átlátható, innen jött az ötlet, hogy az adatok perspektivikus ábrázolása sokkal jobban megfogja a célközönséget. Alapvetően a legtöbb ember vizuális beállítottságú és ezt a pszichológia tényét használtam fel a grafikonrajzoló megtervezésénél.

Megvalósítását tekintve a borítókép is egyértelműen jelzi, hogy az oldalon gráfokkal lehet találkozni. Ezek után egy kisebb bevezető szöveg következik, ami ismerteti hogyan lehet akkurátusan leolvasni a megjelenített adatokat. Természetesen a megjelenítés felépítése nem tér el az átlagtól, így akinek ez lenne a legelső ilyen témájú weboldal, sem fogja magát elvesztve érezni, hiszen a tankönyvek, hirdetések és a közmédia is ezen ábrázolást használja, amely információhalmazzal nagy valószínűséggel találkozott a felhasználó élete során. Ha mégsem így lenne, akkor eljött a megfelelő alkalom az alkalmazásom személyében.

Funkcionalitása úgy épül fel, hogy elsőnek lehetőségünk van kiválasztani milyen fajta gráfot szeretnénk látni. Ötféle opció áll rendelkezésre amelyeket már korábban ismertettem a második fejezet **Grafikonok története és típusai** részben. Mivel a vonaldiagram a legelterjedtebb gráf, így alapértelmezetten az került kiválasztásra, ha a felhasználó nem változtat az opción, akkor aképpen jelenik meg a gráf.

Következő adatsor a befektetési alapok. Jelenleg hat különböző típusú befektetési alap közül lehet választani, amiket a *Befektetések* alfejezetben fogok ismertetni. Hogy ne legyen túl sok adat egyszerre megjelenítve korlátoztam a választási lehetőségeket, miszerint egyszerre csak egyféle alapot lehet választani.

Végül a tőzsdei árfolyamok következnek amiket szintén ismertettem a második fejezet **Tőzsdei árfolyami** cikkében. Szintén ötféle választási lehetőség táru a felhasználó elé, amelyek közül az egyszerű átlátás végett egyet lehet választani.



5.3. ábra. Grafikonrajzoló által készített gráf példa

5.2.3. Befektetések

A Befektetések oldalnak a fő paradigmája az információ közlése és ismertetése. Az **Az Elméleti háttér** fejezetben már taglaltam, hogy a tőzsde és a befektetések elrettentő szavak azon személyeknek akiknek nincs átfogó gazdasági szakértelme, így ez az oldal azt a célt szolgálja, hogy bárki tudását gyarapíthassa. Jelenleg az oldalamon hat különböző befektetés elérhető, mindegyik befektetéshez külön aloldal tartozik, ahol részletes leírás, táblázat és éves nettó értékelés is található. Felépítését tekintve, hogy jobban meglehessen őket jegyezni mindegyik befektetés kapott saját borítóképet és fantázia nevet. A befektetések leírása általam készített kreáció amiket a fő inspirációként szolgáló **Aegon Biztosító Zrt.** hivatalos weboldaláról szereztem [22]. Mindegyik befektetés adathalmazza valós részvények adatait képezik, mint például a *Tempó 2 Andante* alaphoz a Google részvényeinek alapszik. Ezen adatokat a már korábban említett EOD oldalról gyűjtöttem össze API hívásokat alkalmazva. [17] A Befektetések oldalon található hat befektetési alap:

- **Tempó 2 Andante** (Alacsony kockázatú befektetés),
- **Tempó 5 Moderato** (Közepes kockázatú befektetés),
- **Tempó 8 Allegro** (Magas kockázatú befektetés),
- **Abszolút hozamú befektetési alap** (Családi csomag),
- **Feltörekvő Európa Kötvény befektetési alap** (Európai portfólió),
- **Megatrend Részvény befektetési alap** (Megújuló energia). [22]

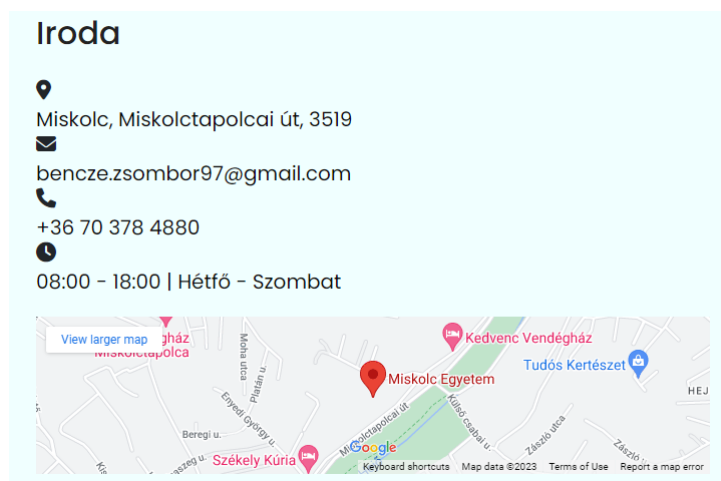


5.4. ábra. A hat befektetési alap közül a harmadik.

5.2.4. Kapcsolat

Amikor a fejlesztési stratégiám még a tervezési fázisban (a 3.1) járt, akkor nem fordítottam nagyobb figyelmet erre a panelre, mert nem tartottam fontosnak. Viszont a konzulensem által is erősen javasolt piackutatási folyamat során meglepő információra bukkantam. A legtöbb blogban és elemzésben elengedhetetlen elemnek tartják a **Kapcsolat** oldalt. "A kapcsolati adatok könnyű megtalálása az egyik legfontosabb dolog, hogy a weboldalad megbízhatóvá váljon. Az email, telefon vagy élő chat elérésedet helyezd el minden aloldalon." [23]. Tehát kétség sem fér hozzá, hogy az oldal kötelező eleme legyen az alkalmazásomnak, viszont felmerült bennem a kérdés, hogy mégis milyen módon állítsam össze?

Nos erre a kérdésre hamar megtaláltam a választ, a Kapcsolat oldal kivitelezését aszerint fogom megalkotni, mint ahogy egy felhasználó elvárna. Elsőként kiválasztottam egy figyelemfelkeltő és megragadó képet ami egyértelműen szimbolizálja milyen elemek találhatóak. Továbbiakban arra fókuszáltam, hogy a látogató számára minden olyan szükséges metaadatokat (az 5.5) jelenjenek meg amivel kapcsolatba lehet lépni velem. A Kapcsolat címmel ellátott blokkban megadtam a "Fő iroda" adatait, amiket a Font Awesome által kiválasztott ismerős ikonokkal jeleztem. Kiegészítettem még egy beszúrt élő Google térképpel, ami az iroda "központját" célszerű mutatni, illetve elhelyeztem még egy üzenet küldő formot, ha bármilyen kérdés felmerülne a felhasználóban.



5.5. ábra. Részlet a kapcsolat oldalról

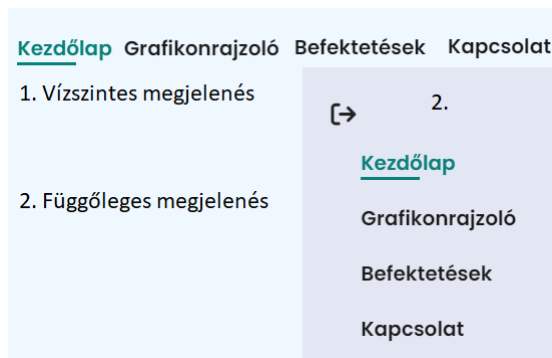
5.2.5. Design megtervezése

Először felmértem, hogy mely oldalak lesznek a weboldalam legfontosabb elemei, majd úgy alakítottam ki a szerkezetüket, hogy egységesen passzoljon mindegyikre. Témának zöldes-kékes színvilágot választottam, ezért lett égszínkék a háttérszín. A szövegek színe megmaradt a fekete és annak árnyalatainak, mint például: #222 RGB kódú világosabb feketeszín. Továbbá a gombok és feliratok zöldes színt kaptak, amihez a leggyakrabban használt #088178 RGB kódú látványt használtam.

5.2.6. Navigáció

A navigációt úgy terveztem meg, hogy a fő oldalak között bármikor lehessen általa váltani. Mindig a képernyő tetején helyezkedik el és követi annak mozgását, így a navigálás felhasználó barát és könnyed. Ha a képernyő piksel mérete kisebb a monitorénál, akkor vízszintes helyzetkedését függőlegesé váltja fel, amit egy gomb megnyomásával lehet előhozni és elrejteni.

A weboldalakon kétféle menürendszer használatos: vízszintes és függőleges. (az 5.6)



5.6. ábra. Navigációs sáv két típusú megjelenítése

5.2.7. Lábléc

"Talán furcsán hangzik, de minden weboldal egyik legfontosabb eleme a lábléc" [23]. Biztos, hogy nem ez a legfigyelemre méltóbb alkotóeleme a weboldalnak, viszont ez az a hely amit a látogató felkereshet, ha információra van szüksége.



5.7. ábra. Lábléc

5.2.8. Logó

A logó az alkalmazás bal oldalán helyezkedik el, mind a navigációs sávon, mind a láblécen. A logóra kattinva bármely oldalról eljuthat a felhasználó a Kezdőlapra. A logó megjelenését én készítettem, a dizájn célja az volt, hogy a látogató számára egyértelműen mutassa, ez egy részvényekkel foglalkozó weboldal.

5.3. Alkalmazás áttekintése

Ebben a szekvenciában szubjektív módon szeretném bemutatni, mi indokolta a weboldalam létrejöttét. Olyan szempontokat fogok felsorolni amik az én alkalmazásomban megtalálhatóak, de más általam felmért weboldalakban hiányoltam.

5.3.1. Más oldalakkal összehasonlítás

Ahogy már említettem a lista teljesen szubjektív és az oldalon található funkciókat szeretném megindokolni, hogy miért lettek lefejlesztve és mi volt velük a célom.

Ami az én oldalalom megtalálható:

- Könnyű navigálás, hiszen a főoldalról is elérhető bármelyik oldal, illetve a navigációs sáv követi a képernyő mozgását.
- Képekkel ellátott befektetések, hogy befogadhatóbbak legyenek.
- Többfajta diagram típus közül lehet választani.
- Többféle árfolyam is választható.

Ami az én oldalalom nem található:

- Vonaldiagramnál több gráf választható,
- Bejelentkezési felület,
- Nyelvesítés,
- Adatok letöltése.

5.3.2. Miért fontosak a tőzsdét elemző weboldalak?

A tőzsdére történő befektetés jövedelmező vállalkozás lehet, de alapos kutatást és elemzést igényel a megalapozott döntések meghozatalához. Az egyik értékes erőforrás, amely jelentősen segítheti a befektetőket a döntéshozatali folyamatban, a grafikonrajzoló weboldalak. Átfogó adatvizsgálat, pénzügyi kimutatásokhoz és iparági betekintésekhez szokás használni. A tőzsde rendkívül dinamikus, az árfolyamok különböző tényezők miatt gyorsan ingadoznak. A grafikonrajzoló segítségével folyamatosan figyelemmel kísérhetők a piaci viszonyok, és azonnali betekintés nyerhető. A technológiai fejlődésnek és a különféle platformok elérhetőségének köszönhetően minden eddiginél egyszerűbb megtalálni a releváns befektetéseket.

5.3.3. Hogyan lehet felhasználni adatok elemzéséhez?

A grafikon gyakori módszer az adatok összefüggéseinek vizuális szemléltetésére. A grafikon célja, hogy olyan adatokat mutasson be, amelyek túl sok vagy bonyolult ahhoz, hogy a szövegben és kisebb helyen megfelelően leírhatók legyenek. Ha az adatok kifejezett trendeket mutatnak, vagy a változók közötti kapcsolatokat tárják fel, grafikont kell használni. A gráfokon használt egyik gyakori elemzési technika a **részvényelemzés**.

5.3.4. Részvényelemzés

A részvényelemzés népszerű a technikai elemzés, a múltbeli részvényárfolyam- tevékenységre támaszkodva a jövőbeli árfolyamtevékenység előrejelzésében. Gyakori módszer a befektetők és kereskedők számára vételi és eladási döntések meghozatalára. A múltbeli és jelenlegi adatok tanulmányozásával és értékelésével a befektetők és a kereskedők megalapozott döntések meghozatalával próbálnak előnyt szerezni a piacokon. Az alábbiakban tárgyalt elsődleges módszerekben a befektetők pénzügyi kimutatásokat, részvényárfolyamokat, piaci mutatókat vagy iparági trendeket használnak befektetési döntéseik meghozatalához. [24]

6. fejezet

Összefoglalás

A szakdolgozatom pénzügyi adatok elemzéséhez használható interaktív megjelenítő weboldalt mutatott be elméleti és gyakorlati példákkal egyetemben, valamint a hozzá elkészített alkalmazást. A dolgozat felépítése jellemzően az elméleti tudástárból a gyakorlati megvalósítás felé terjedt. Az elméleti részben feltártam a Grafikonrajzoló lapon elérhető diagrammok típusait, ismertettem tőzsdei alapfogalmakat, illetve kifejtettem az oldalon található kulcsszavakat, amelyek tudása elengedhetetlen. Ezek után az általam írt program tervezésének lépéseit mutattam be, majd a hozzá felhasznált technológiákat részleteztem.

Az alkalmazás fejlesztése Visual Studio Code környezetben történt, munkámat elősegítette a Live Server bővítmény által kínált kényelmes élő szerver szolgáltatás. A szerverhez használt kódokat Node.js nyelven írtam meg és az adatokat az EOD Historical Data nevezetű webhely szolgáltatta. Mivel az portálon fellelhető teljes adatkészlet nem elérhető ingyenes verzióban, ezért ezt a problémát úgy abszolváltam, hogy Postman alkalmazás segítségével lekértem a hat befeketetésem adatait és külön json fájlban mentettem el őket. Az alkalmazás ezen adatok 1 éves halmazával tud operálni.

A klienst JavaScript és jQuery nyelveken vittem véghez, emellett HTML elemekkel építettem fel a sémáját, továbbá CSS és Bootstrap komponensekkel vittem véghez a megjelenésének kialakítását. A gráfok szerkezetében a Chart.js előre definiált könyvtárát vettem igénybe és úgy alakítottam, hogy prominens legyen a látogató számára. Úgy gondoltam, hogy ne kelljen mindig végigkattintani a diagrammok adatait, ezáltal a Grafikonrajzoló három opciójának alapértelmezett választásként adtam meg, amiket természetesen lehet kombinálni a felhasználó igénye szerint. A grafikon megjelenésének nélkülözhetetlen részei a kezdeti és záró dátumok, ennek okán csakis akkor jelenik meg ha a vizsgálati feltételeknek eleget tévő paraméterek kerülnek kijelölésre. Ehhez szükséges a szerver és kliens kommunikációja amit API hívásokon keresztül valósítottam meg.

A kész szoftver legnagyobb előnye, hogy önálló weboldalként is képes megállni a helyét. Struktúrája modern webfejlesztési módszerek szerint készült, tehát könnyen kezelhető, gyorsan működik és interaktív. A program a grafikonok változatos megjelenítésével, emellett többféle árfolyam lehetőségével szándékozik kiemelkedni a többi hasonló pénzügyi adatokat elemző alkalmazások közül.

Véleményem szerint sikerült egy működő weboldalt létrehoznom, amihez néhány továbbfejlesztési lehetőség ötletével álltam elő.

6.1. Továbbfejlesztési lehetőségek

A szakdolgozatom készítése alatt rengeteg ötlet és koncepció jutott eszembe, de mivel a rendelkezésre álló idő véglleges, így korlátoznom kellett magam a fontosabb funkciók megalkotására. Az első és talán legszembetűnőbb opció ami nincs a szoftveremnél az adatbázis. A Tervezési fázisnál még szerepelt, de később elvetettem az ötletet, mert nem éreztem feltétlenül szükségesnek. Döntésemet az is indokolta, mint ahogy a dolgozatom címében is szerepel "Interaktív megjelenítő eszköz", így az energiaforrást a grafikonrajzolóra fordítottam. Viszont ha mondjuk a tanulmányaim során megismert MySQL adatbázist bevezettem volna, akkor az ötödik fejezetben említett hiányzó funkciók közül kettőt kipipálhatok.

Amivel még lehet emelni az oldal színvonalát, ha több befektetés áll rendelkezésre, és egyszerre nem csak egy alapot lehet kiválasztani. Ezen funkció nagyon átgondolt struktúrát igényel, mert ha több alapot is ki lehet választani, akkor mindegyik megjelenítési típusnál le kell tesztelni miként működik. Hiszen ami mondjuk működőképes egy vonaldiagramnál, az nem biztos, hogy hasonlóképpen működne a kördiagramnál. Tehát alaposan kell tanulmányozni a diagrammok típusait, hogy milyen szabályosságok érvényesek a különféle típusokra és a kellő háttérinformáció megszerzése után úgy kellene megtervezni a rendszert, hogy minden elemre működjön. Talán ezért is hiányzik a rangosabb oldalakról a többféle típus kiválasztása.

Amit még a kutatásaim során néhány oldalon felfedeztem, az a bejelentkezés és profil részleg kialakítása. A szubjektív véleményem, hogy nem tartom túl szükséges elemnek egy ilyesfajta weboldalon, de példának okáért egy alternatívaként arra hasznos lehet, hogy a kedvenc részvényeket nyomon lehessen követni.

6.2. Summary

My thesis presented an interactive display website that can be used for analyzing financial data, together with theoretical and practical examples, as well as the application prepared for it. The structure of the thesis typically extended from the theoretical knowledge base to the practical implementation. In the theoretical part, I explored the types of diagrams available on the Graph Drawing tab, explained basic stock concepts, and explained the keywords on the page, of which knowledge is essential. After that, I presented the steps of designing the program that I wrote, and then detailed the technologies used for it.

The application was developed in the Visual Studio Code environment, my work was facilitated by the convenient live server service offered by the Live Server extension. The code that used for the server is written by Node.js and the data was provided by the EOD Historical Data website. Since the complete data set found on the portal is not available in the free version, I solved this problem by using the Postman application to retrieve the data of my six blackouts and save them in a separate json file. The application can operate with a 1-year set of these data.

I implemented the client in JavaScript and jQuery languages, in addition, I built its schema with HTML elements, and I implemented the design of its appearance with CSS and Bootstrap components. In the structure of the graphs, I used the predefined library of Chart.js and designed it so that it is prominent for the visitor. I thought that it would not be necessary to always click through the data of the charts, so I gave a default choice for the three options of the Graphic Designer, which of course can be

combined according to the user's needs. The start and end dates are indispensable parts of the appearance of the graph, which is why they only appear when parameters that meet the test conditions are selected. This requires server and client communication, which I implemented through API calls.

The biggest advantage of the ready-made software is that it can also stand its ground as an independent website. Its structure is made according to modern web development methods, so it is easy to use, works quickly and is interactive. The program intends to stand out from other applications that analyze similar financial data with the varied display of graphs and, in addition, the possibility of multiple exchange rates.

Irodalomjegyzék

- [1] Forrás: <https://www.portfolio.hu/prof/20181005/a-grafikon-ami-eleteket-mentett-299542>, *Felhasználtam a történet bevezetését, szerző: András Bence 2018. október 05.*
- [2] Forrás: <https://admiralmarkets.com/hu/education/articles/forex-basics/tozsde-grafikonok>
- [3] Forrás: <https://dataschools.education/history-of-graphs/>
- [4] Forrás: <https://www.uni-miskolc.hu/~wwwfemsz/exc5.htm>, *Segédlet az Excel 5.0 használatához előadásban található grafikon típusokat használtam. Dr. Szabó László: Segédlet az Excel 5.0 használatához Miskolc, 1997 © Szabó László*
- [5] Forrás: <https://www.mnb.hu/letoltes/tozsdei-alapok.pdf>, *Forrásomul szolgáltat az MNB által leírtak. Magyar Nemzeti Bank Tőzsdei-alapok.pdf 2020 február*
- [6] Forrás: <https://hold.hu/lexikon/reszveny-fogalma/>
- [7] Forrás: https://www.otpbank.hu/otpalapkezeslo/hu/Befektetesi_alapok/Tudnivalok_OTP_Alapkezeslo
- [8] Forrás: <https://docplayer.hu/7069114-Arfolyamok-miskolci-egyetem-mesterkepzes.html>, *Tóthné Klára Miskolci Egyetem, 2016*
- [9] Forrás: <https://promanconsulting.hu/mi-az-agilis-modszertan-legelterjedtebb-agilis/> *Az oldalon található Extreme Programming leírásából informálódtam*
- [10] Forrás: <https://www.bitdegree.org/learn/inline-css>
- [11] Forrás: <https://getbootstrap.com/docs/5.3/getting-started/introduction/>
- [12] Forrás: https://www.w3schools.com/bootstrap/bootstrap_grid_system.asp, *Felhasználtam az oldalon található képet*
- [13] Forrás: <https://nodejs.org/en/about>
- [14] Forrás: <https://en.wikipedia.org/wiki/JavaScript>, *Felhasználtam az oldalon található JavaScript, JQuery és JSON leírásokat*
- [15] Forrás: <https://www.chartjs.org/docs/latest/>
- [16] Forrás: <https://en.wikipedia.org/wiki/Chart.js>, *Felhasználtam az oldalon található összehasonlítást*

-
- [17] Forrás: <https://eodhistoricaldata.com/financial-apis/stock-etfs-fundamental-data-feeds/>, *Tőzsdei adatok API hívásokkal történő lekéréséhez szolgált célul az oldal*
- [18] Forrás: <https://code.visualstudio.com/learn>
- [19] Forrás: https://www.simplilearn.com/tutorials/git-tutorial/what-is-git#what_is_git, *Haifa Perveez munkássága*
- [20] Forrás: https://www.w3schools.com/nodejs/nodejs_http.asp, *Az itt látható kódrészlettel hoztam létre a szerveret*
- [21] Forrás: <https://app.diagrams.net/>, *Draw.io oldalon készítettem el a folyamat-ábrát*
- [22] Forrás: <https://www.vigam.hu/>, *Az oldal hatalmas inspirációt jelentettem a web-oldalam struktúrájában. Aegon Alapkezelő*
- [23] Forrás: <https://www.honlapdiszkont.com/milyen-a-jol-konvertalo-kezdolap/>, *Felhasználtam az oldalon található leírásokat és tippeket*
- [24] Forrás: <https://www.investopedia.com/terms/s/stock-analysis.asp>, *Felhasználtam az oldalon található "Understanding Stock Analysis" bejegyzést*

CD Használati útmutató

A mellékelt CD tartalma

- frontend mappa: tartalmazza a program kliens oldali forráskódját.
- backend mappa: tartalmazza a program szerver oldali forráskódját.
- szakdolgozat mappa: tartalmazza a szakdolgozat \LaTeX forráskódját.
- Szakdolgozat.pdf: tartalmazza a szakdolgozat PDF formátumban.
- GitHub link: tartalmazza az elkészített program **GitHub** linkjét.

A program futtatása

A programok használatához, szükséges a negyedik fejezetben említett függőségek telepítése.

- Töltsük le a Visual Studio Code-t a böngészőről és adjuk hozzá a live server bővítményt!
- Töltsük le a Node.js-t a hivatalos oldalról! (link: <https://nodejs.org/en#home-downloadhead>)
- Lépünk be a backend mappába és írjuk be az alábbi parancsokat a terminálba:

```
npm install  
npm run start
```

- Ha a terminál azt az üzenetet írta ki, hogy: "Server listening on PORT 8080", akkor már csak a Live Server van hátra,
- Jobb kattintás a frontend/index.html fájlra és "Open with Live Server".