



Eötvös Loránd Tudományegyetem
Informatikai Kar
Programozási Nyelvek és Fordítóprogramok
Tanszék

P4 programok gráf alapú statikus elemzése

Témavezető:

Tóth Gabriella
Doktorandusz

Szerző:

Tábi Zoltán
Programtervező Informatikus, BSc

Budapest, 2020

Tartalomjegyzék

1. Bevezetés	3
1.1. Alapfogalmak	3
1.1.1. P4	3
1.1.2. Vezérlésfolyamgráf	4
1.1.3. Adatfolyamgráf	4
2. Felhasználói dokumentáció	6
2.1. Program célja	6
2.2. Használat	6
2.2.1. Feltételek	6
2.2.2. Futtatás	6
2.2.3. Vizsgálható kód	7
2.3. Rendszer ismertetése	7
2.3.1. Gráfok ismertetése	9
2.3.2. Kezdő oldal	11
2.3.3. Gráf megjelenítő oldal	12
2.3.4. Elemzéseket végző oldal	17
2.3.5. Fájl oldal	21
2.3.6. Kényelmi funkciók	22
3. Fejlesztői dokumentáció	24
3.1. Fejlesztői környezet	24
3.1.1. Környezet felépítése	24
3.2. Program leírása	25
3.2.1. GraphForP4	25
3.2.2. Persistence	46
3.2.3. AngularApp	48
3.3. Tesztelés	57

3.3.1. Szerver oldal	57
3.3.2. Weboldal tesztelése	59
3.4. További fejlesztések.....	61
3.4.1. P4 fordítóprogram beépítése	61
3.4.2. Vizsgált résznyelv kiterjesztése	61
3.4.3. Felhasználó azonosítása.....	62
3.4.4. Gráf elemzés összesített verziója.....	62
3.4.5. Felület optimalizálás	62
4. Összefoglalás	64
5. Irodalomjegyzék	65

1. Bevezetés

Szakdolgozatom célja egy olyan P4 [1] programozási nyelvet elemző program elkészítése, ahol a felhasználói élmény ugyanannyira fontos, mint a háttérben lefutó számítások és elemzések precizitása.

Programom háttérfolyamatait kezelő részét C# programozási nyelven írtam, amivel az egyetemen találkoztam és hamar megszerettem, ezért mindenféleképpen el akartam mélyíteni benne a tudásomat.

A programot megjelenítő rész sokáig kérdéses volt számomra, hogy mi legyen, mivel szakdolgozatomban nem ezen van a fő hangsúly. Az egyetemen tanult felhasználó felületek, mint például a Xaml [2] vagy a Razor Pages [3] nem tartoznak kedvenceim közé, ezért valami új dologgal szerettem volna megismerkedni, így esett a választásom az Angularra [4], amivel a felület és a háttérfolyamatok teljesen elkülönülnek egymástól. A fő okai a döntésemnek, hogy az Angular keretrendszeren belül is lehet objektumorientált szemlélettel programozni, valamint, hogy az interaktív felhasználói felületek elkészítésére az egyik legjobb választás napjainkban.

1.1. Alapfogalmak

Szakdolgozatom témájából eredően a probléma megoldásához P4 nyelv megismerése, valamint a vezérlés- és adatfolyamgráffal való megismerkedés szükséges volt. Emiatt elengedhetetlennek tartom, hogy ezekre a fogalmakra külön is kitérjek.

1.1.1. P4

A P4 egy olyan új programozási nyelv melynek segítségével a hálózaton közlekedő csomagok feldolgozása és tovább küldése a feladata. Említenék pár szót a program azon részeiről, amelyet szakdolgozatomban elemeztem, használtam:

- Fejléc – Csomagformátum leírására, valamint a csomagban található mezők leírására használják. Neve tetszőleges lehet, de programon belül egyedinek kell lennie. Szakdolgozatom fő témája, az ilyen szerkezetekkel kapcsolatos változtatások, felhasználások és lehetséges hibák kiszűrése.

- Parser – Kicsomagolást végzi, ahol meghatározza, hogy mely fejlécek lesznek kezdetben inicializált állapotban. A program ezen részét nem vizsgálom. Funkcionalitását kiváltja, hogy a felhasználó megadhatja, hogy mely fejlécek inicializáltak kezdetben.
- Ingress kontroll – A program fő blokkja, amit szakdolgozatomban elemzek. Részei:
 - Kontrollfüggvény törzse – Az Ingress kontroll lefutását végző függvény, ez alapján megy végbe a folyamat.
 - Table – Olyan szerkezetek, melyek tartalmaznak kulcsokat és egy akciókból álló listát. Egy P4 program lefutásakor egy külső vezérlőtől kapott tábla vizsgálata után választja ki az akciót. Szakdolgozatomban én ezt elágazásként kezeltem, ahol minden akció ugyanakkora eséllyel fut le.
 - Akció – Lényegében függvények, ahol egy kódrészlet lefut. Változtathatja a fejlécek inicializáltságát, valamint mezőket deklarálhat vagy adhat nekik új értéket.
- Deparser – A program fő blokkjának lefutása végeztével összecsomagolja a csomagot. A program ezen részét nem vizsgálom. Értékeit a felhasználó adhatja meg. Funkcionalitását kiváltja, hogy a felhasználó megadhatja, hogy mely fejléceket csomagoljuk vissza.

1.1.2. Vezérlésfolyamgráf

A számítástechnikában a vezérlésfolyamgráf egy olyan gráf, amely bemutatja a program lefutását és a kódrészletek kapcsolatát. Ez a fajta gráf nélkülözhetetlen számos fordító optimalizálásához és statikus elemző eszközökhöz. A gráfban szereplő csúcsok mindegyike a programnak egy blokkja, melyek egymást követik a program lefutása során. A legtöbb esetben a gráf reprezentációja tartalmaz kettő fő csúcsot. Az egyik a belépési blokk, ahonnan a folyamat elindul, a másik pedig a kilépési blokk, ahova az összes kódrészlet befut.

1.1.3. Adatfolyamgráf

Az adatfolyamgráf nagyon hasonló az előbb említett vezérlésfolyamgráfhoz, de ebben az ábrázolási módban nem a blokkok közötti kapcsolatot vizsgáljuk, hanem a blokkban történő utasításokat, vagyis értékadások, aritmetikai műveletek,

változókra hatással lévő függvények vannak a középpontban. Ez a gráf is ugyanúgy használatos elemzésekhez és optimalizáláshoz.

2. Felhasználói dokumentáció

2.1. Program célja

A feladatom egy olyan felhasználóbarát és könnyen kezelhető weboldal elkészítése volt, amely P4-es programok gráf alapú elemzését végzi. A programkód bevitelét meg lehet tenni szövegbeviteli mező segítségével vagy fájlfeltöltés útján. A gráfokkal interaktív események hajthatóak végre. Többek között gráf bejárás szimulálása, csúcshoz és csúcsból vezető út kirajzolása és gráfok egymás közötti kapcsolatának megjelenítése. Ezek után meg lehet adni a kezdeti- és végértékeket, majd több megadása után csomagot összeállítani és végezetül az elemzett gráfok és eredmények megtekintését.

A weboldal főleg a P4 programnyelvvvel foglalkozó közösség számára szól, de bárki számára használható az oldalon megtalálható fájlok segítségével, valamint a dokumentáció vagy a webhelyen található Sűgó oldal átolvasásával.

2.2. Használat

2.2.1. Feltételek

Egy weboldal lévén a felhasználásnak nincsenek rendszert érintő követelményei. A weboldal használható mobileszközről, tabletről, valamint számítógépről is, ezek közül ajánlott a számítógép használata nagyobb képernyő és átláthatóság érdekében. Valamint az alapvető webböngészés képességével is rendelkeznie kell a felhasználónak.

Támogatott böngészők:

- Google Chrome
- Mozilla Firefox
- Opera
- Microsoft Edge

2.2.2. Futtatás

Az alábbi címen elérhető a weboldal: <https://p4analyst.azurewebsites.net/>, így telepítésre nincsen szükség.

2.2.3. Vizsgálható kód

A weboldal P4 programok elemzését végzi, így az elfogadott fájlok kiterjesztése a .p4, de a .txt kiterjesztésű fájlok feltöltését is megengedi az oldal.

A megadott fájlnak vagy szövegnek helyes P4 kódnak kell lennie, mivel ennek ellenőrzését nem teszi meg a rendszer. Ez azt jelenti, hogy a feltöltött fájl helyességét ajánlott a P4 fordítóprogramjával ellenőrizni. Amennyiben nem rendelkezik ezzel a fordítóval és telepíteni nem akarja, akkor a fájlok oldalon található olyan helyes P4 kódokat, mellyel megtekinthető a weboldal működése.

Azonban a program nem rendelkezik a P4 kódok teljes funkcionalitásainak ismeretével, ezért megkötésekkel rendelkező P4 programok ellenőrzésére és vizsgálatára van mód a weboldalon. A hibátlan működés miatt ezeket be kell tartani. Fontos kiemelni, hogy a program fő funkciója a fejlécek vizsgálata, így az olyan műveleteket, melyek nem ilyen egységgel kapcsolatosak nem tudom kezelni.

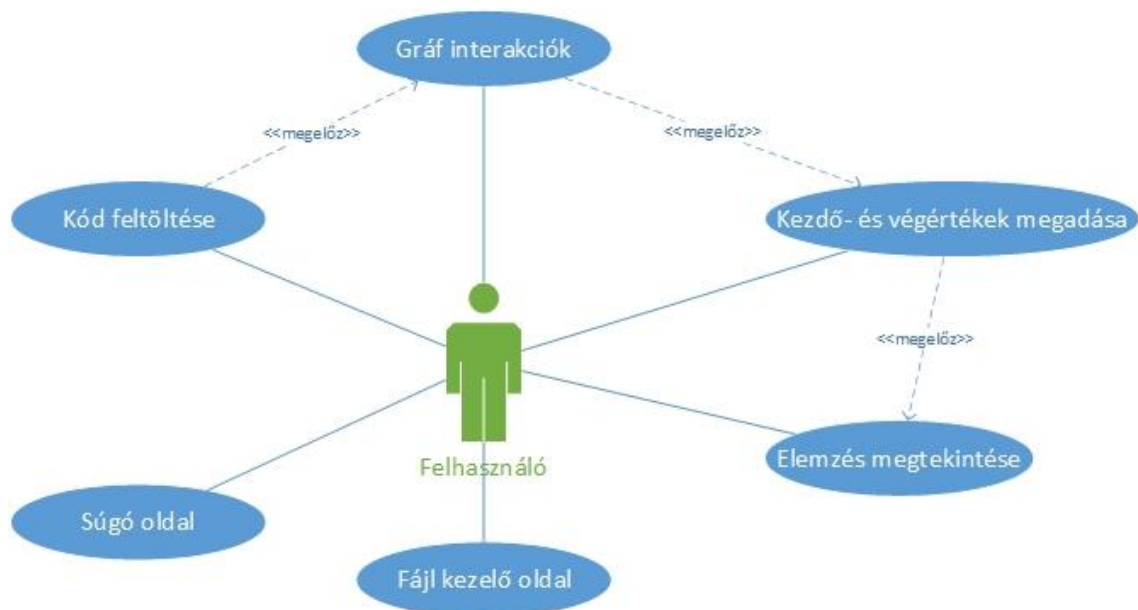
Kivételek:

- Akción belüli elágazás nem megengedett.
- Else if elágazás nem megengedett.
- Szakdolgozatomban a P4₁₆-os verzióit elemzem. Más verzió használata nem támogatott.

2.3. Rendszer ismertetése

Az *Általános felhasználói eset diagram* ábrán láthatóak a weboldal fő funkciói. Ez csak egy összefoglaló ábra, melynek funkcióit bővebben kifejtem később, amikor is a weboldal összes oldalát és azokban rejlő lehetőségeket részletesen leírom.

Az ábrán jól látható, hogy a weboldal egy része lineáris lefutással bír, vagyis egymásra épülő, ki nem kerülhető folyamatok követik egymást.



1. ábra – Általános felhasználói eset diagram

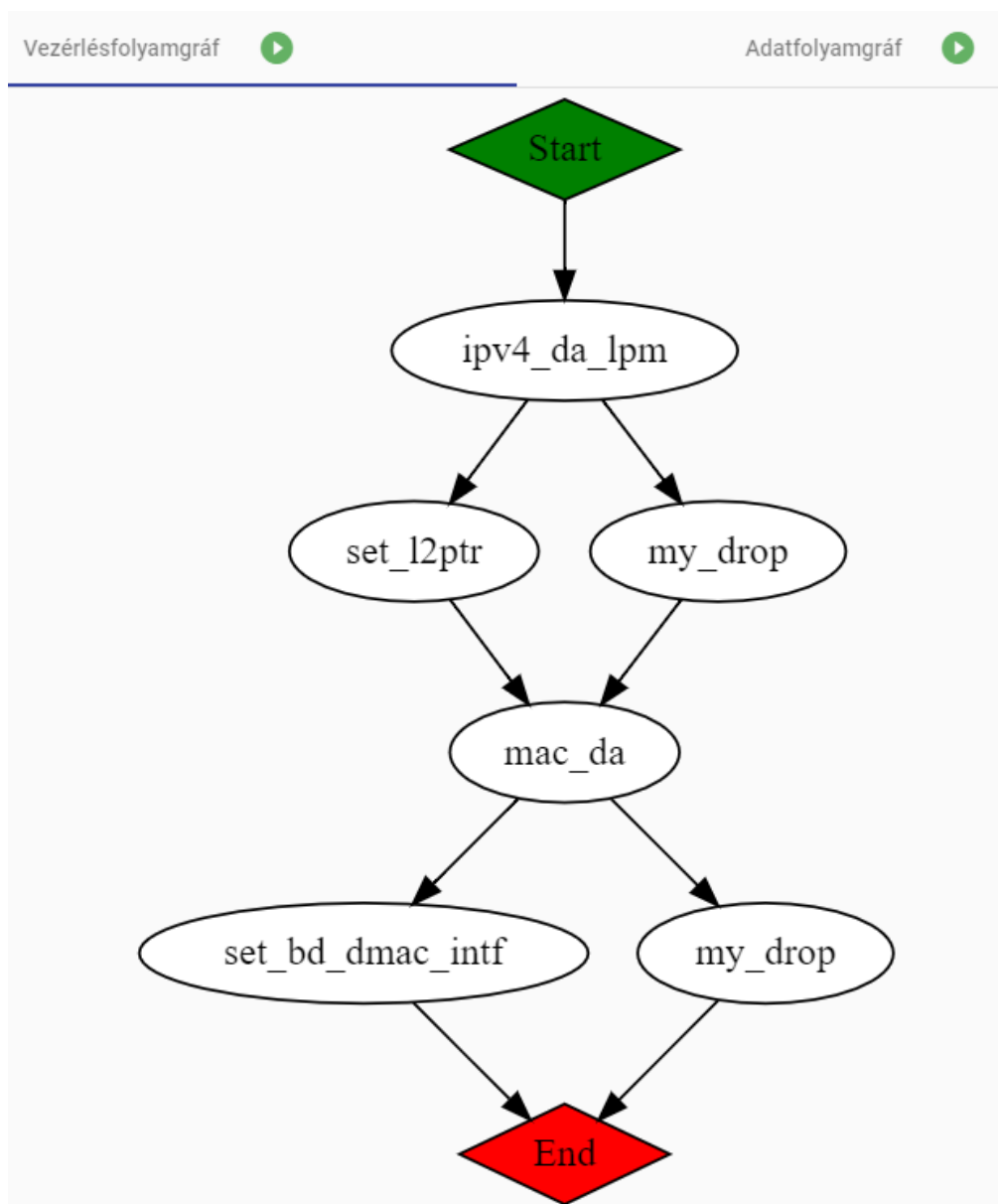
Funkciók rövid ismertetése:

- Kód feltöltés – A felhasználónak itt lehetősége van egy P4 kód megadására fájlból vagy szöveges bevitellel.
- Gráf interakciók – A felhasználónak itt lehetősége van a kódból generált gráfokkal interakciókat végre hajtani. Ezek közé sorolható a gráf bejárás szimulálása, az adott csúcsba vezető út és abból kivezető utak színezése, valamint a vezérlésfolyamgráf és az adatfolyamgráf közötti kapcsolatokat megismerése.
- Kezdő- és végértékek megadása - A felhasználónak itt lehetősége van a kódban lévő fejlécekről eldönteni, hogy kezdetben mely fejléceit szeretné inicializáltra állítani, valamint az elemzett rész után mely fejléceket szeretné felhasználni. Ez a funkció a *Parser*-t és *Deparser*-t váltja ki, mivel azokat nem dolgozom fel.
- Elemzés megtekintése - A felhasználónak itt lehetősége van az elemzések megtekintésére, ahol is megjelenik az összes megadott kezdő- és végértékhez generált gráf, valamint diagramok, melyeket a rendszer számolt ki.
- Fájl kezelő oldal – A felhasználónak itt lehetősége van fájlokat keresni, azok tartalmát vágólapra helyezni, vagy a fájlal azonnal megkezdeni az elemzési folyamatot, ami a gráf interakciókkal kezdődik.

- Súlyó oldal - A felhasználónak itt lehetősége van megismerkedni a weboldal céljával, valamint alapvető használatával. Elolvasásával könnyen elsajátítható a tudás, amely az oldal használsához szükséges.

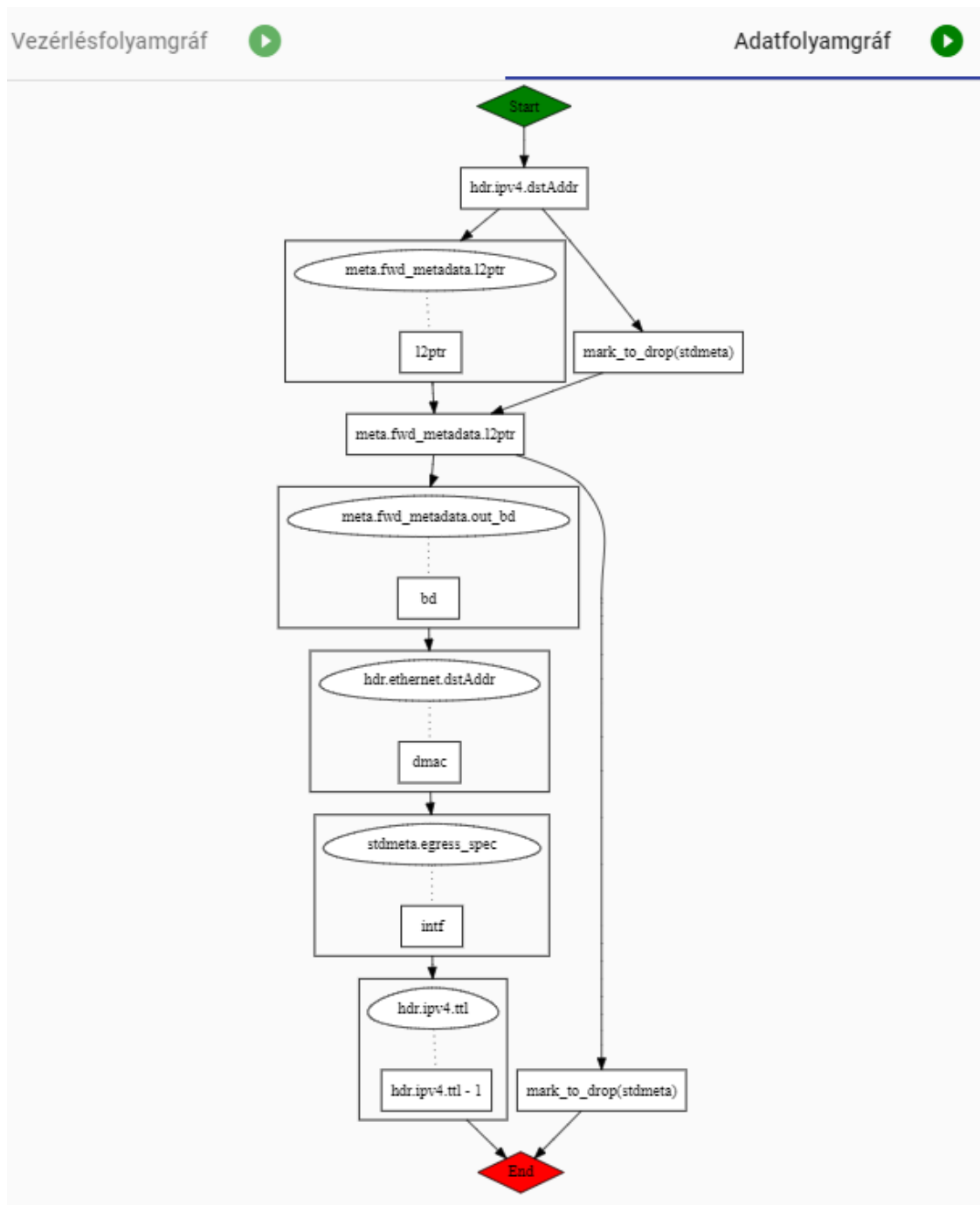
2.3.1. Gráfok ismertetése

Szakdolgozatomban sok munkát fektettem a gráfok megismerésére és a megfelelő ábrázolásuk kidolgozásukba. Pár mondatban összefoglalnám, hogy a gráfok, valamint a csúcsok között milyen kapcsolatok állnak fel és ezeket milyen módon ábrázolom. Ahogy a *Vezérlésfolyamgráf* és az *Adatfolyamgráf* ábráján lentebb látszik, hogy mindkét gráfban szerepel egy kezdő, valamint egy vég csúcs.



2. ábra - Vezérlésfolyamgráf

A vezérlésfolyamgráfon a többi csúcs között nincsen különbség. Minden csúcs egyenlő értékkel bír és a program lefutásával megegyező módon követik egymást. Az egeret egy csúcs fölé helyezve megjelenik a típusa, melyek lehetnek *if*, *table*, *action*. Az élek közül kivételt jelentenek az *if* csúcsból kiinduló élek, itt az igaz ág zöld színű éllel van jelölve, míg a hamis ág piros színű.



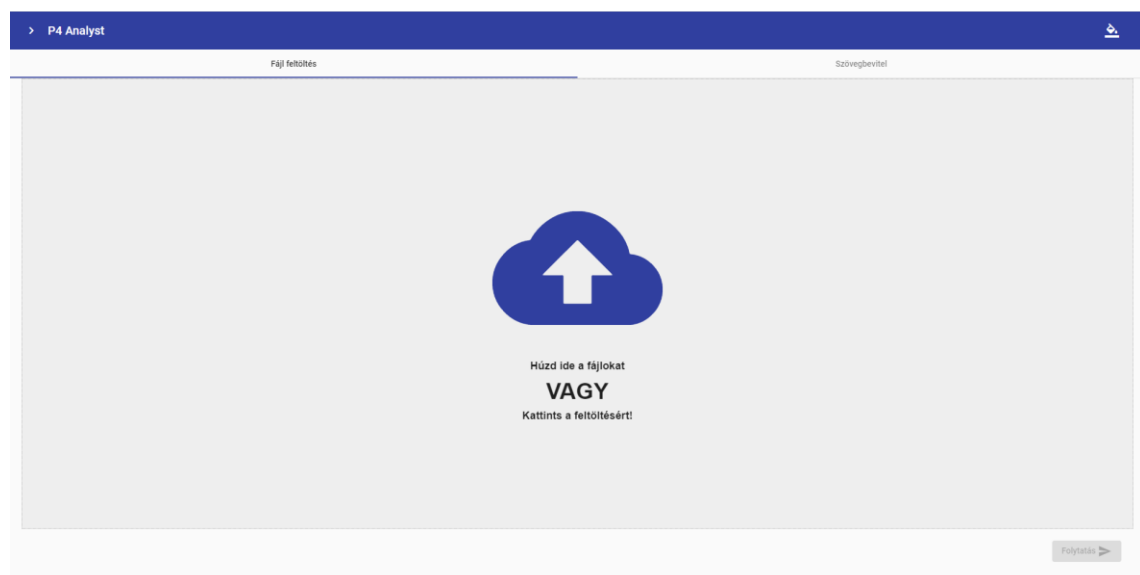
3. ábra - Adatfolyamgráf

Az adatfolyamgráfon láthatunk szögletes, valamint tojás alakú csúcsokat ezek jelentése pedig az, hogy a szögletes csúcsban lévő adat olvasásra kerül, míg a tojás alakú csúcsban lévő adat írásra kerül. Az itt megjelenő csúcsok mindegyike egy adott vezérlésfolyamgráf blokkja vagy változó felhasználása. Itt is *if* csúcsból kiinduló élek egyike zöld, másik piros színnel vannak jelölve, ha az *if* feltétele több csúcsra bontható, akkor ezek mindegyike szögletes alakú csúcsba kerül és szaggatott vonallal vannak összekötve, ezzel jelezve összetartozásukat. Az *Adatfolyamgráf* ábrán látható, hogy vannak szögletes és tojás alakú csúcs párok ezek jelölése is szaggatott vonallal történik meg, ezek értelmezése, hogy az itt olvasott csúcsnak jelölt kódrészlet értéke kerül bele az írt csúcsban lévő változóba.

2.3.2. Kezdő oldal

Ezen az oldalon adhatja meg elemzendő kódját, itt egy tab-os elrendezésben választhatja ki, hogy milyen módon adja át kódját. Az opciók kizárják egymást, szóval, ha feltöltött fájlt, addig nem léphet át a szövegbeviteli részre, amíg azt a fájlt ki nem törölte és fordítva is igaz, tehát, amennyiben a beviteli mező nem üres úgy nem léphet vissza a fájl feltöltő részlegre. A következő oldalra lépés, akkor érhető el, ha az egyik eset által adott meg szöveget. A szövegbeviteli mező egy egyszerű beviteli mező, ahova szöveget írhat be vagy illeszthet be a vágólapról.

Fájl feltöltés



4. ábra - Fájl feltöltő

A felhasználó a *Fájl feltöltő* ábrán látható felületen töltheti fel kódját, ebben az esetben két különböző úton adhatja meg a kiválasztandó fájlt. Az első eset, hogy bal

klikkelés után a fájlkezelő ablakból kiválasztja a feltöltendő fájlt. A második eset, hogy úgynevezett *drag and drop* módszerrel, vagyis a fájl odahúzásával és elengedésével adja meg a kiválasztott fájlt. Minden esetben ellenőrizzük, hogy a fájl megfelelő, .p4 vagy .txt kiterjesztésű-e, más kiterjesztéssel rendelkező fájl feltöltésére nincs lehetőség, ezt figyelmeztető üzenet formájában jelezzük a felhasználónak. A második módszer esetén a feltöltött fájlok mennyiségére is ellenőrzést hajtunk végre, mivel több fájl feltöltése nem megengedett, ekkor is megtagadjuk a fájl(ok) feltöltését és figyelmeztető üzenet formájában ezt jelezzük. Az elfogadott fájl megadása után a képernyőn megjelenik a fájl neve, valamint mellette kettő akció gomb is. Az első esetben a fájl szerkesztését végezheti, ekkor egy felugró ablakban jelenik meg a fájl tartalma, ahol szerkeszthető. Az ablak bezárása négy módon történhet meg:

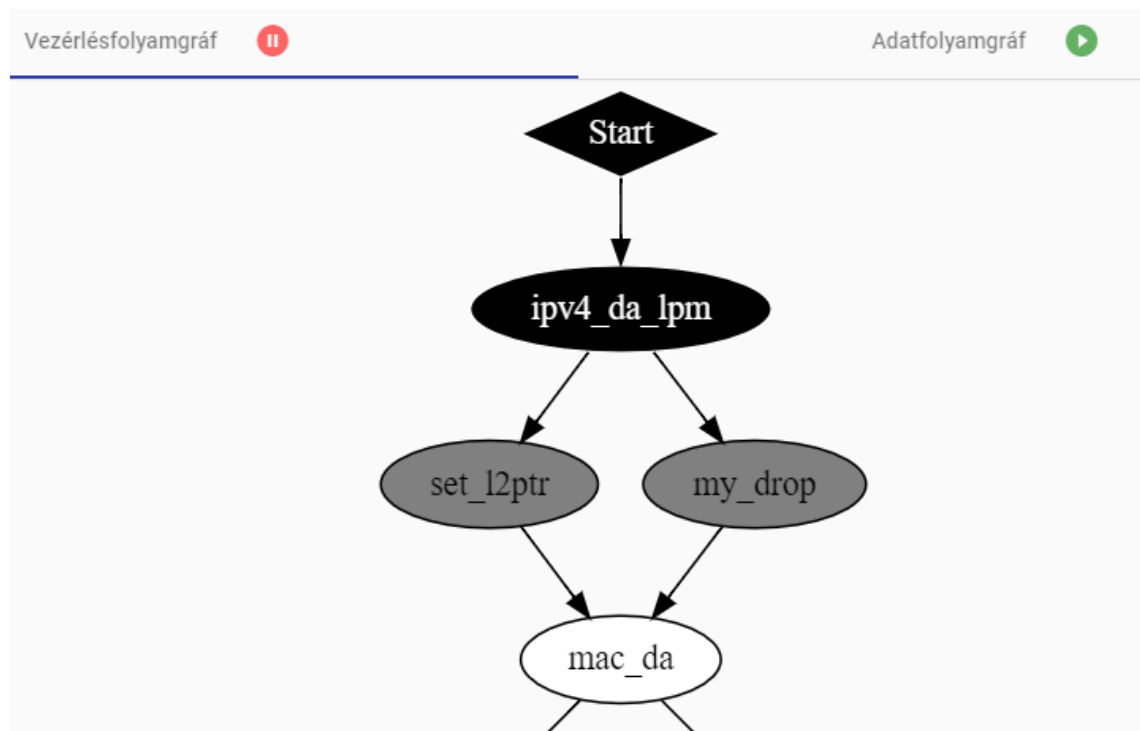
- *Mentés* gombra kattintás – a szerkesztést jóváhagyja, ezt egy felugró üzenettel erősítjük meg
- *Mégsem* gombra kattintás
- Bezáró gombra kattintás
- Az ablak területén kívülre kattintás

A törlés gombra való kattintás esetén a fájl feltöltése semmisnek tekintendő és lehetőség nyílik másik fájlt feltölteni, vagy a szövegbeviteli mezőre átlépni.

2.3.3. Gráf megjelenítő oldal

Az oldal működésének előfeltétele, hogy a kód feltöltése már megtörtént, ezért ha az nem történt volna meg, akkor hibaüzenettel jelezzük, hogy először fájlt fel kell tölteni. Amennyiben már ez megtörtént, akkor az oldalon a tab-os elrendezésben alapértelmezetten kettő oldal található meg, melyek az oldalon állandóan jelen vannak. Az első a vezérlésfolyamgráfot megjelenítő tab, a második pedig az adatfolyamgráfot reprezentáló oldal. A gráfok betöltésekor egy információs üzenettel jelezzük, hogy a gráf betöltését várja meg, mivel csak így tudjuk garantálni a hibátlan és maximális felhasználói élményt. A gráf betöltődése addig tart, amíg a képernyőn megjelenő forgásban lévő ikon a képernyőn marad, utána a gráf kirajzolása következik, ahol a gráf csúcsai és azokat összekötő élek beúsznak az oldalra. Amikor a gráf kirajzolása is megtörtént onnantól érhetőek el az oldal interakciói.

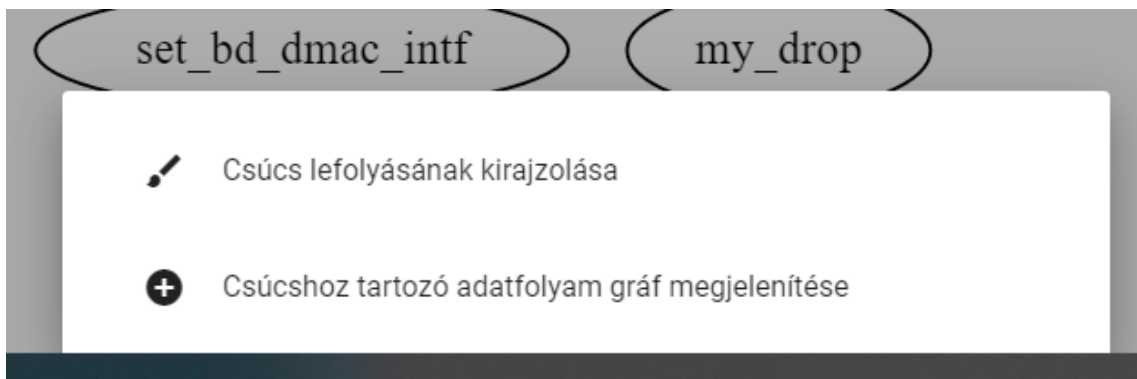
Gráf bejárás szimulálása



5. ábra - Gráf bejárás

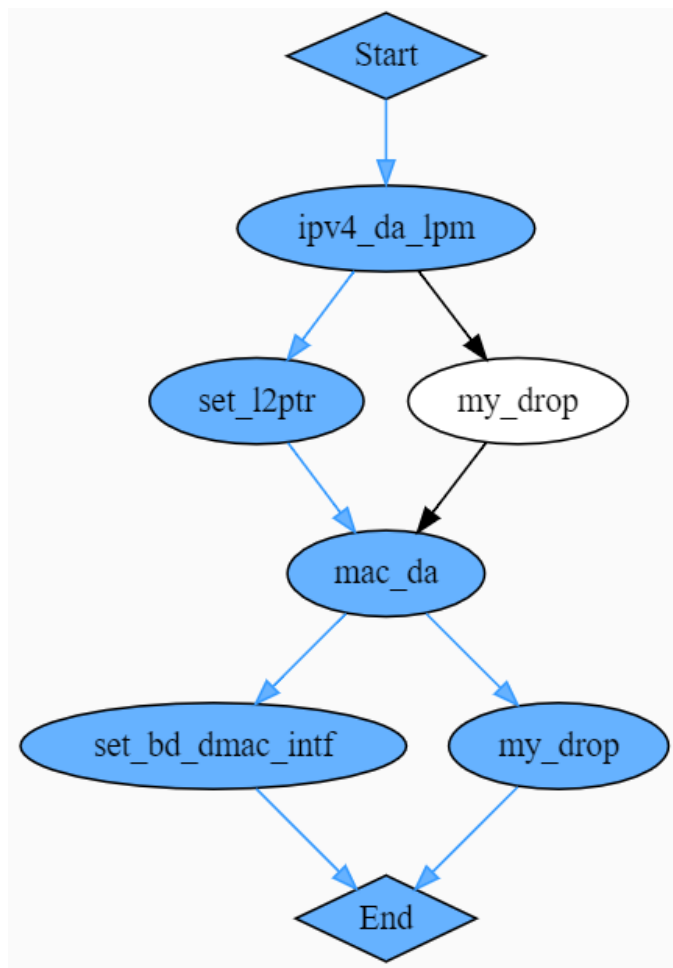
A két fő gráf bejárásának szimulálásra van lehetőség. A funkció elindításához a tabban, név mellett látható zöld lejátszási ikon megnyomását kell megtenni. Lenyomásakor a lejátszási ikonból szüneteltetési ikon lesz, valamint a zöld színből piros színbe vált át a gomb, amit a *Gráf bejárás* ábrán láthatunk is. A bejárás prezentálásához a szélességi bejárást választottam, amit egyetemi éveim alatt sajátítottam el. A bejárás lényege, hogy szintenként halad és csak akkor ugrik a következő szintre, ha az adott szinten már mindent csúcstól elért. A bejárás szüneteltethető a szüneteltetési ikon lenyomásával vagy a tab-ról való elkattintással. A folyamat a megszakított állapottól folytatható. A funkció csak, akkor aktív, ha a felhasználó azon a gráfon van, amire alkalmazni szeretné.

Csúcs interakciók



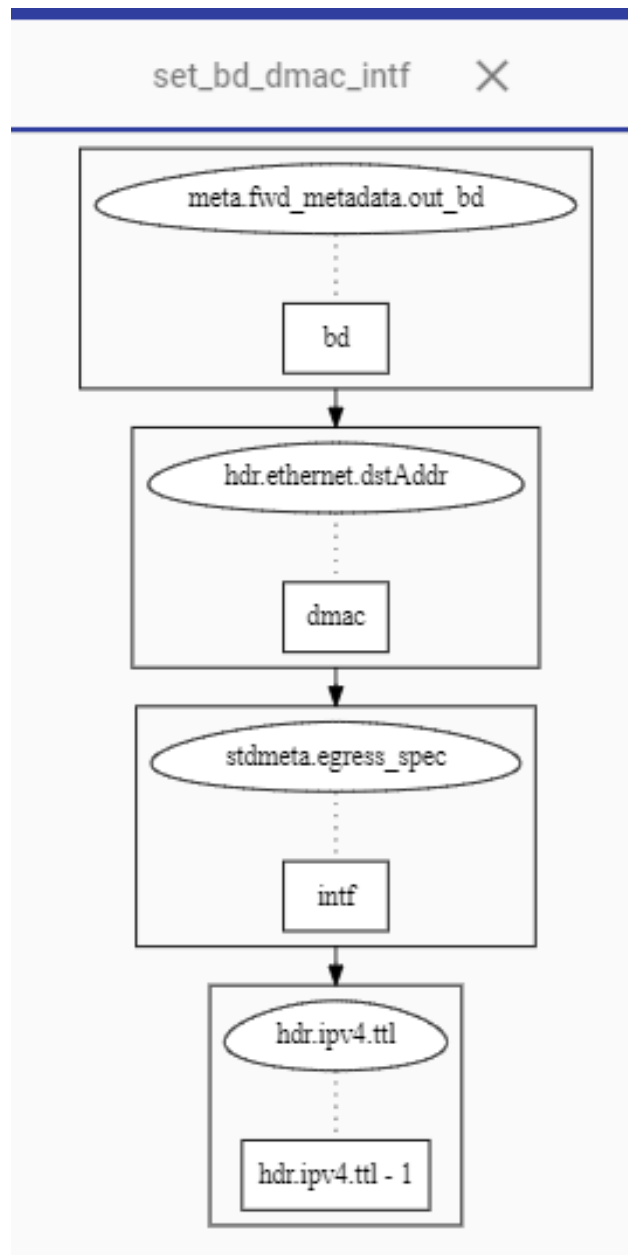
6. ábra - Interakció kiválasztása

Csúcs interakciók is csak a két fő gráfon hajthatók végre. Egy csúcsra kattintás esetén az *Interakció kiválasztása* ábrán látható felugró felületen van lehetőség az opciók közül választani.



7. ábra - Csúcs lefolyásának színezése

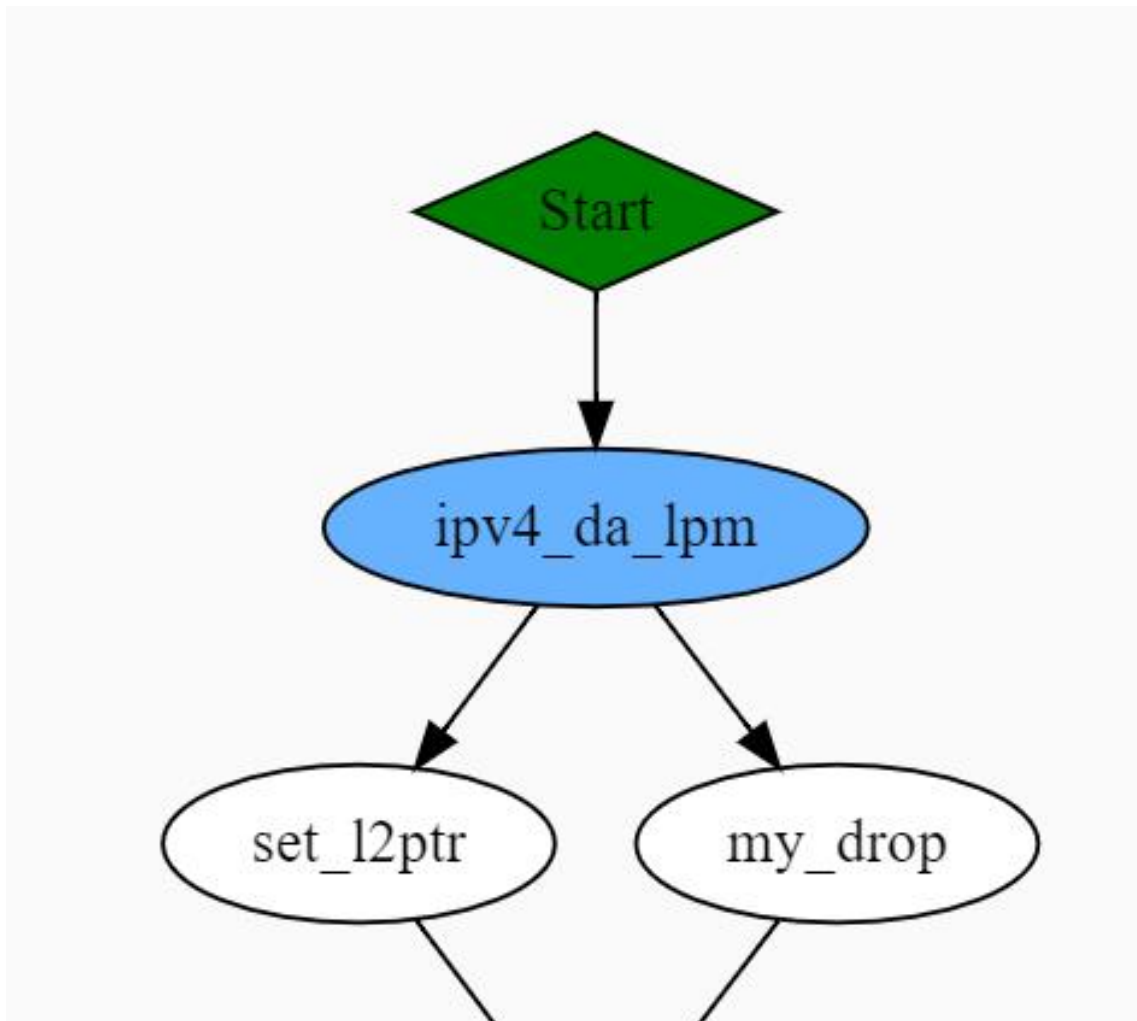
Mindkét gráf esetében, minden csúcsnál van lehetőség a csúcshoz vezető és abból kimenő utak színezésére. Ezt a *Csúcs lefolyásának színezése* ábra segítségével reprezentálom, ahol a *set_12ptr* csúcson került kiválasztásra ez a funkció.



8. ábra - Részgráf megjelenítés

A vezérlésfolyamgráf esetében, ha olyan csúcsra kattint, amihez tartozik adatfolyam részgráf, akkor az opciók között szerepelni fog a *Csúcshoz tartozó adatfolyam gráf megjelenítése* is. Az opció kiválasztása esetén az adatfolyam részgráf egy új tab-on fog megnyílni, amelyre egyből odaugrik az oldal, ha ez a részgráf már meg volt nyitva, akkor az oldal egyszerűen csak arra a tab-ra ugrik. Az így megnyílt gráfon

nem használható egyetlen interakció sem, a tab-ot az *X* gomb segítségével be lehet zárni. A *Részgráf megjelenítés* ábrán látható, hogy itt egy részgráf jelenik meg, valamint a bezárás funkcióval rendelkező ikon is.

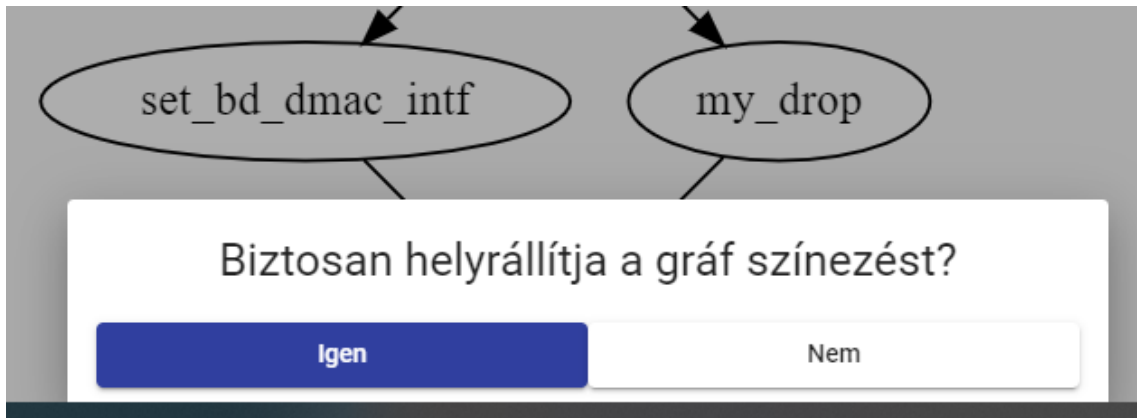


9. ábra - Csúcs kiválasztás

Az adatfolyamgráf esetében, ha olyan csúcsra kattint, ami egy vezérlésfolyamgráf csúcsból alkotható adatfolyam részgráfnak egy csúcsa, akkor az opciók között szerepelni fog a *Csúcs*hoz tartozó vezérlésfolyam gráf csúcs kijelölése is. Az opció kiválasztása esetén az oldal átugrik a vezérlésfolyamgráfhoz és az előbb említett csúcsot kék színnel jelzi. A *Csúcs kiválasztás* ábrán látható, hogy az adatfolyamgráf csúcs

Egyéb funkciók

Az oldalon további két funkció elérhető, az egyik ezek közül a gráf színezés helyreállítása. Ez a funkció is csak a két fő gráfra, vagyis a vezérlésfolyamgráfon és az adatfolyamgráfon alkalmazható.



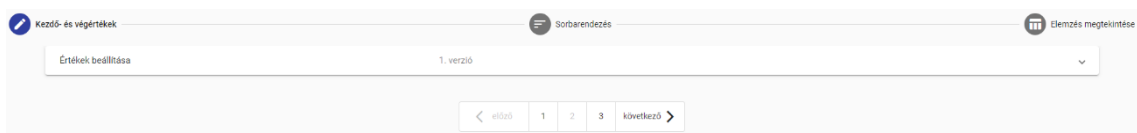
10. ábra - Színezés helyreállítása

Amennyiben a gráf színezésre került valamilyen módon, akkor csúcson kívülre kattintva a *Színezés helyreállítása* ábrán látható felugró ablakrész jelenik meg, ahol eldöntheti, hogy a színezést vissza akarja-e állítani alapértelmezett helyzetébe. A visszaállítást az *Igen* gomb lenyomásával lehet megtenni.

A másik hátralévő funkció a *Tovább lépés a következő oldalra*. Ezt az oldal alján található *Következő* gomb lenyomásával tehetjük meg, ilyenkor az oldal átugrik az elemzéseket végző felületre.

2.3.4. Elemzéseket végző oldal

Az oldal működésének előfeltétele, hogy a kód feltöltése már megtörtént, ezért ha az nem történt volna meg, akkor hibaüzenettel jelezzük, hogy először fájl fel kell tölteni.



11. ábra - Elmező oldal egészében

Az *Elemző oldal egészében* ábrán látható, hogy léptetési rendszerben működik. Az első oldalán adhatóak meg az állapotok melyekkel elemezni kívánja a kódot. A második oldal csak akkor elérhető, ha az első oldalon kettő vagy annál több verziót adott meg, ilyenkor itt egy csomagot állíthat össze. A harmadik oldalon pedig az

elemzett gráfok tekinthetőek meg, valamint azon diagramok melyek a program felhasználtásáról adnak egy átfogóbb képet.

Kezdő- és végértékek megadása



12. ábra - Értékek megadása

Ezen az aloldalon belül adhatja meg lenyíló modulok segítségével a kívánt verziókat. Az *Értékek megadása* ábrán látható, hogy a verziót lehet törölni a *Törlés* gomb lenyomásával, valamint új verziót is hozzá lehet adni az eddigiekhez ezt a *Plusz hozzáadás* gomb lenyomásával teheti meg. A kezdeti értékeket egy lenyíló többes kiválasztó mező segítségével adhatja meg, melynek bezárása után a bepipálásra kerülő értékek fognak megjelenni. A végértékeket ugyanilyen módszerrel lehet megadni, ennek a helye az ábrán nem látszódik.

Tetszőleges mennyiségű verziót lehet megadni, de ugyanolyan értékekkel rendelkezőt nem érdemes ugyanis a *Sorba rendezés* oldalon adható meg csomagszerkezet. Verzióból egynek kötelezően lennie kell, ezt nem tudja kitörölni, itt a *Törlés* gomb nem jelenik meg.

Sorba rendezés

<u>Megadott verziók</u>	<u>Sorba állított verziók</u>
ethernet	ethernet
ipv4 - ethernet; ipv4	ipv4 - ethernet; ipv4
	ethernet

< előző 1 2 3 következő >

13. ábra - Sorba rendezés

Amennyiben több verzió kerül megadásra, akkor elérhetővé válik ez az oldal. Ilyenkor a *Sorba rendezés* ábrán is látható módon jelenik meg a felület. Bal oldalon a *Kezdő- és végértékek megadása* aloldalon megadott verziók szerepelnek, ahonnan a már említett *drag and drop* módszerrel, vagyis egyszerű áthúzással lehet átvinni az elemeket a jobb oldali *Sorba állított verziók* oszlopba. Ilyen módon lehet csomagot összeállítani, vagyis egyes érték variációkat többször megadni, másikat pedig kevesebbszer. Ennek az oldalnak a kitöltése nem kötelező, tehát üresen hagyható a *Sorba állított verziók* oszlop, ilyenkor a *Megadott verziók* oszlopában lévő sorrendet követi a rendszer.

Elemzés megtekintése

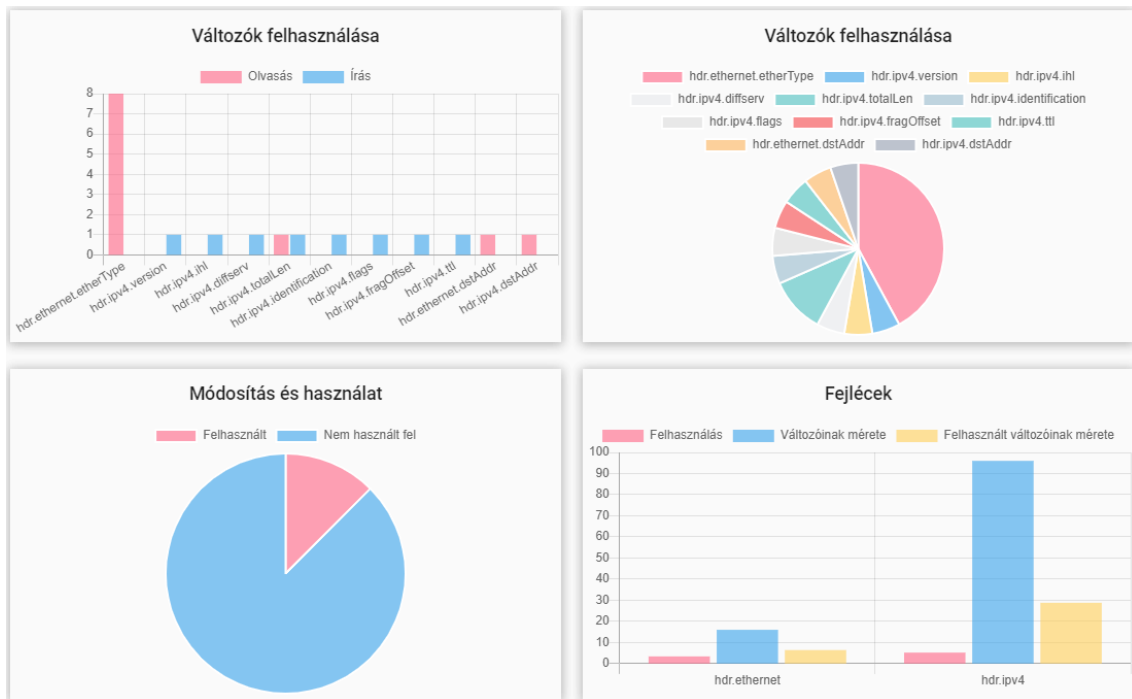
Ezen az oldalon megjelennek az elemzett vezérlésfolyamgráfok, adatfolyamgráfok, valamint olyan diagramok melyek segítségével könnyebben optimalizálható a kód.

Gráfok

Az összes megadott kezdő- és végértékhez meg lehet tekinteni az elemzett gráfokat. Itt a gráfok között lehet lapozni, mindkét gráf fajta külön-külön jelenik meg és külön-külön is lehet köztük lapozni. A gráfoknak színezve vannak, melyek értelmezése az alábbi:

- Zöld – A kód a megadott értékekkel helyes és nem keletkezik hiba.
- Sárga – A kód a megadott értékkel lehet, hibára fut, de ez nem megállapítható biztosan.
- Piros – A kód a megadott értékekkel hibára fut.

Diagramok

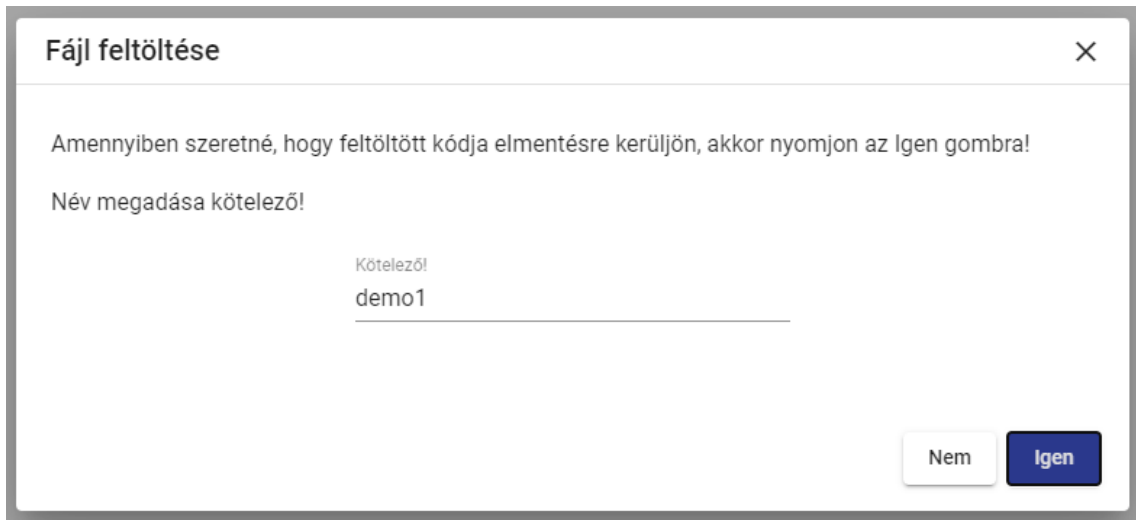


14. ábra - Diagramok

Az oldalon négy diagram található, ahogy az a *Diagramok* ábrán is látható. Melyeknek magyarázata és értelme a következő:

- *Változók felhasználása* (oszlop diagram) – Itt megtekinthető, hogy egy változót hányszor írtak, illetve olvastak átlagban.
- *Változók felhasználása* (kör diagram) – Itt megtekinthető, hogy egy változót hányszor használtak átlagban.
- *Módosítás és használat* – Itt megtekinthető, hogy a változók összesítve hányszor voltak újra felhasználva miután módosítottak rajtuk. Ezzel ellenőrizve, hogy a változó módosítása célszerű volt-e.
- *Fejlécek* – Itt megtekinthető, hogy egy fejléc átlagban hányszor volt használva, továbbá változóinak összesített méretét bit-ben kifejezve, valamint felhasznált változóinak átlagos méretét bit-ben kifejezve. Ebben az esetben a méret nem mindig pontos, mivel vannak olyan változók melyeknek nem ismerem a méretét.

Kód elmentése



Fájl feltöltése

Amennyiben szeretné, hogy feltöltött kódja elmentésre kerüljön, akkor nyomjon az Igen gombra!

Név megadása kötelező!

Kötelező!
demo1

Nem Igen

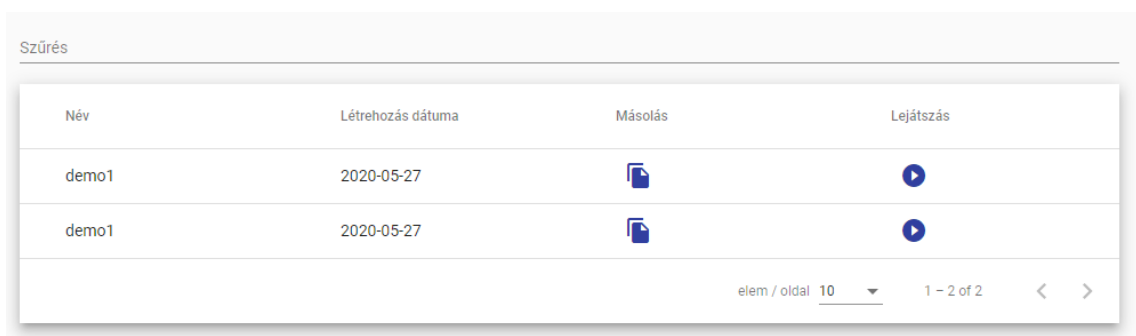
15. ábra - Kód elmentése





Amikor az elemzés oldalra lép és itt is hibátlanul lefutnak a vizsgálatok, akkor megjelenik a *Kód elmentése* ábrán látható felugró ablak. Ilyenkor lehetőség van a kód elmentésére, azzal a feltétellel, hogy megadunk egy nevet neki. Ez az elmentett kód majd a Fájl oldalon lesz megtalálható. Az ablak bezárása négy módon történhet meg:

- *Igen* gombra kattintás – A fájl feltöltése megtörténik, ennek sikerességéről üzenetet kap
- *Nem* gombra kattintás
- Bezáró gombra kattintás

2.3.5. Fájl oldal

A felhasználónak itt lehetősége van korábban már általa vagy más személy által elemzett kódok között böngészni. Ez egy kényelmi funkció, hogy olyan felhasználó is tudja használni az oldalt, aki saját maga nem tudna vagy nem szeretne P4 kódot írni.



Név	Létrehozás dátuma	Másolás	Lejátszás
demo1	2020-05-27		
demo1	2020-05-27		

elem / oldal 10 1 - 2 of 2 < >

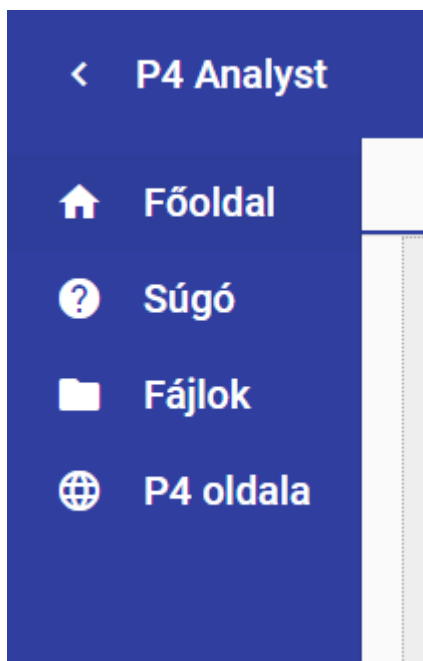
16. ábra - Fájl oldal

Ahogy a *Fájl oldal* ábra is mutatja az oldalon egy táblázat található, ahol a szövegbeviteli mező segítségével lehet szűrni a *Név*-re és a *Létrehozás dátum*-ára, valamint ugyanezek szerint rendezhetőek is az oszlopok. Az átláthatóság miatt egy oldalon csak tíz elem jelenik meg, de ezt változtathatja ötre, vagy húszra is és több oldal esetén lapozhat is köztük.

Az oldalon kettő akció hajtható végre az itt felsorolt fájlokra. Az első a *Másolás*, ami a fájl tartalmát vágólapra helyezi, a második pedig a *Lejátszás*, ilyenkor a fájl tartalmával átlép a Gráf megjelenítő oldalra, vagyis a fájlfeltöltés lépést átugorja.

2.3.6. Kényelmi funkciók

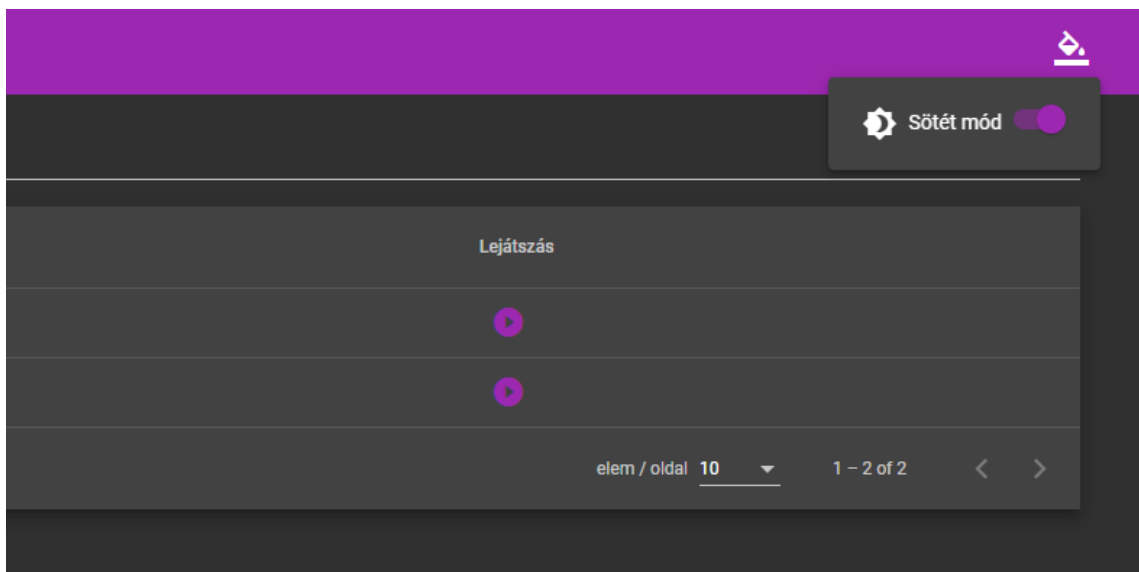
Navigációs rész



17. ábra - Navigációs sáv

A felületen mindig elérhető a bal felső sarokban elhelyezkedő ikon, melynek megnyomásakor a *Navigációs sáv* ábrán látható felület jelenik meg az oldalon. Ennek segítségével könnyen megnyithatjuk az ábrán látható három oldal egyikét.

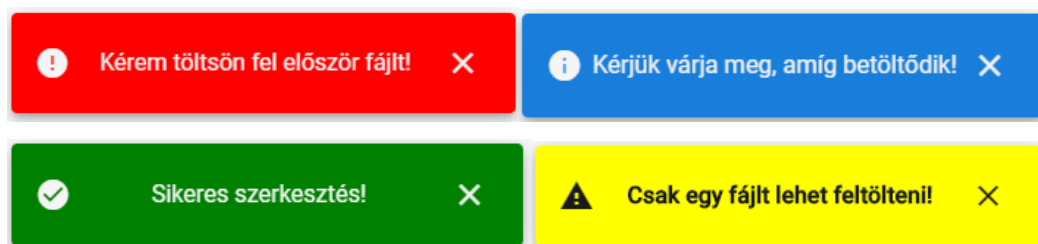
Sötét téma



18. ábra - Sötét téma

A felület jobb felső sarkában található ikonra rányomva a *Sötét téma* ábrán látható menü jelenik meg, ahol ki- és bekapcsolható a sötét mód.

Értesítések



19. ábra - Értesítési panelek

A felületen az *Értesítési panelek* ábrán látható módon jelennek meg üzenetek a felhasználó számára. Ezek értelmezése:

- Zöld – Sikeresen végbehajtott művelet
- Kék – Információval szolgáló üzenet
- Sárga – Figyelmeztető üzenet
- Piros – Hiba üzenet

3. Fejlesztői dokumentáció

3.1. Fejlesztői környezet

Szakdolgozatom fejlesztéséhez két alkalmazást használtam. Az első a Visual Studio 2019 integrált fejlesztői környezet, melyben alkalmazásom háttérfolyamatait készítettem C# objektumorientált programozási nyelven. Használt verziók:

- .NET Core 3.1+
- .NET Standard 2.1+

A másik alkalmazás pedig a Visual Studio Code, ahol az Angular keretrendszert alkalmaztam.

3.1.1. Környezet felépítése

Node.js 12.16.1+ verzió letöltése és telepítése innen [5]

Visual Studio 2019 bármelyik verziójának letöltése és telepítése innen [6]

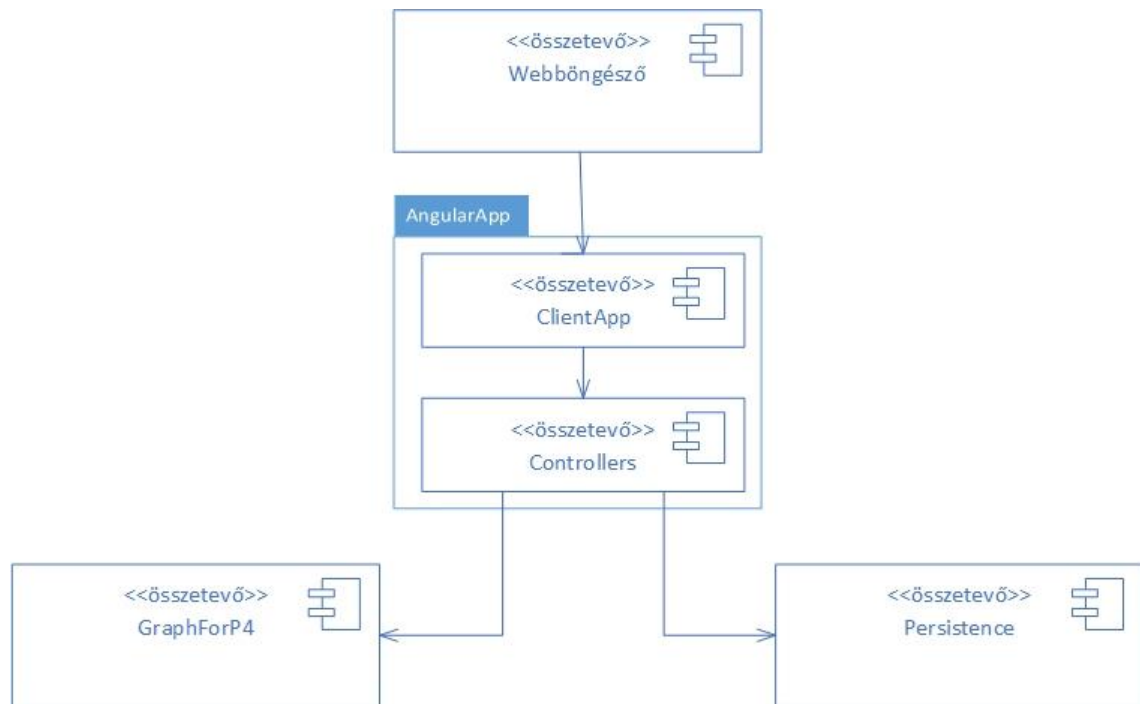
Angular telepítése ez alapján [7]

Ekkor a kód futtatható és fejleszthető. Az Angular cli adta lehetőségeket, mint például a komponens generálást a *Node.js command* prompt alkalmazásban lehet végezni.

Visual Studio Code-hoz ajánlott kiegészítők:

- Angular Language Service – Programozást segítő kiegészítő. Osztályok, változók megadásánál segít opciós lista megjelenítésével.
- TSLint (deprecated) – Kód helyességet figyel és programozási szabvány betartásában segít.
- Material Icon Theme – A mappákat, fájl ikonokat változtatja meg. Jobban láthatóak az Angular egyes részei.

3.2. Program leírása



20. ábra - Program komponens diagramja

A *Program komponens diagramja* ábrán látható az alkalmazás különböző komponenseinek kommunikációja. Egy webes alkalmazás ennek megfelelően a felhasználó webböngésző útján kommunikál a programmal. Rövid ismertetés:

- ClientApp – Angular applikáció, mely http protokoll alapján kommunikál a Controller-ekkel.
- Controllers – Rest Api, ami a kliens oldal kiszolgálásáért felel.
- GraphForP4 – A program fő része itt található, az elemzés, valamint a gráf felépítése is.
- Persistence – Perzisztencia réteg, mely az adatbázis leírását és elérését biztosítja.

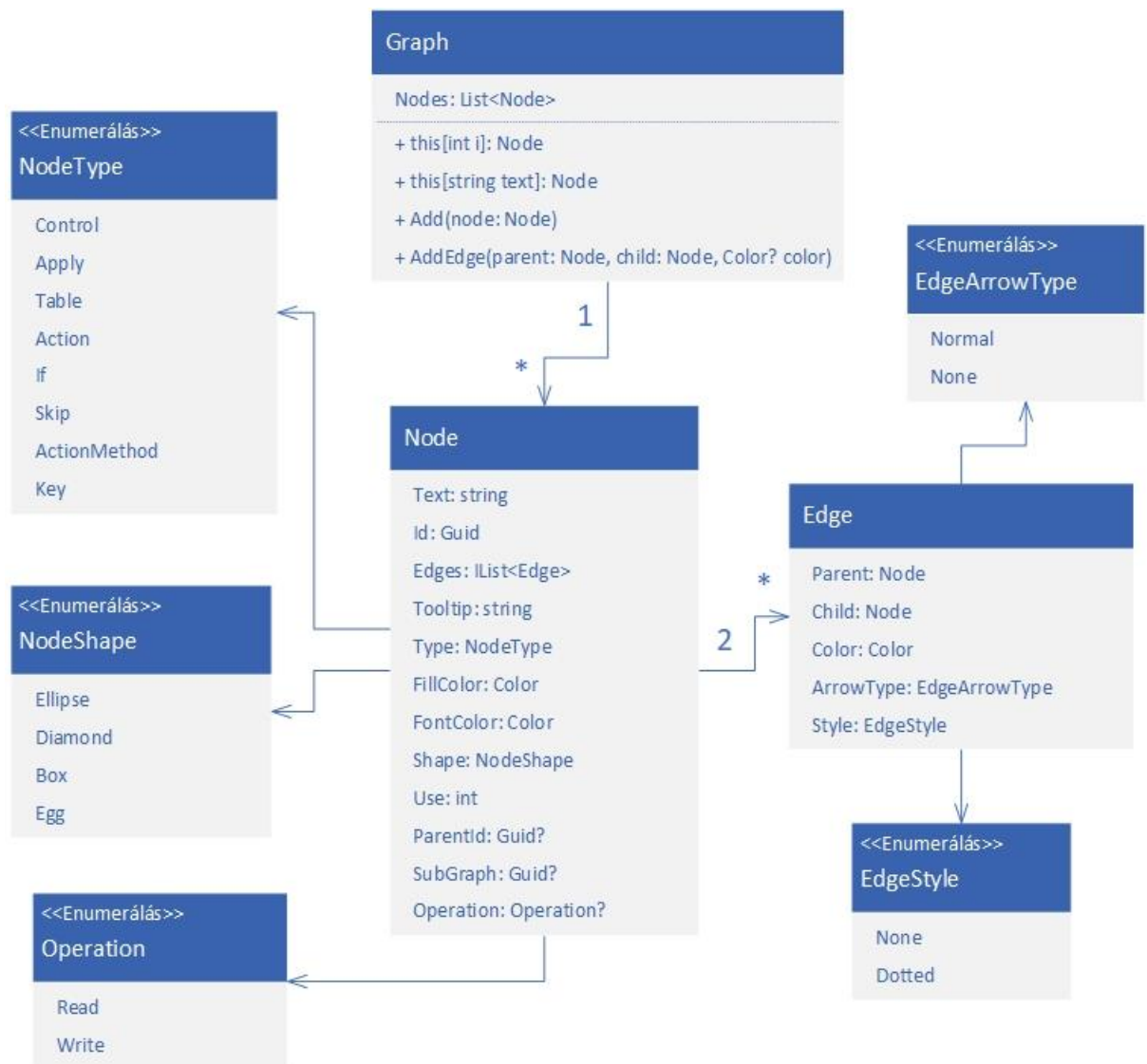
3.2.1. GraphForP4

A GraphForP4 a következő komponenseket tartalmazza:

- Models – Az adat leíró modellek, melyeket a számításokhoz és elemzésekhez használnak itt találhatóak.
- ViewModels – Nézetnek átadandó vagy nézettől kapott modellek szerepelnek itt. Az Angularban is ugyanezek a modellek megtalálhatóak.
- Enums – Használt enumerációk gyűjtő helye.

- Services – Az üzleti logika ezeknek az itt található osztályoknak a segítségével lett megalkotva.
- Helpers – Olyan metódusok gyűjtő helye, melyek több komponensben előfordulnak, így kiszervezésük szükséges volt.
- Extensions – Használt kiegészítők kerültek ide.

Gráf modell



21. ábra - Gráf felépítése

A *Gráf felépítése* ábrán látszódik, hogy az adott osztályok milyen módon kapcsolódnak egymáshoz. Jól látható, hogy minden *Graph*-nak tetszőlegesen számú csúcsa lehet, de egy csúcs csak egy gráfhoz tartozhat. Ugyanezt a megállapítást lehet megtenni a *Node* és az *Edge* osztály között, de ott az élek osztálynak kettő csúcsa lehet.

Graph

Gráf leíró objektum, ami a csúcsok tárolására szolgál, hogy azok könnyen elérhetőek legyenek. Osztály tagjai:

- Nodes – Csúcsokat tartalmazó lista

Osztály függvényei:

- this[int i] – Lehet használni lekérdezésre, vagy érték beállítására. Mindkét esetben a Nodes lista i-edik elemén hajtódik végre a művelet.
- this[string text] – Csak lekérdezésre szolgál. A Nodes lista első olyan elemét adja vissza, melynek a Text mezője megegyezik a bemeneti paraméterrel.
- Add – A bemeneti paramétert hozzáfűzi a Nodes listához.
- AddEdge – A parent Edges listájához hozzáad egy új élt, ami a child csúcsra mutat. Az él színe a bemeneti paraméter color lesz, amennyiben az nem üres, mert akkor alapértelmezetten fekete színű lesz.

Node

Csúcs leíró objektum, ami a csúcs tulajdonságainak tárolására, valamint a belőle kimenő irányított élek tárolására. Osztály tagjai:

- Text – Csúcson megjelenő szöveget tartalmazza.
- Id – A csúcs egyedi azonosítója, ami a későbbi megjelenítés miatt volt fontos, de a vezérlésfolyamgráf és adatfolyamgráf csúcsok közötti kapcsolatok leírásakor is nagyon fontos szerepet kapott.
- Edges – Csúcsból kimenő éleket tartalmazó lista.
- Tooltip – Megjelenés miatt bevezetett adat, ami azt a karaktersorozatot tárolja, ami akkor jelenik meg, ha a felhasználó az egerét a csúcs fölé viszi.
- Type – A csúcs típusát jelzi. Ez által megkülönböztetem, hogy egy csúcs a kód milyen részéhez tartozik, például table, vagy action, de ezt később részletezem.
- FillColor – Megjelenítés miatt fontos, hogy a csúcs milyen színnel lesz kitöltve.
- FontColor - Megjelenítés miatt fontos, hogy a csúcsszövege milyen színnel jelenik meg.
- Shape – Megjelenítés miatt fontos, hogy a csúcsok alakzata megkülönböztethető legyen szerepük alapján.

- Use – Elemzéskor használt változó, ami tárolja, hogy egy adott csúcs hányszor volt használva.
- ParentId – Adatfolyamgráf csúcsainál használt tulajdonság, ami a csúcs szülőjének Id mezőjét tárolja, vagyis annak a vezérlésfolyamgráf csúcsnak az egyedi azonosítója, amiből elő lett állítva a csúcs.
- SubGraph – Adatfolyamgráf olyan csúcsainál használt változó, ahol több csúcs összetartozik, ilyen például az értékadás vagy ha egy elágazásnak több feltétele van.
- Operation – Adatfolyamgráf csúcsainál használt enumeráció, ami megmondja, hogy az adott csúcsban lévő adatot, kifejezést írjuk vagy olvassuk.

Edge

Él leíró objektum, ami az él tulajdonságainak tárolására, valamint a csúcsok összekapcsolására szolgál. Osztály tagjai:

- Parent – Szülő csúcs, amiből kiindul az él.
- Child – Gyerek csúcs, amibe vezet az él.
- Color – Megjelenítés miatt fontos, hogy az él milyen színnel jelenik meg.
- ArrowType – Megjelenítés miatt fontos, hogy az élen szereplő nyíl, hogy jelenik meg.
- Style – Megjelenítés miatt fontos, hogy az él, hogy jelenik meg.

NodeType

Enumerációs adat, ahol a csúcs lehetséges típusai vannak felsorolva. Ez alapján lehet osztályozni, hogy egy csúcs a P4 program milyen részéből lett elkészítve.

NodeShape

Enumerációs adat, ahol a csúcs lehetséges alakzatai vannak felsorolva. Megjelenítéskor fontos, hogy a felhasználó könnyebben tudja értelmezni a gráf felépítését.

Operation

Enumerációs adat, ahol a csúcs lehetséges felhasználási módjai vannak felsorolva. Jelenleg csak az írás és olvasás szerepel, de a későbbi esetleges bővítés miatt használók *enum*-ot.

EdgeArrowType

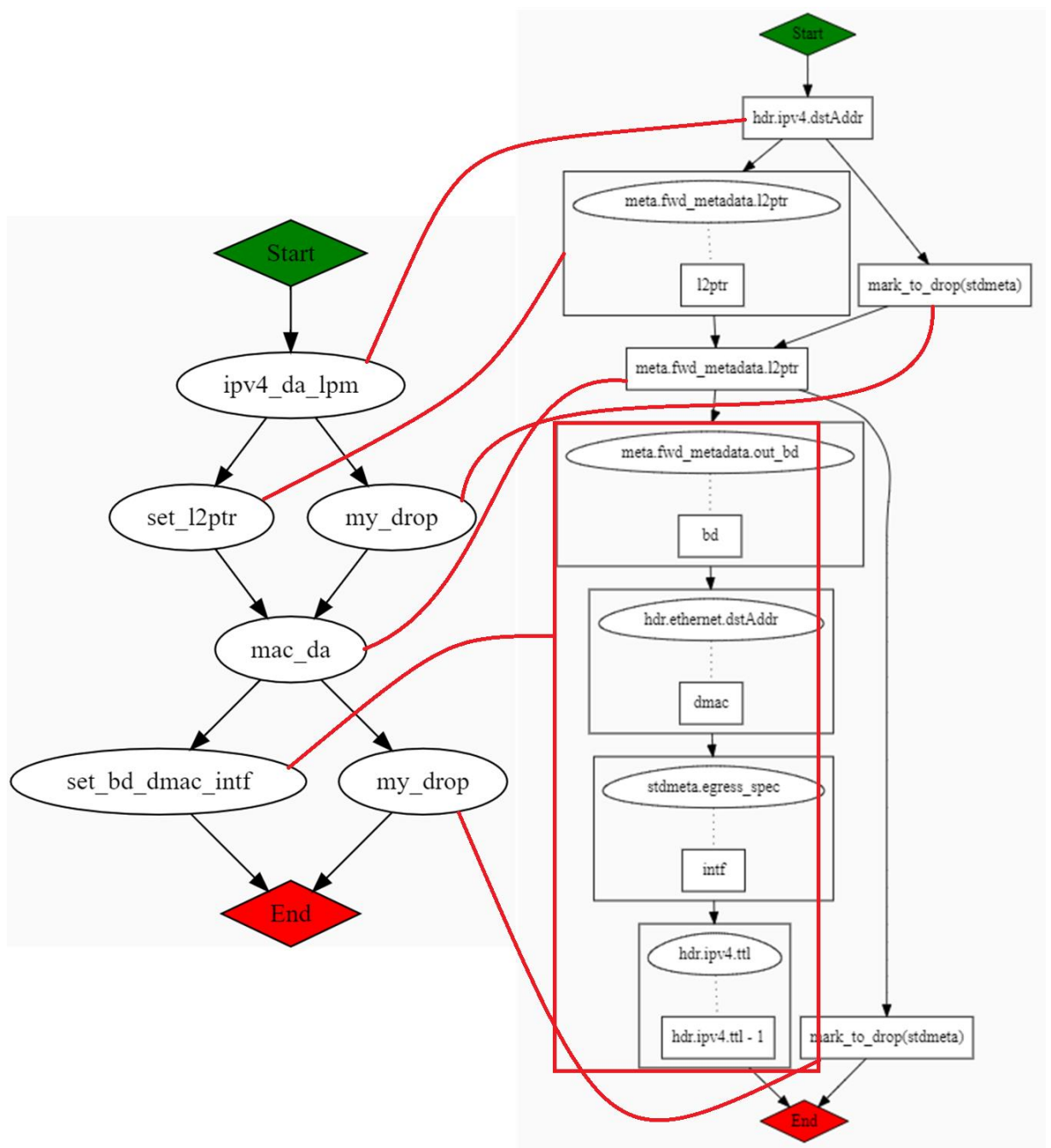
Enumerációs adat, ahol az él lehetséges nyíl típusai vannak felsorolva. Jelenleg csak azt lehet megadni, hogy legyen nyíl vagy ne, de az előbb írt okok miatt lett *enum* típusú.

EdgeStyle

Enumerációs adat, ahol az él lehetséges kinézetei vannak felsorolva. Jelenleg csak azt lehet megadni, hogy az él pontozottan vagy folyamatos vonalként jelenjen meg, de az előbb írt okok miatt lett *enum* típusú.

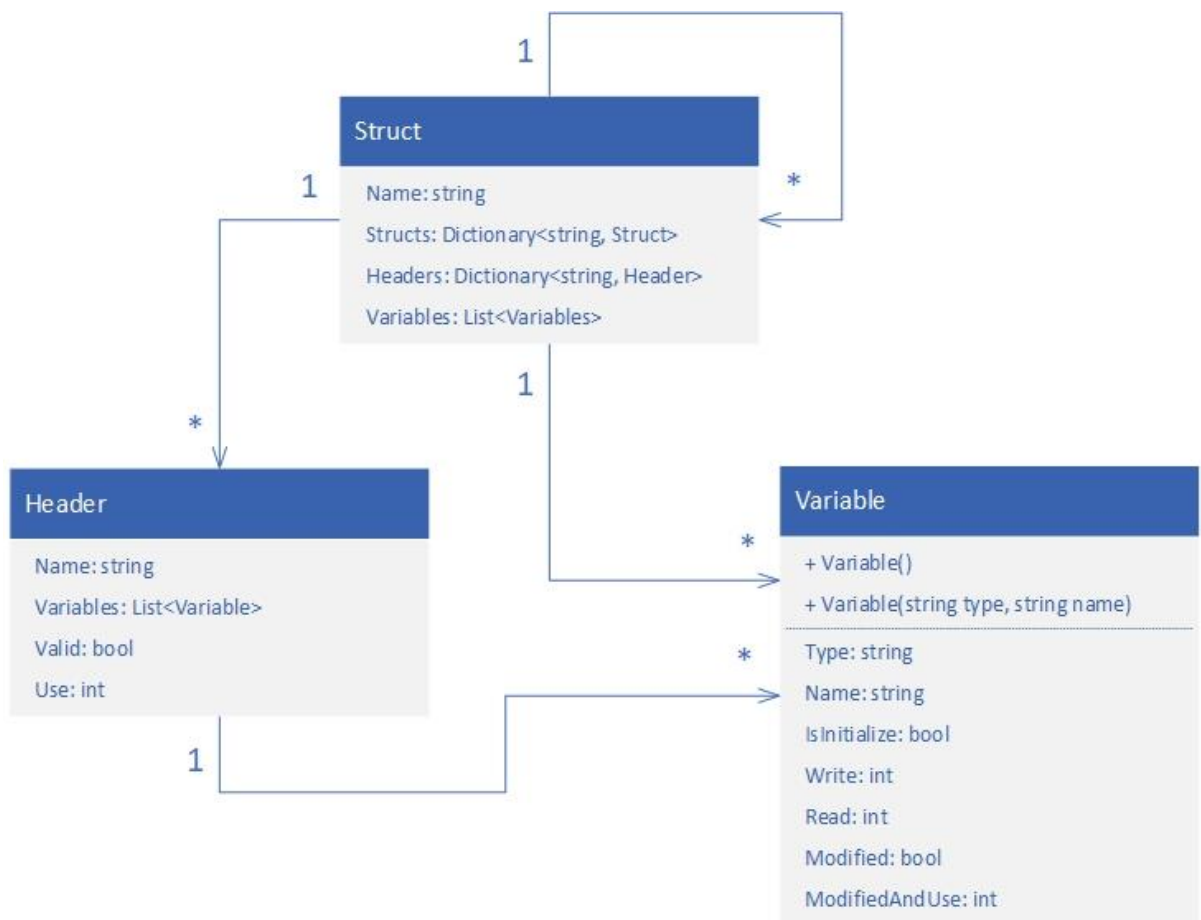
Gráf kapcsolat

A *Gráf kapcsolat* ábrán reprezentálom, hogy a vezérlésfolyamgráf, valamint az adatfolyamgráf milyen kapcsolatban állnak egymással. Ez a kapcsolat nincs referencia szinten tárolva, csak *Node* osztálynál említett *ParentId* tulajdonság jelzi, hogy egy adott adatfolyamgráf csúcsnak melyik vezérlésfolyamgráf csúcs a szülője.



22. ábra - Gráf kapcsolat

Struktúra modell



23. ábra - Struktúra felépítése

A *Struktúra felépítése* ábrán látható, hogy a P4-ben található struktúrákat, fejléceket, valamint mezőket milyen módon reprezentáltam saját programomban. A modell azt a célt szolgálja, hogy a megkapott P4 kódban lévő struktúrákat megfelelően építsem fel. Ennek a modellnek a reprezentációja megtalálható a kliens oldali Angular kódban is, de mivel erre a modellre üzleti logika épül, ezért a programom modelljei közé sorolom.

A *Mintakód* ábrán szerepel egy P4 kód struktúra felépítése, ami a modell felépítésének megértését szolgálja.


```

header ethernet_t {
    bit<48> dstAddr;
    bit<48> srcAddr;
    bit<16> etherType;
}

header ipv4_t {
    bit<4> version;
    bit<4> ihl;
    bit<8> diffserv;
    bit<16> totalLen;
    bit<16> identification;
    bit<3> flags;
    bit<13> fragOffset;
    bit<8> ttl;
    bit<8> protocol;
    bit<16> hdrChecksum;
    bit<32> srcAddr;
    bit<32> dstAddr;
}

struct fwd_metadata_t {
    bit<32> l2ptr;
    bit<24> out_bd;
}

struct metadata_t {
    fwd_metadata_t fwd_metadata;
}

struct headers_t {
    ethernet_t ethernet;
    ipv4_t      ipv4;
}

```

24. ábra – Mintakód

Struct

A P4 program *struct* objektumait reprezentálja. Osztály tagjai:

- Name – Struktúra neve.
- Structs – String-Struct kulcspárokat tartalmazó lista, ahol a String a struct változó neve, a Struct pedig az adott nevű struct-ra mutató referencia.
- Headers – String-Header kulcspárokat tartalmazó lista, ahol a String a header változó neve, a Header pedig az adott nevű header-re mutató referencia.
- Variables – Egy lista, ami a Struct-on belüli változókat tartalmazza. Ezeket csak tároljuk tényleges elemzés nincs rájuk, általában a metadata struct-ok esetében fordul ilyen elő.

Header

A P4 program header objektumait reprezentálja. Osztály tagjai:

- Name – Header neve.
- Variables – Egy lista, ami a header mezőit tárolja.
- Valid – A Header validitását tárolja.
- Use – Elemzéskor használandó tulajdonság, ami azt mutatja, hogy egy fejléct hányszor használnak a program lefutása során.

Variable

A P4 program mező objektumait reprezentálja. Az osztály rendelkezik kettő konstruktorral az első, ahol nem kell megadni paramétert ez a Json objektumból való visszaalakítás miatt volt szükséges. A másik pedig, ahol megadjuk az osztály *Type* és *Name* értékét, ez a könnyebb példányosítás miatt került bele. Osztály tagjai:

- Type – Mező típusa (például: bit<24>).
- Name – Mező neve
- IsInitialize – Azt jelzi, hogy a mező inicializált állapotban van-e. A mező, akkor kerülhet inicializált állapotba, ha egy írott csúcsban szerepel, vagy ha egy Header-ben van, akkor az adott Headert validra állította a felhasználó.
- Write – Elemzésnél használt változó, ami jelzi, hogy egy változót hányszor írtak felül.
- Read - Elemzésnél használt változó, ami jelzi, hogy egy változót hányszor olvastak ki. Ebben az esetben hibát jelzünk, ha a változó nincs inicializálva.
- Modified – Jelzi, hogy a változót írták-e, ha újra olvassák, akkor az értéke hamis lesz.
- ModifiedAndUse – Azt számolja, hogy módosítások után hányszor volt használva.

Nézetmodellek

Ezek az osztályok megtalálhatóak az Angular modelljei között ugyanilyen reprezentációval.

Gráf modell



25. ábra - Nézet gráf osztálydiagram

A *Nézet gráf osztálydiagram* ábráján látszódik, hogy modellben rendre ugyanazok a változók szerepelnek, mint a már említett Gráf modellben, annyi kivétellel, hogy a Guidokat és a Colorokat string típusok, az enumerációkat, pedig int típusok váltották fel. A Color típusokat hexadecimális alakba alakítottam át. Egy új segéd változó került bele, ami a Number és azt jelzi, hogy a gráf melyik szintjén helyezkedik el az adott csúcs, vagyis a kezdő csúcsból a legrövidebb úton hányadik csúcsként szerepel.

FileData

A *FileData osztálydiagram* ábráján látható a kliens oldalról való fájl vagy szöveg feltöltésére szolgáló objektum, valamint a fájl oldalon való böngészésre.



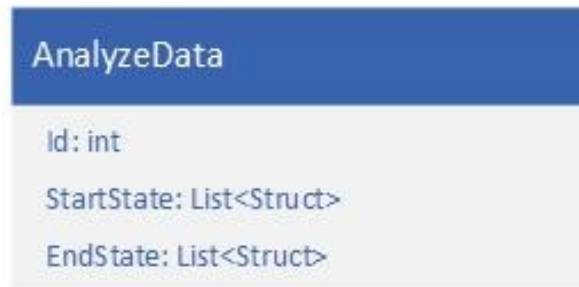
26. ábra - FileData osztálydiagram

Mezőinek leírása

- Id – Egyedi azonosító, amit az adatbázisba feltöltéskor generálódik.
- Name – A fájl neve.

- Content – A fájl tartalma vagy a felhasználó által beírt szöveg.
- CreateDate – Létrehozás dátuma, ez is az adatbázisba történő feltöltéskor generálódik.

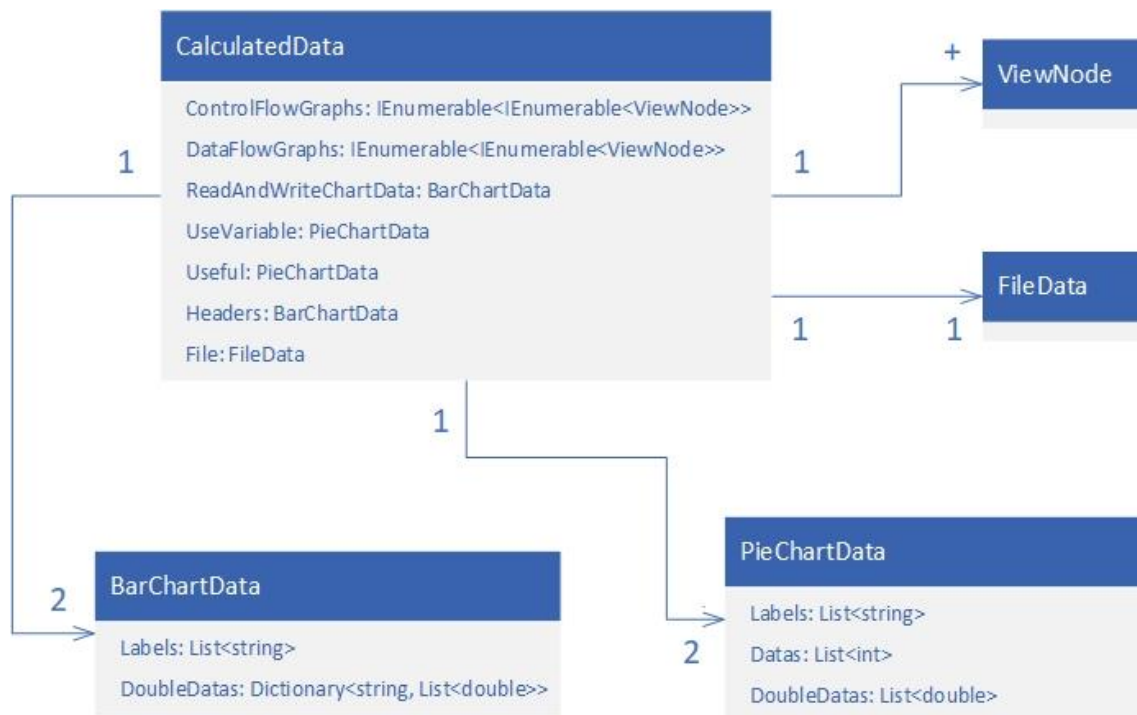
AnalyzeData



27. ábra - AnalyzeData osztálydiagram

Az *AnalyzeData* osztálydiagram ábráján a nézettől megkapott környezetek látahtóak. Tartalmazza a kezdeti- és végállapotokat, valamint egy egyedi azonosítót, ami alapján később meg lehet határozni, hogy melyik állapotok futottak le többször és a gráfokat ez alapján szűrni.

CalculatedData



28. ábra - CalculatedData osztálydiagram

A *CalculatedData* osztálydiagram ábrán látható, hogy elemzés után milyen objektumot adok vissza a nézetnek. Tartalmazza egy vagy annál több gráf *ViewNode*

reprezentációját, egy *FileData*-t, mivel a felhasználónak lehetősége van elmenteni a kódját és kettő-kettő oszlop- és kördiagram objektumát.

P4ToGraph



29. ábra - P4ToGraph osztálydiagram

Olyan statikus osztály, ami P4 programot az elemzendő gráffá alakítja. A *P4ToGraph osztálydiagram* ábrán láthatóak az osztály konstans adattagjai, amik rendre ugyanazt az értéket veszik fel, mint változó nevük csak kis betűvel. Az osztály kettő publikus művelete segítségével előállítható a vezérlésfolyamgráf és az adatfolyamgráf. A privát műveleteket három csoportba tettem értelmezésük szerint. Az első csoport az, amit a *ControlFlowGraph* metódus lefutásakor használok fel, a második csoportot a *DataFlowGraph* metódus meghívásakor használok, az utolsó pedig egy generikus metódus. A fontosabb metódusok leírása lejjebb található. A *Method*, valamint *Node* végződésű metódusok értelmezése pedig, hogy előbbi

rendre a vezérlésfolyamgráf csúcsot generálja le az adott kód eleméhez, utóbbi pedig ugyanezt teszi, csak egy vezérlésfolyamgráf csúcsból generálja le az adatfolyamgráf csúcsokat.

ControlFlowGraph

A paraméterként megadott inputból előállít egy vezérlésfolyamgráfot. Megkeresi a gráf magjaként szolgáló Ingress Control-t. A gráf felépítését a privát metódusok végzik. Ez a függvény az elején hozzáad a gráfhoz egy Start csúcsot, majd a végén kiszűri a *Skip* csúcsokat és hozzáfűz egy End csúcsot a végéhez.

Bemenő paraméterek

- input – A P4 kód, amiből előállítjuk a vezérlésfolyamgráfot. Referencia szerinti átadás, azért történik, mert a kódot megtisztítom a felesleges elemektől és ezt felhasználom a *DataFlowGraph* metódusnál.

Visszatérési érték

Graph típusú tér vissza, ami az elkészített vezérlésfolyamgráf referenciájára mutat.

Search

A gráf építés központja, ami a kapott *method* listát feldolgozza és meghívja a megfelelő metódust. Elágazásnál megkeresi az igaz ágot, valamint, ha van hamis ág, akkor azt is. Más esetben, pedig megnézi, hogy egy táblát akarnak meghívni, vagy egy akciót. Utolsó esetben pedig lekezeli, mint egy egyszerű utasítást.

Bemenő paraméterek

- graph – A vezérlésfolyamgráf, amihez a csúcsokat hozzá kell adni.
- currentNodes – Csúcsok listája, amikhez az új csúcsot hozzá kell fűzni.
- method – String lista, amiben a P4 kód egy blokkja van benne szétszeletelve.
- ingressMethod – Ingress Control blokk.
- color – A szín, hogy milyen legyen az él, amivel az új csúcsot hozzá fűzöm a currentNodes-ban lévő csúcsokhoz.

Visszatérési érték

Egy lista csúcsokkal, amik azokat tartalmazzák, melyeket utoljára adtam hozzá a gráfhoz.

DataFlowGraph

A folyamat először végig iterál a *controlFlowGraph Nodes* mezőjén és rendre meghívja a megfelelő privát metódusát a csúcs *Type* mezője alapján. A

visszkapott érték rendre egy *Graph* típusú objektum, amit egy kulcs-érték pár listába helyezem. Az kulcs a *controlFlowGraph* csúcsának egyedi azonosítója az érték pedig a visszkapott gráf. Ezek után külön álló részgráfokat a *controlFlowGraph*-on történ szélességi bejárás közben egymáshoz fűzi.

Bemenő paraméterek

- input – A P4 kód, amiből előállítjuk a vezérlésfolyamgráfot.
- controlFlowGraph – Az inputból előállított vezérlésfolyamgráf.

Visszatérési érték

Graph típusú érték, ami az elkészített adatfolyamgráf referenciájára mutat.

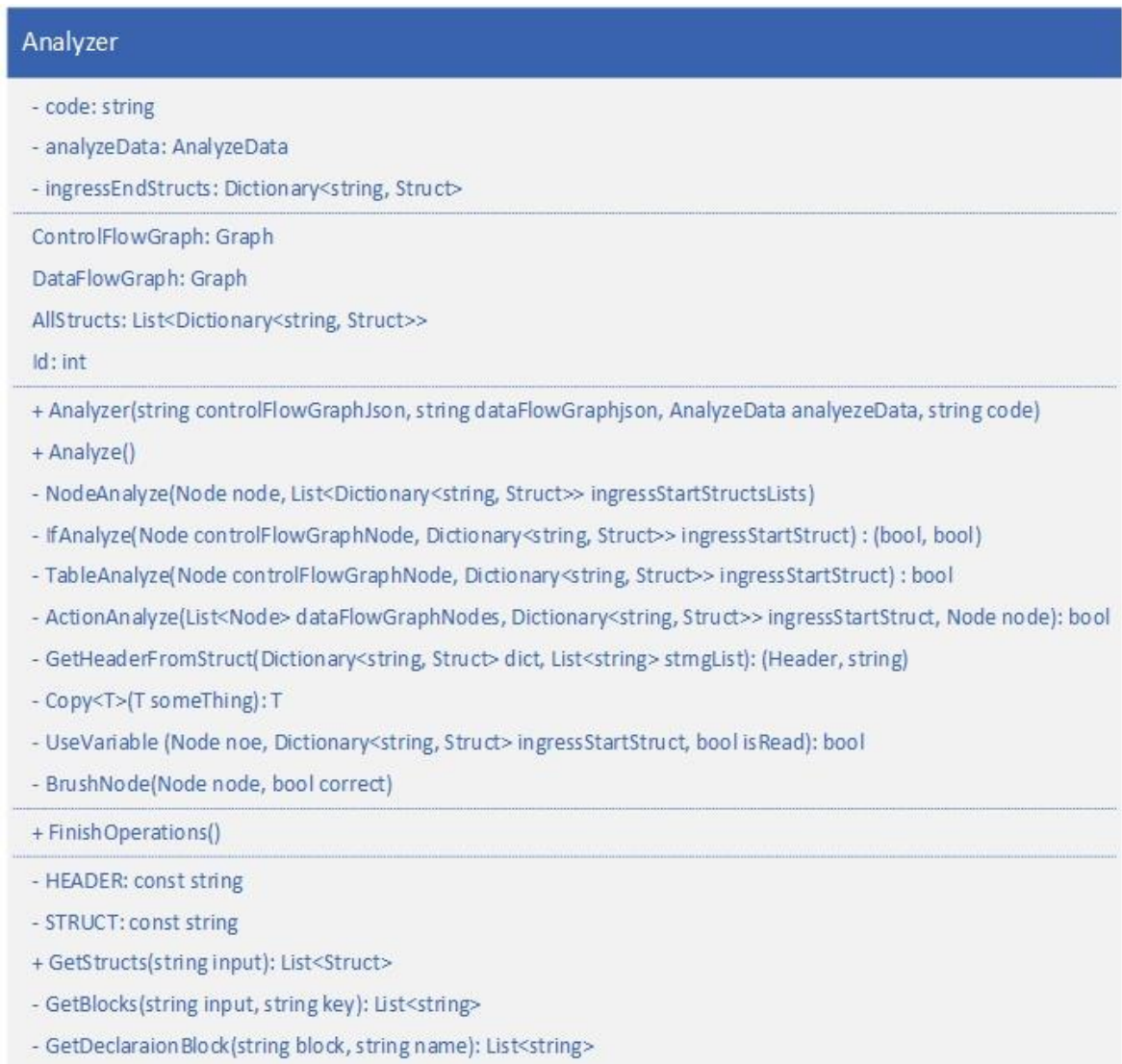
BFSHelper

Segéd függvény, ami a vezérlésfolyamgráf szélességi bejárásában segít. A megértéséhez egy fontos információ, hogy nem tartozik minden vezérlésfolyamgráf csúcshoz egy adatfolyam részgráf, ezért is kell külön szedni a *parentNode*-ot és a *currentNode*-ot.

Bemenő paraméterek

- parentNode – Vezérlésfolyamgráf csúcs, aminek gyerekeit (adatfolyam részgráfot) hozzá akarjuk fűzni a currentNode éllistájának gyerek csúcsainak a hozzájuk tartozó adatfolyam részgráfhoz.
- currentNode – Vezérlésfolyamgráfcsúcs, az előbb leírt logika miatt kell.
- graph – Felépített adatfolyamgráf.
- graphs – DataFlowGraph-ban felépített kulcs-érték pár lista.
- queue – A szélességi bejáráshoz használt sor.

Analyzer



30. ábra - Analyzer osztálydiagram

Elemzést segítő adatszerkezet. Az osztályt lehet példányosítva használni, de van statikus függvénye is. Az *Analyzer osztálydiagram* ábrán látható, hogy több részre bontottam az osztály felépítését. Az első részben a példányosított egyed által használt privát tagok kerültek, a másodikban pedig a szintűgy létrehozott objektum által használt publikus mezők. Utána jön az osztály konstruktora, valamint az *Analyze* publikus függvény, amit az elemzéshez kell meghívni és ezt követik a privát függvények, amik az elemzést elősegítik. A *FinishOperations* egy olyan függvény, amit az *Analyze* után hívunk meg közvetlenül, itt állítom elő a végső állapotot. Ezek után jön az a statikus rész, ami a P4 kód *struct*, *header* és mező részét dolgozza fel, mivel ez erősen kapcsolódik az elemzéshez, ezért kapott ebben az osztályban helyet. Osztály mezői:

- code – P4 kód.
- analyzeData – Ebben vannak benne a Struct-ok, Header-ök és Variable-ök kezdő- és végértékei.
- ingressEndStructs – Olyan kulcs-érték pár, ahol az érték egy Struct, a kulcs pedig a Ingress Control-on belüli neve, amit a függvényhívásból szedek ki.
- ControlFlowGraph – Az előzetesen előállított vezérlésfolyamgráf.
- DataFlowGraph – Az előzetesen előállított adatfolyamgráf.
- AllStructs – Az elemzés során a kezdeti értékekből nagyon sok jöhet létre a sok elágazás miatt, az így keletkezett struktúrákat tárolom itt.
- Id – Az analyzeData Id mezőjét adja vissza.

Analyze

Először előállítja az Ingress Control paraméterei alapján a már említett *ingressEndStructs*-ot, valamint ugyanilyen módon az kezdőállapotokat. Majd a *ControlFlowGraph* Start csúcsának összes gyerekére meghívja a *NodeAnalyze* metódust.

NodeAnalyze

A csúcsnak növeli a *Use* mezőjét. majd az összes kezdeti állapotra meghívja a csúcs típusának megfelelő elemző függvényt. Amennyiben az elemzés sikeresen lefutott a csúcs színét zöldre állítom. Továbbiakban pedig, ha jól lefutott és még nem a végén járunk, akkor a csúcs többi gyerekére is meghívom ezt a metódust, különben pedig megnézem, hogy a környezetet hozzáadjam-e az összes környezethez és, ha igen akkor hozzáadom.

Bemenő paraméterek

- node – Vezérlésfolyamgráf csúcs,
- ingressStartStructsLists – Analyze által előállított kezdőállapotok.

GetHeaderFromStruct

A kapott kulcs-érték pár listából a *stringList* alapján megpróbál megkeresni egy Header-t.

Bemenő paraméterek

- dict – Struktúrák szerkezete, amiben keresem a Header-t.
- stringList – Azon változók neveinek listája, amiknek segítségével megtalálhatom a Header-t.

Visszatérési érték

Ha megtalálja a *Header*-t, akkor a *Header*-rel tér vissza, valamint a listának a rá következő elemével, ha nem találta meg, akkor nullal és üres stringgel tér vissza.

UseVariable

Először az előbb említett *GetHeaderFromStruct* függvénnyel megpróbálja megtalálni a *Header*-t, amiben a változó helyet foglal, ha ez sikerült, akkor növeli a *Header* felhasználtságát, majd a változónak az írás, vagy az olvasás mezőjét értelemszerűen. Végül pedig színezi a csúcsot.

Bemenő paraméterek

- node – A csúcs, aminek szövege alapján keresem a változót.
- ingressStartStruct – A környezet, amiben keresem.
- isRead – Ha olvasni akarjuk a változót, akkor igaz, ha írni, akkor hamis.

Visszatérési érték

A változónak az inicializáltságával tér vissza, ha nem találta meg a változót, akkor alapértelmezetten igazzal.

BrushNode

A csúcsot a *correct*-ség alapján színezi, amennyiben a csúcs még nem volt színezve, akkor jó lefutás esetén zöld, hibás lefutás esetén piros színűre. Ha már volt színezve, akkor, ha pirosra volt színezve, de most jól lefutott, akkor sárga lesz, ha zöldre volt színezve és most rosszul futott le, akkor is sárga lesz. Ha egy csúcs sárga lett, akkor az sárga is marad, mivel jelzi, hogy lehet, hogy rosszul fut le, de lehet, hogy jól, ezt a program nem tudja biztosra eldönteni.

Bemenő paraméterek

- node – A csúcs, amit színezni szeretnénk.
- correct – A csúcsban lévő érték helyessége.

FinishOperations

Miután lefutott az *Analyze* metódus, ezzel véglegesítem az elemzést. Ilyenkor megnézem, hogy a végállapot szerint melyik fejléceket csomagolom vissza. A visszacsomagolandó fejlécek változóit megnézem, hogy voltak-e módosítva, és ha igen, akkor jelzem, hogy módosítás után fel voltak használva. Ezek után a vezérlésfolyamgráf csúcsok színezését állítom be, mivel ezek színezését elemzés közben nehéz lett volna állítani.

GetStructs

A kapott kódból előállítja a *Structokat* tartalmazó listát. A *Mintakód* ábrán látható egy olyan szerkezet, amit átalakít a *Struktúra modell* alakjára.

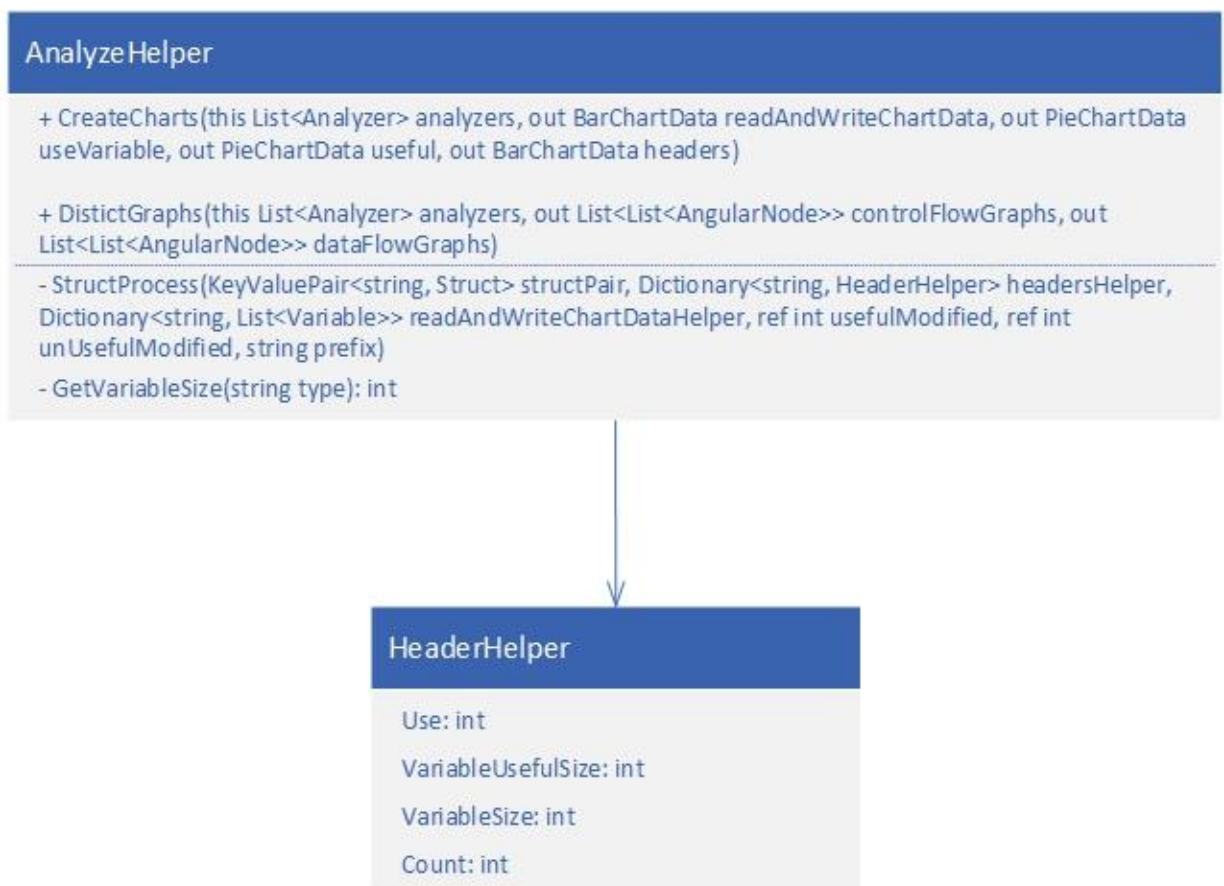
Bemenő paraméterek

- input – P4 kód.

Visszatérési érték

Struktúrák listájával tér vissza.

AnalyzeHelper



31. ábra - AnalyzeHelper osztálydiagram

Az *AnalyzeHelper* egy segéd osztály, ami egy *Analyzer* listából segít adatokat előállítani. Ez egy statikus osztály és az *AnalyzeHelper osztálydiagram* ábrán látszik, hogy kettő publikus metódusa van, valamint felhasznál egy segéd osztályt a *HeaderHelper*-t.

CreateCharts

Egy *Analyzer* listából előállítja a grafikus megjelenítéshez szükséges adatokat.

Bemenő paraméterek

- analyzers – Analyzer lista, amire meg lett hívva a metódus.

Visszatérési érték

A metódusnak nincsen visszatérési értéke, de van négy kimenő a paramétere, így ebben a részben azokat írom le:

- readAndWriteChartData – Olyan BarChartData objektum, ami a mezőkről tartalmazza, hogy hányszor lettek írva, illetve olvasva.
- useVariable – Olyan PieChartData objektum, ami a mezőkről tárolja, hogy hányszor voltak összesen felhasználva.
- useful – Olyan PieChartData objektum, ami azt számolja össze, hogy a változók hányszor voltak hasznosan felhasználva, illetve nem hasznosan.
- headers - Olyan BarChartData objektum, ami tartalmazza, hogy egy fejléc hányszor volt felhasználva, valamint a mezőinek méretét, amit meg lehet számolni, és hogy átlagban mekkora méretű a felhasznált változók összessége.

DistinctGraphs

Mivel egy kezdőkörnyezetet többször is megadhat a felhasználó, de a gráfokat nem akartam sokszorozva megjeleníteni, ezért ezzel a metódussal szűröm ki, hogy egy adott kezdő- és végkörnyezethez tartozó gráf csak egyszer jelenjen meg a felületen.

Bemenő paraméterek

- analyzers – Analyzer lista, amire meg lett hívva a metódus.

Visszatérési érték

A metódusnak nincsen visszatérési értéke, de van kettő kimenő a paramétere, így ebben a részben azokat írom le:

- controlFlowGraphs – A vezérlésfolyamgráfok listája már kliens oldalnak átadható formátumban.
- dataFlowGraphs – A adatfolyamgráfok listája már kliens oldalnak átadható formátumban.

HeaderHelper

Olyan segéd osztály, ami a fejléceket felhasználását segíti kiszámolni. Mezői:

- Use – A fejléc felhasználásának számosságát tárolja.

- VariableUsefulSize – Azon változók méretének összege, amit ténylegesen felhasznált a program futás közben.
- VariableSize: - Változók összmérete.
- Count – Fejléc számossága a programban. Abban különbözik a Use-től, hogy abban az esetben a Header-ök Use mezőjét számoljuk össze, itt pedig csak előfordulást számolok.

FileHelper



32. ábra - FileHelper osztálydiagram

A *FileHelper* osztálydiagram ábráján egy olyan segéd osztály látható, ami az elemzendő kód megtisztításában, kód részletek megtalálásában segít. Rövid leírások:

- InputClean – A kapott inputot regex kifejezések segítségével megtisztítja, valamint olyan formára hozza, ami biztosabbá teszi, hogy a feldolgozás közben nem keletkezik váratlan esemény.
- GetMethod – A kapott inputból a firstEqual-től megkeresi a startChar-t és az endChar-ig bezárólagosan visszaadja a blokkot. Mivel ott blokkokat adok vissza, ezért számolni kell, hogy csak akkor térjen vissza, ha ugyanannyi start- és end karaktert talált.
- SplitAndClean – A kapott inputot feldarabolja szóközők alapján és kiveszi a lista első és utolsó elemét. Ez azért szükséges, mivel ezt blokkokra hívom meg, így ki kell szedni belőle a szeparáló jelet.
- GetIngressControlName – A kapott P4 kódból megkeresi az Ingress Control nevét és visszaadja azt.

StringExtensions

StringExtensions
+ AllIndexesOf(this string str, string value): List<int>
+ SubStringByEndChar(this string str, int startIndex, char end): string

33. ábra - String kiegészítők

A *String kiegészítők* ábrán láthatóak azok a plusz stringre végrehajtható metódusok, amiket a programban felhasználtam. Rövid leírás:

- AllIndexesOf – A kapott stringben megkeresi az összes olyan pozíciót, ahol a value szerepel benne és ezt egy listában visszaadja.
- SubStringByEndChar – A kezdő indextől az end karakterig bezárólagos visszaadja a string egy részét.

GraphExtensions

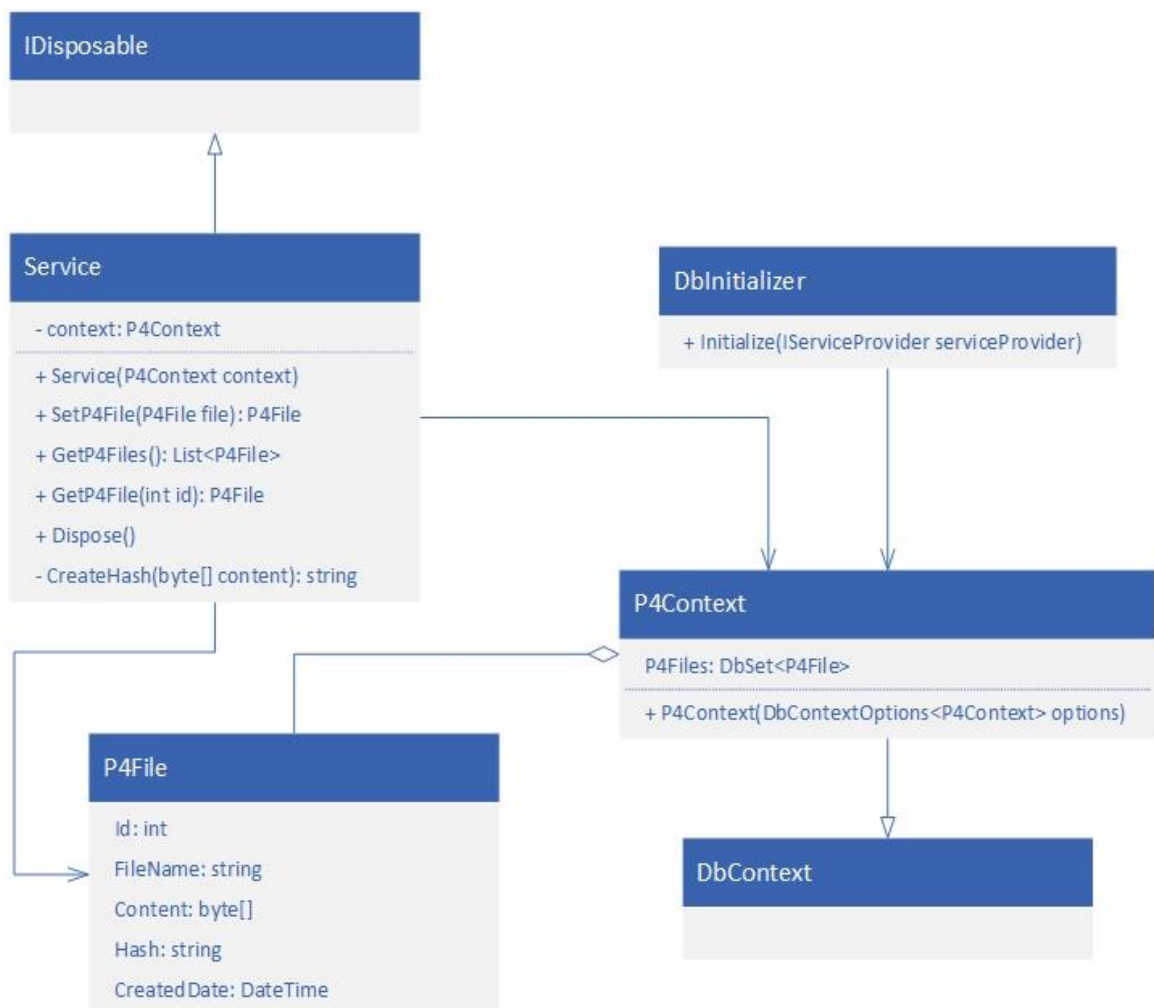
GraphExtensions
+ MainNodes(this Graph graph): List<Node>
+ MainNodes(this List<Node> nodes): List<Node>
+ EndNodes(this Graph graph): List<Node>
+ Serialize(this Graph graph): IEnumerable<ViewNode>
- GenerateLevel(List<ViewNode> angularGraph, List<Node> currentNodes, int level): (List<Node>, int)
- CheckExisting(List<Node> childNodes, List<ViewNode> angularGraph)
+ ToJson(this Graph graph): string
- GetNodeJson(Node node): string
- GetEdgeJson(Edge edge): string
- GetColorJson(Color color): string
+ FromJson(this Graph graph, string jsonString)
- GetObjects(string jsonString): List<string>
- GetFirstJsonObject(string jsonString, string firstCharPair, out string firstObject, out string otherObjects): bool
- GetEdges(Graph graph, List<string> edges, Node node): List<Edge>
- GetColor(string jsonString, string attributeName, out Color color): string
- DeserializeJson<T>(string jsonString): T

34. ábra - Graph kiegészítők

A *Graph kiegészítők* ábrán látható, hogy a *Graph* modellnek milyen kiegészítő metódusai vannak. Azért kerültek külön, mivel az egy adatleíró osztályszerkezet. Rövid leírás:

- MainNodes – Mindkét esetben a csúcsokból kiszedi azokat a csúcsokat, amikbe a csúcshalmazon belülről nem mutat él.
- EndNodes – Azon csúcsok listája, amiknek élei nem mutatnak a csúcshalmazon belüli csúcsra.
- Serialize – A gráfot átalakítja ViewNode lista szerkezetre.
- ToJson – A gráfból json objektumot állít elő.
- FromJson – Egy json objektumból felépíti a gráfot.

3.2.2. Persistence



35. ábra - Perzisztencia réteg

A *Perzisztencia réteg* ábrán látható a perzisztencia felépítése. A réteg csak az olyan felhasználók miatt készült el, akik nem tudnak P4 kódot előállítani és így az oldalról elérnek fájlokat. Ennek megfelelően csak a fájlokat tartalmazó tábla szerepel az adatbázisban.

P4Context

Adatbázis szerkezetét leíró osztály, amit így kötelezően a *DbContext* osztályból kell származtatni. Az osztály egy mezője van, ami az adatbázisban tárolt *P4File*-ok lekérdezésére szolgál, valamint egy konstruktorral rendelkezik.

DbInitializer

Egy statikus osztály, aminek annyi a feladata, hogy a program elindításakor létrehozza az adatbázist, ha az még nem létezett.

P4File

Fájlokat leíró adatszerkezet. Mezőinek leírása:

- Id – Adatbázis által generált egyedi azonosító
- FileName – Tárolt fájl neve.
- Content – Fájl tartalma bájt tömbként tárolva.
- Hash – Fájl tartalma SHA256 algoritmussal [8] kódolva. Segéd változó, hogy az adatbázisba ne kerüljön kettő ugyanolyan Contenttel rendelkező fájl. Ezzel az algoritmussal generált stringeket összehasonlítani kevesebb erőforrásba kerül, mint a byte tömbök összehasonlítása.
- CreatedDate – Létrehozás dátuma. Mindig az adatbázisba feltöltés idejét jelöli.

Service

Kiszolgáló osztály, aminek segítségével egyszerűsödik az adatbázis használata. Az osztály az *IDisposable* interfész lett származtatva ennek köszönhetően a memóriában foglalt helye egyből felszabadul, ha a változó már nincs használva. Egy darab privát adattagja van, ami az adatbázis kapcsolatot létesítő *P4Context* objektum. Metódusainak leírása:

- Service – Konstruktor, a privát adattagját beállítja a kapott *P4Context* objektumra.
- SetP4File – A kapott *P4File*-nak legenerálja a Hashét, beállítja a CreatedDate-jét és felölti az adatbázisba, majd visszatér a feltöltött objektummal. A paraméterben lévő fájl neve és tartalma kötelezően feltöltendő, ennek hiányában kivételt dob a metódus. Ha valamilyen oknál fogva nem sikerül a feltöltés például már van ilyen tartalommal fájl, akkor null értékkel tér vissza.

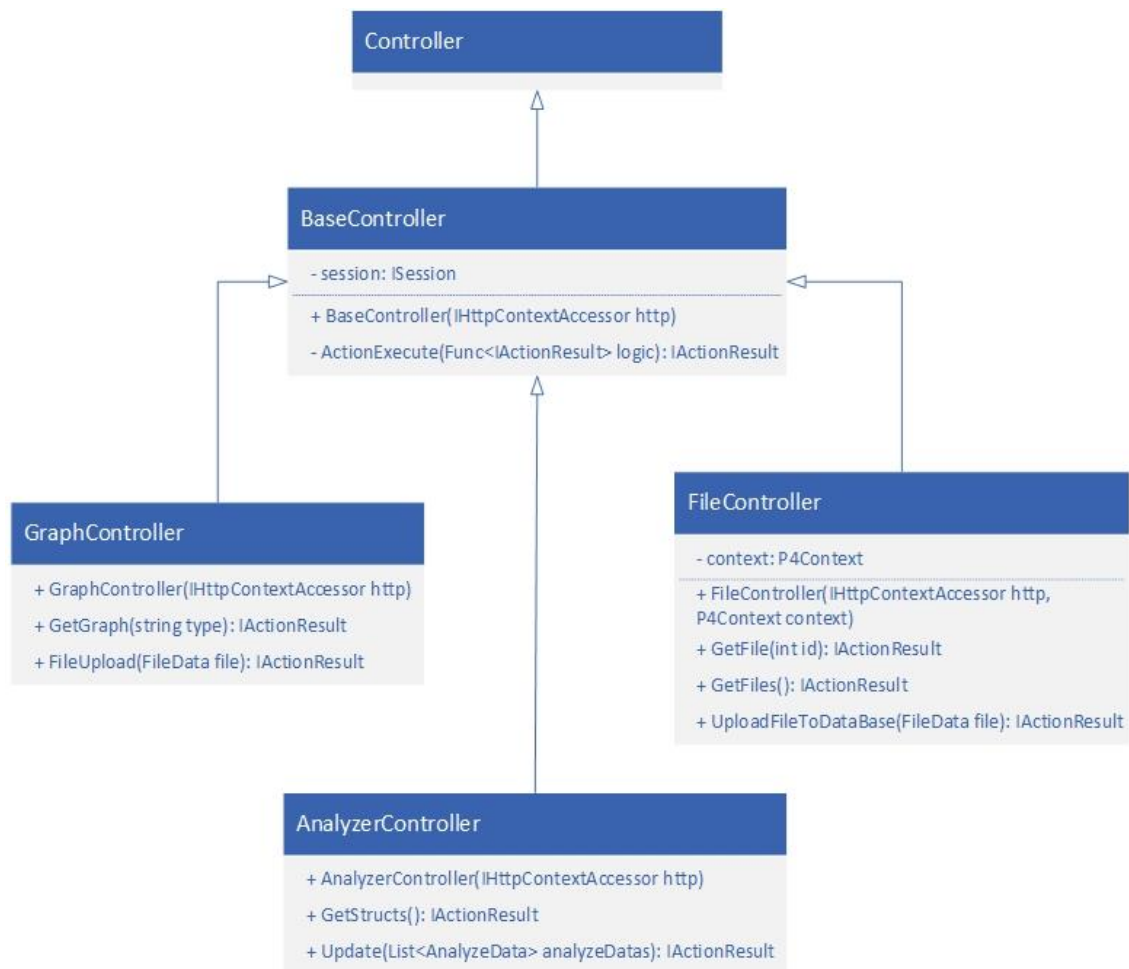
- GetP4Files – Az adatbázisban lévő összes fájl adatát visszaadja egy listába. A listában szereplő P4File objektumok Id, FileName és CreatedDate mezőit tölti fel értékkel.
- GetP4File – Az id alapján lekérdezi az egész P4File tartalmát és visszatér vele, ha nincs ilyen id-val fájl, akkor kivételt dob.
- Dispose – Az IDisposable-ből származtatás miatt kötelezően megadandó metódus.
- CreateHash – A fájl tartalmából előállítja az SHA256 algoritmussal a hasht.

3.2.3. AngularApp

Az alkalmazás megjelenítéséért felelős webes applikáció.

Controllers

A *Kontrollerek felépítése* ábrán látható, hogy a kontrollerek milyen módon vannak felépítve. Programozói konvenció, hogy minden kontrollert a *BaseController*-ből kell származtatni, valamint, hogy minden Api hívás metódusát az *ActionExecute* metódusba kell beleépíteni az alapvető hibakezelés miatt.



36. ábra - Kontrollerek felépítése

BaseController

A Rest Api alap kontrollere, ami a *Controller* osztályból van származtatva. Egyetlen védett adattagja a *session* változó. Konstruktórában egy *IHttpContextAccessor* típusú objektumot vár, aminek *Session* adattagjával feltölti az előbb említett *session* változót.

ActionExecute

Védett metódus, amit minden controllerben kötelező használni a függvények implementálásakor. Tartalmazza az alapvető hibakezelést, ezzel biztosítva a hibátlan működést. Bemeneti paraméterébe egy olyan funkciót kap, aminek a visszatérési értéke *IActionResult*, ezt a metódust megpróbálja lefuttatni. Sikeres lefutás esetén a metódusban meghatározott adatszerkezettel tér vissza, minden más esetben két féle hibakezelés van. Az első, ami az *ApplicationException*-t kapja el, ez olyankor van, amikor a programban programozói utasításra keletkezik egy kivétel.

A második esetben egy nem kezelt hiba keletkezik. Mindkét esetben hibaüzenettel térünk vissza és *BadRequest* hibakódot adunk.

GraphController

A gráfok lekérdezéséért és a P4 kód feltöltéséért, valamint gráf generálásért felelős kontroller.

GetGraph

Típus: `HttpGet`

Visszatérési érték: `IEnumerable<ViewNode>`

Visszaadja az adott kulccsal Sessionben tárolt gráfot.

Ellenőrzést végez arra, hogy a `type` egy kulcs érték-e, amivel ténylegesen tárolhatunk gráfot, valamint, ha nem talál gráfot, akkor azt is jelzi, hogy ilyenkor még nem történt fájl feltöltés.

FileUpload

Típus: `HttpPost`

Visszatérési érték: `FileData`

Megcsinálja a kapott kódból a gráf generálást. Elmenti Sessionbe a gráfokat és a kódot. Majd visszatér a kapott kóddal.

AnalyzerController

Elemzésért felelős kontroller.

GetStruct

Típus: `HttpGet`

Visszatérési érték: `List<Struct>`

Ellenőrzést végez, hogy töltött-e fel fájlt a felhasználó, ha igen, akkor a fájlból legenerálja a `Struct`-okat és visszaadja.

Update

Típus: `HttpPost`

Visszatérési érték: `CalculatedData`

Ellenőrzést végez, hogy volt-e feltöltött fájl és a gráfok is generálásra kerültek. A megadott környezetek alapján megcsinálja az elemzést és visszatér az elemzés adataival.

FileController

Adatbázisba tölti fel a fájlokat, valamint onnan lekérdezi őket.

GetFile

Típus: HttpGet

Visszatérési érték: FileData

Visszaadja az adatbázisban adott id-val tárolt fájlt, ha nincs ilyen, azt jelzi.

GetFiles

Típus: HttpGet

Visszatérési érték: List<FileData>

Visszaadja az adatbázisban található összes fájlt a tartalma nélkül.

UploadFileToDataBase

Típus: HttpPost

Visszatérési érték: FileData

Feltölti a fájlt az adatbázisba, ha nem sikerült valamiért azt jelzi.

Extensions

SessionExtension



37. ábra - Session kiegészítő és kulcs enumeráció

A *Session kiegészítő és kulcs enumeráció* ábrán látható, hogy a Sessionben való tárolásához, milyen extra funkciókat hoztam létre. A *Graph*-ra azért kellett külön függvény, mivel itt nem a keret által használt Json konvertert használom. A *Key*

enumeráció, azért került feltüntetésre, mivel itt vannak felsorolva a kulcsok, amik egyedi azonosítóval látják el a Sessionben tárolt adatokat.

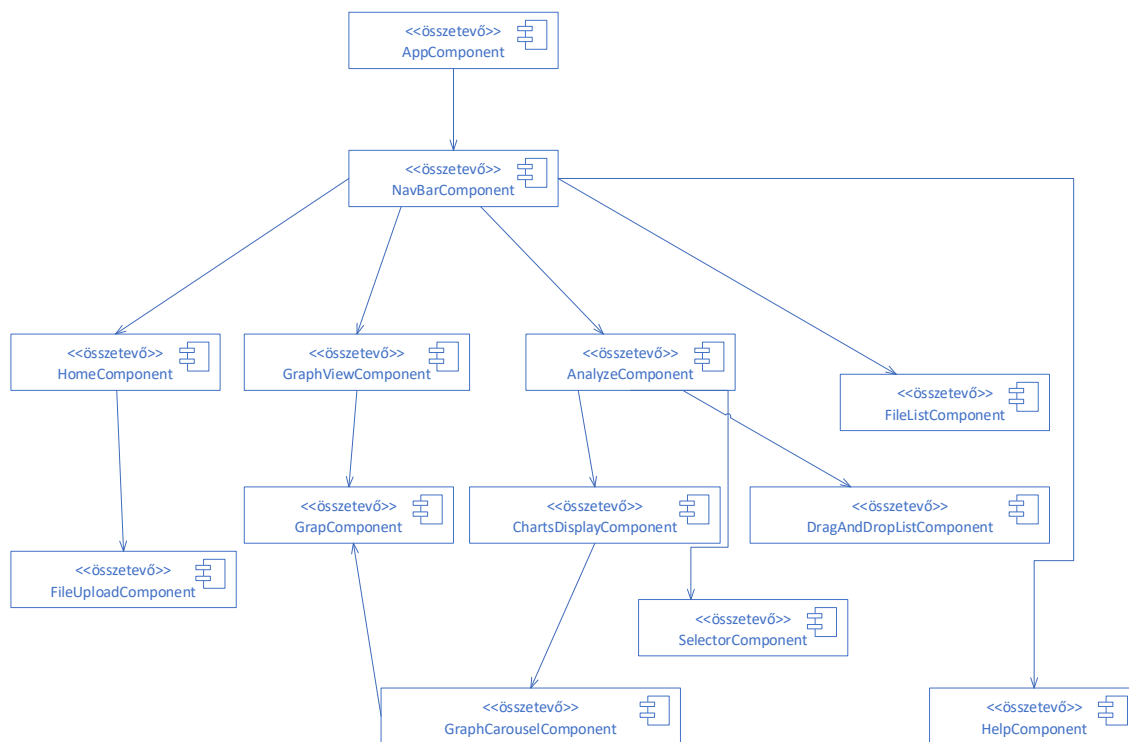
FileConverter



38. ábra - FileConverter osztálydiagram

A *FileConverter osztálydiagram* ábrán látható, hogy ebben az osztályban kettő metódus kapott helyet, ezek funkciója, hogy az adatbázis által használt *P4File* és a nézet által használt *FileData* között könnyen tudjak váltani, szóval egyikből generálni a másikat.

ClientApp



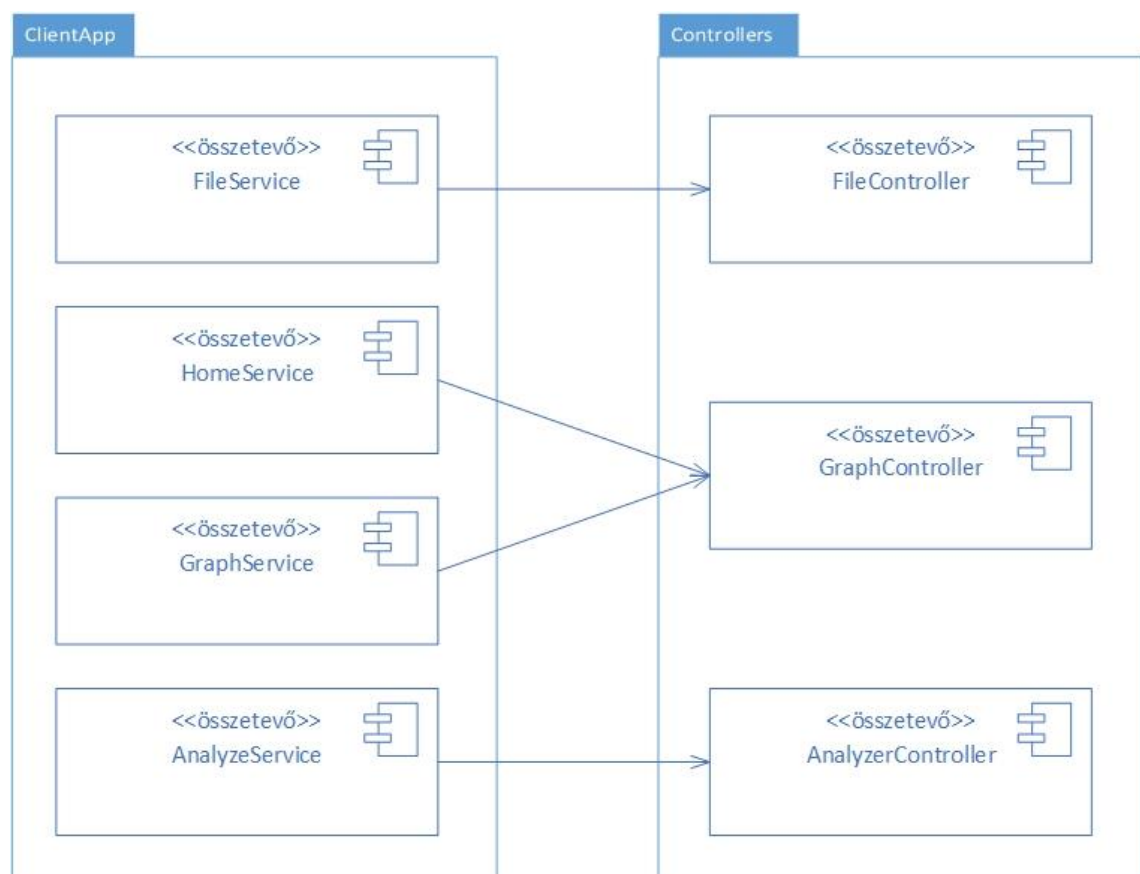
39. ábra - ClientApp komponens diagram

A *ClientApp komponens diagram* ábrán látható a kliens oldali megjelenítésért felelős Angular applikáció szerkezete. A program több kisebb újrahasznosítható komponensből áll. Az oldal kerete a *NavBarComponentben* található, ahol is helyet foglal egy fejléc, valamint egy oldal navigációs sáv, ami ki- és becsukható. A weboldal láthatóan öt oldalból áll, ebből három elérhető a navigációs sáv segítségével. A

programban több nyílt forráskódú kiegészítőt használok a fontosabbak a következők:

- d3 [9] – A gráf megjelenítése miatt használom. D3-as objektumok formázásában segít.
- d3-graphviz [10] – Gráf rajzoló javascript könyvtár, ami a Graphviz [11] könyvtárra épül. Azért választottam ezt, mivel megjelenése konzisztens, valamint hivatalos. Felhasználása nem egyszerű, mindent a programozóra bíz.
- ng2-charts [12] – Diagramokat ábrázoló komponens. Választásom azért esett erre, mivel megjelenése reszponzív és használata könnyű, valamint a megbízhatósága miatt (hetente több tíz-ezer felhasználó tölti le).
- ngx-store [13] – Böngésző cache tartalmának kezelését megkönnyítő szolgáltatásokat tartalmaz.

Szerver kommunikáció

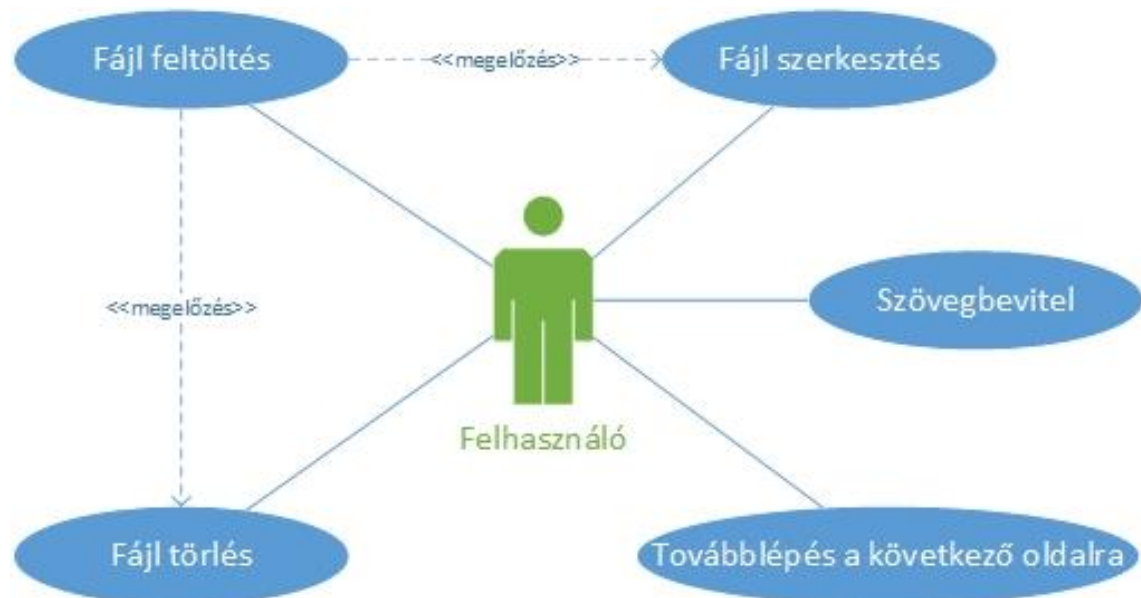


40. ábra - Szerver kommunikáció

A *Szerver kommunikáció* ábrán látható, hogy mely Angular szolgáltatás melyik Rest Api kontrollert hívják. A hívás http protokoll alapján folytatódik és Json objektumok

közlekednek. A szerver oldali kommunikáció számossága minimálisra van csökkentve azzal, hogy a weboldali session-ben eltárolom a kapott adatokat. Cél, hogy csak akkor történjen szerver hívás, amikor az szükséges.

HomeComponent



41. ábra - Kezdő oldal felhasználói eset diagram

A *Kezdő oldal felhasználói eset diagram* ábráján látható, hogy a felhasználónak milyen funkciók és lehetőségek állnak rendelkezésére, amikor a kezdőoldalon van. A *HomeComponent*, valamint a *FileUploaderComponent* ezeket a felhasználói interakciókat képesek kiszolgálni. Egy mat-tab -os elrendezésben egy szövegbeviteli mező, valamint a *FileUploaderComponent* helyezkedik el. Egyik kitöltése kötelező a tovább lépéshez, de mindkettőt ne tudja feltölteni adattal a felhasználó. A fájl feltöltés egy drag and drop direktíva miatt történhet drag and drop módszerrel vagy fájlkezelő ablakból történő feltöltéssel. Fájl feltöltéskor ellenőrizni kell a kiterjesztést és, ha nem .p4 vagy .txt, akkor ezt jelzem, valamint a drag and drop módszernél előfordulhat, hogy több fájlt is odahúz a felhasználó, ilyenkor ezt is kezelem. A *FileUploadComponentben* lehetőséget biztosítok a fájl módosítására egy felugró dialógussal, valamint a fájl törlésére, hogy újat tölthessen fel.

GraphViewComponent

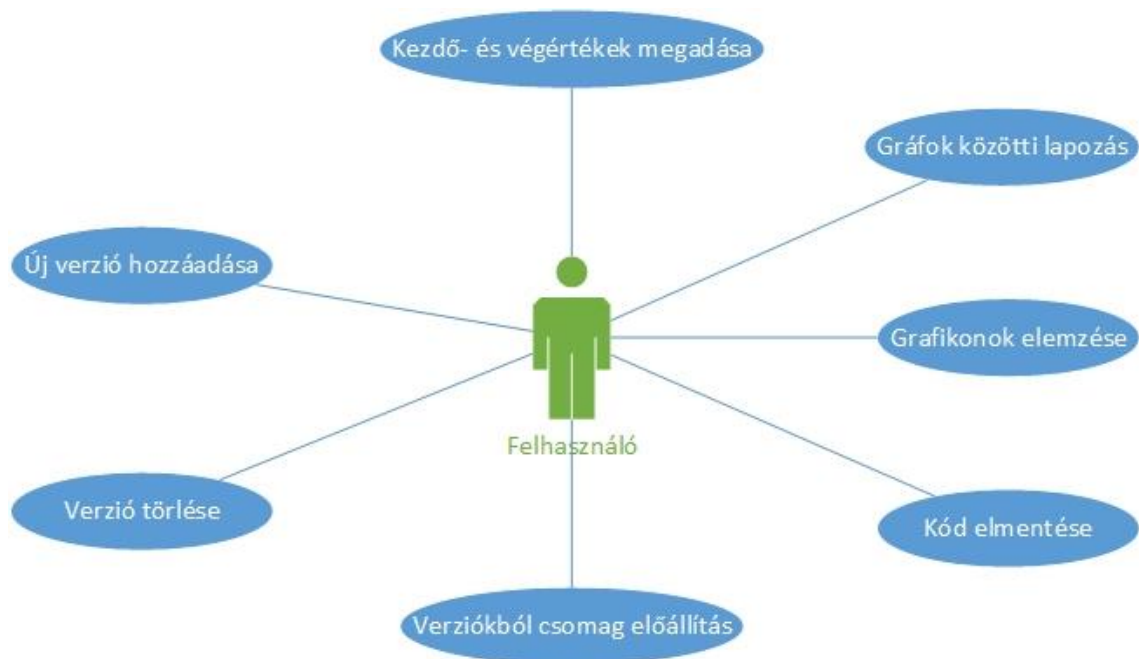


42. ábra - Gráf megjelenítő oldal felhasználói eset diagram

A *Gráf megjelenítő oldal felhasználói eset diagram* ábráján látható, hogy a felhasználónak milyen funkciók és lehetőségek állnak rendelkezésére, amikor a gráf megjelenítő oldalon van. Itt a felhasználónak különféle gráf interakciókat kellett biztosítani. Ehhez hasonló a gráf bejárás, amit egy szélességi bejárással [14] szimulálok. A *GraphViewComponentben* is mat-tab -os elrendezést használok, de itt dinamikusán lehet hozzáadni és törölni belőle, kivéve az első kettő vezérlés- és adatfolyamgráfot megjelenítő tabokat, azok mindig fixek. Ezen az oldalon feugró dialógusok helyett mat-bottom-sheet -et használok, ami egy ilyen interaktív oldalhoz jobban passzol és a felhasználónak is könnyebb felhasználást biztosít.

AnalyzeComponent

A *Elemző oldal felhasználói eset diagram* ábráján látható, hogy a felhasználónak milyen funkciók és lehetőségek állnak rendelkezésére, amikor az elemzéseket végző oldalon van. Itt meg kellett oldani, hogy a felhasználó oldal váltás nélkül kedve szerint változtathasson a megadott környezeteken, ezért egy mat-stepper szerkezetbe helyeztem bele a komponenseimet. A selector komponensben lehetőséget biztosítottam, hogy dinamikusán tudjon megadni kezdő- és végállapotokat, itt mindig egy állapotnak muszáj maradnia. A *dragAndDropList* komponensben biztosítani kellett a felhasználónak, hogy a *selectorban* megcsinált környezeteket kedve szerint sorba rendezhesse, ezáltal egy csomagot létrehozva.



43. ábra - Elemző oldal felhasználói eset diagram

Erre egy drag and drop módszert biztosítottam, a fő cél volt, hogy minimalizáljam a felhasználói hiba lehetőségét és a legegyszerűbb módon tudja sorba rendezni az adatokat. A *chartDisplay* komponensben pedig a szerver oldalról kapott elemzett adatokat jeleníti meg. Itt meg kellett oldani, hogy az összes visszakapott vezérlés- és adatfolyamgráfot emészthető formában vizsgálhassa a felhasználó, ezért egy carousel komponenst csináltam, ahol a gráfok között tud váltogatni.

FileListComponent



44. ábra - Fájl oldal felhasználói eset diagram

A *Fájl oldal felhasználói eset diagram* ábráján látható, hogy a felhasználónak milyen funkciók és lehetőségek állnak rendelkezésére, amikor az fájlkezelő oldalon van. Itt

egy egyszerű mat-table található egy mat-paginator-ral. A felhasználónak lehetőséget biztosítottam szűrésre, valamint rendezésre. Továbbá a fájlok tartalmáak vágólapra másolására és a gráf lejátszás megkezdésére.

3.3. Tesztelés

A szerver és a kliens oldali tesztelés külön válik.

3.3.1. Szerver oldal

A szerver oldal C# nyelven készült, az ott található metódusokat, valamint adatbázist és Rest Api-t egység tesztekkel ellenőriztem le. A három réteget külön-külön vizsgálva teszteltem, ezért három teszt fájl készült.

ControllerTest

Ebben az osztályban a három kontroller osztályom belépési pontjait teszteltem, helyes, illetve helytelen használatra is készítettem teszteseteket. A kontrollerek használnak Session-t, illetve a *FileController*-nek adatbázis elérése is van. Előbbit Mock-olással pótoltam, utóbbinak pedig a teszteléshez létrehozott teszt adatbázis szolgált a kiváltására. A következő sorokban ismertetem a teszteseteket.

GraphController

Teszteseteimbe kitértem az üres fájl esetére, valamint olyan hibák megfogására is, amikor a programban olyan rész van, amit a megvalósításom nem képes elemezni. Ennek megfelelően a következő tesztesetek történtek meg:

- Fájl feltöltése, a specifikációnak megfelelő P4 kóddal.
- Fájl feltöltése tartalom nélkül.
- Olyan fájl feltöltése, ahol az Ingress kontrollfüggvényében *else if* elágazás található.
- Olyan fájl feltöltése, ahol valamelyik akción belül *if* elágazás található.
- Vezérlés- és adatfolyamgráf lekérdezése.
- Gráf lekérdezés egy olyan kulcs alapján, amit nem ismer a rendszer.
- Gráf lekérdezése anélkül, hogy előtte a P4 kód feltöltése megtörtént volna.

AnalyzerController

Ebben a kontrollerben csak olyan kivételt kezelek, amikor előzetes fájl felöltés nélkül lettek meghívva a belépési pontjai, ezeket tesztelem. Rövid ismertetés:

- *Struct* lista lekérdezése a Session-ben tárolt P4 programból.
- Struktúrák lekérdezése előzetes fájl feltöltés nélkül.
- Elemzés elkészítése és visszaadása. Ebben az esetben a visszaadott fájl, valamint a megjelenítendő diagramok adatait ellenőrzöm.
- Elemzés indítása anélkül, hogy az előzetes fájl feltöltés megtörtént volna.

FileController

Ebben az esetben meg kellett oldanom, hogy a kontroller konstruktorának át tudjam adni az adatbázis eléréséért felelős *P4Context* objektumot, valamint a teszt adatbázis minta adatokkal ellátását. Ismertetem a teszteseteket:

- Fájl lekérdezése az egyedi azonosítója alapján.
- Fájl lekérdezése olyan azonosítóval, ami nem található meg az adatbázisban.
- Összes fájl lekérdezése, ebben az esetben a fájl tartalma üres marad.
- Fájlok feltöltése az adatbázisba.
- Fájl feltöltése név nélkül.
- Olyan fájl feltöltése, aminek tartalmával már rendelkezik az adatbázis.

GraphTest

Ebben a részben a gráf képzést, struktúra létrehozást, valamint az elemzést teszteltem. Mindegyik esethez egy darab tesztesetet készítettem. A folyamat nagyon körülményes és pontjait nehéz megfogni, ezért készült mindegyikből csak egy eset. A tesztelésnél lényegében egy egész folyamatot tesztelek, ami lezajlik, akkor, amikor egy felhasználó használja a weboldalt. Ennek megfelelően minden tesztesethez ugyanazt a fájlt használtam fel.

PersistenceTest

Az adatbázis működését tesztelő rész. Mivel az adatbázisomra nem fektettem nagy hangsúlyt lévén, hogy egy táblából áll és kényelmi szerepet tölt be főleg, mintsem valódi üzleti logikát. Rövid leírás a tesztesetekhez:

- P4 program feltöltése az adatbázisba.
- Olyan P4 program feltöltése az adatbázisba, ami már szerepelt benne előzetesen.
- P4 fájl lekérdezése egyedi azonosítója alapján.
- Táblában nem található azonosítóval való lekérdezés.

- Összes P4 kód letöltése az adatbázisból. Ennél a működésnél a tartalom üres marad.

3.3.2. Weboldal tesztelése

A Weboldal teszteléséhez nem használtam tesztelő eszközt, hanem egy teszt jegyzőkönyvet írtam, aminek segítségével vizsgáltam az oldalam működését.

Minden tevékenységet kettő adattal fogok jellemezni, az egyik a folyamat leírása, amit végrehajtottam a másik pedig az elvárt eredmény.

- Fájl feltöltőre kattintás – Fájlkezelő ablak megnyílik.
- Megfelelő kiterjesztéssel rendelkező fájl kiválasztása – A fájl feltöltődik. A fájlfeltöltő eltűnik és a fájl neve jelenik meg a képernyőn egy módosító és egy törlő ikonnal, valamint a szövegbeviteli mezőre nem lehet átlépni.
- Módosítás ikonra nyomás – Felugró dialógusa ablak megjelenik, ahol a fájl tartalma módosítható.
- Módosítás elmentése – A felugró ablak becsukódik és kapunk egy üzenetet, hogy a módosítás sikeres volt.
- Módosítás elvetése – Dialógus bezáródik.
- Fájl törlése – A fájl eltűnik, újra megjelenik a fájl feltöltő felület és a szövegbeviteli mezőre is át lehet navigálni.
- Drag and drop funkcióval kettő vagy több fájl feltöltésének kísérlete – Feltöltés nem történik, hibaüzenetet jelez rendszer, hogy csak egy fájlt lehet feltölteni.
- Rossz kiterjesztésű fájl feltöltése - Feltöltés nem történik, hibaüzenetet jelez rendszer, hogy csak adott kiterjesztésű fájlt lehet feltölteni.
- Szövegbeviteli mezőbe elkezdünk gépelni – A fájl feltöltő részlegre nem lehet navigálni.
- Szövegbeviteli mező üresre törlése – A fájl feltöltő részleg újra elérhető.
- Érvényes kód feltöltése és tovább lépés – Az oldal átnavigál a gráf nézet oldalra, ahol a vezérlésfolyamgráf elkezd megjelenni.
- Átlépés az adatfolyamgráf tabjára – Az adatfolyamgráf tabon is elkezdődik a kirajzolás.
- Vezérlésfolyamgráf csúcsra nyomás és csúcshoz vezető út kirajzolása – A csúcshoz vezető út, valamint abból kimenő élek kiszíneződnek.

- Vezérlésfolyamgráf csúcsra nyomás és adatfolyamgráf megjelenítése – Új tabon megjelenik az adatfolyam részgráf.
- Adatfolyamgráf csúcsra nyomás és szülő csúcs mutatása – Az oldala vezérlésfolyamgráfra ugrik és a csúcs szülőjét kiszínezi.
- Szélességi bejárás szimuláció elindítása – A szimuláció elindul, szüneteltetés gomb lesz.
- Szélességi bejárás szimuláció szüneteltetés – A szimuláció leáll, lejátzás gomb lesz.
- Gráfon kívülre kattintás és színezés helyreállítása – A színezés eredeti állapotába kerül vissza.
- Következő oldalra lépés – Az oldal átnavigál az elemző oldalra.
- Kezdő, vagy végértékek megadása – A megadott elemek megjelennek a vonalon.
- Új környezet hozzáadása – Az aktuális lenyíló fül bezáródik és az újonnan létrehozott megnyílik.
- Környezet törlése – A lenyíló fül eltűnik, új nem nyílik meg.
- Következő oldalra nyomás – Egy környezet esetén a harmas oldalra ugrik, több környezet esetén a másodikra.
- Sorba rendezés oldalon bal oszlopból elem áthúzás a jobb oldalra – A bal oldalról elhúzott elem újra feltűnik. A jobb oldalhoz hozzáadódik.
- Sorba rendezés oldalon jobb oszlopból elem áthúzás a bal oldalra – Jobb oldalon eltűnik az áthúzott elem. Bal oldalon az elemek számra ilyenkor is változatlan.
- Elemzések megtekintése oldalon a gráfnál lévő nyíllra nyomás – Átugrik egy másik gráfra.
- Fájl oldalon egy fájlnál lévő másolás gombra nyomni – A felület jelzi, hogy a másolás sikeres, a fájl tartalma ténylegesen a vágólapra kerül.
- Fájl oldalon egy fájlnál lévő lejátzás gombra nyomni – A gráf nézet oldalra ugrik az oldal, ahol a vezérlésfolyamgráf egyből megjelenik.

3.4. További fejlesztések

3.4.1. P4 fordítóprogram beépítése

A vizsgált nyelv kibővítésének és a megfelelő hibajelzéseknek előfeltétele, hogy az elemzendő kód helyességét a P4 fordító segítségével ellenőrizzük. Mivel a P4 fordító egyetlen hivatalosan támogatott operációs rendszere az *Ubuntu 16.04* és az alkalmazás jelenleg Azure-ra van kitelepítve, ezért ennek beépítésére jelenleg két módszert látok megvalósíthatónak.

Egész projekt Ubuntu-on történő futtatása

Az első megoldás, hogy létrehozunk egy Ubuntu operációs rendszerrel rendelkező virtuális gépet, amelyre telepítjük a P4 fordítóját és az elemzett programot lefuttatjuk rajta. A futtatást C#-ból meg lehet tenni parancssori parancsok futtatásával. Ezt a megoldást nem ajánlom, mivel így fejlesztői környezetben is ki kell alakítani egy virtuális gépet, ahol tesztelhetők a módosítások.

Szerver a P4 fordítóprogramhoz

A második megoldás, hasonló az elsőhöz, de itt csak egy szerver applikációt telepítünk a virtuális gépre, aminek egy belépési pontja van, amit meg lehet hívni a futtatandó kóddal és visszatér a program helyességével. Ehhez a megoldáshoz is kell fejlesztői környezetet kialakítani, de itt csak egy applikációt kell tesztelni rajta és a jövőben kevés változtatást kell eszközölni rajta, így az új fejlesztőknek nem feltétlenül van szükség a környezet kialakítására. Így én ezt a megoldást javaslom.

3.4.2. Vizsgált résznyelv kiterjesztése

A vizsgálat során kikötésekkel éltem, hogy milyen programkódot tud feldolgozni az elemző programok, ezen kikötések feloldása lenne a feladat. A következőkkel egészíthető ki a résznyelv:

- Parser és Deparser feldolgozása – A fejlécek inicializálása és tovább küldése ne a felhasználó beállítása szerint menjen, hanem a kettő függvény alapján.
- Apply függvényen belül Else if lekezelése
- Action-ön belül elágazás lekezelése.

3.4.3. Felhasználó azonosítása

A felhasználó regisztrálhasson vagy belépjen meglévő fiókjába. A felhasználóbarát megközelítés miatt legyen lehetőség *Google*, *Facebook* vagy *Github* felhasználóval belépni. A felhasználó azonosítása nagyon sok lehetőséget von maga után. Lehetőség nyílik, hogy a felhasználó feltöltsön kódot és eltárolja saját maga, megjelölt másik felhasználók, vagy az egész közösség számára. Legyen lehetőség egymás kódjaira reagálni, ötleteket, megjegyzéseket adni, vagy kedvencnek jelölni, ezzel később is gyorsan megtalálható. A fájl kereső oldalon így bővíthetők a keresési funkciók felhasználó és kedveltségi szint alapján.

A már feltöltött kód módosítását és előzmények megtekintését is meg lehet csinálni, valamint elmenteni a kód elemzéseit, hogy a felhasználó nyomon tudja követni, hogy mennyit fejlődött kódjának optimalizáltsága. Amennyiben a programba beépítésre kerül az [5.1](#)-es pontban említett P4 fordítóprogram, akkor a felhasználó tudja fordítani programját, ezzel ellenőrizve helyességét. Ezen funkciók segítségével, akár egy fejlesztői környezet kialakítása is eszközölhető lenne.

Projektek létrehozása, ahol egy vagy több felhasználó képes módosítani a kódot.

3.4.4. Gráf elemzés összesített verziója

Az elemzés megtekintésénél jelenleg minden program lefutásnak, amit a különböző kezdeti értékek segítségével ad meg a felhasználó külön-külön kirajzoljuk a gráfot, de nincs egy olyan összesítő gráf, amely bemutatná a leggyakrabban használt kódrészleteket. Ennek megalkotására az adatok már rendelkezésre állnak, a különböző gráfokban szerepel, hogy egy rész hányszor lett használva, ezeket összesíteni kell és ezek alapján beállítani a csúcsok színezését.

3.4.5. Felület optimalizálás

Sötét téma

A felületen sötét téma került kialakításra, amely megfelelő kontrasztot és láthatóságot nyújt egy eset kivételével. A gráf éleinek színe alapértelmezetten fekete, ez a szín sötét téma esetén a sötét szürkén látható, de a szemnek fárasztó fókuszálni rá, ezért ezt módosítani kell. A gráfot teljesen újra kell építeni egy másik a sötét szürkével kontrasztban álló élszínezéssel.

Mobil használat

A felület jól használható mobilos környezetben, de vannak szépítendő és a felhasználói élményt javító változtatási lehetőségek.

Mobil optimalizáció:

- Az oldalsáv menü megjelenítése elhúzással. Ezzel könnyítve a felhasználót a navigációs menü megnyitásában, hogy ne kelljen mindig a megnyitó gombot lenyomni. Fejlesztéskor figyelni kell, hogy az animációs gombot használjuk.
- Gráf megjelenítő oldalon a tab-os elrendezésben elhúzással (swipe) is tudjon oldalt váltani.
- A megjelenített gráfok mozgásának és nagyításának megvalósítása. A felhasználó a gráfra nyomással nyisson meg egy felugró ablakot, ahol ezek a funkciók aktívak. Elegendő lenne csak a nagyítás funkció aktiválása, azonban a használt gráf ábrázoló könyvtár ezt a megoldást nem támogatja.
- A gráf valamiért nem a képernyő közepén jelenik meg és az újra méretezhetőség sem működik rajta teljes értékűen. Ezen esetekre nem tudtam megoldási ötletet találni.

4. Összefoglalás

A P4 új, napjainkban használt hálózati csomagokat kezelő programozási nyelv.

A szakdolgozatom célja az egy P4 programozási nyelv egy résznyelvében megírt programok optimalizálásának elősegítése egy gráfalapú elemzés segítségével.

Az elemzésem során a P4 programból kinyert csomagleíró fejleceket, és a hozzájuk tartozó változókat vizsgálom, hogy a program ingress kontrollfüggvényében szereplő adatmódosítási részben lévő utasításokban hányszor változott az értékük. A fejlecek, és ebből adódóan a változók kezdeti inicializáltságát a felhasználó adja meg az elemzést megelőzően.

A program elemző részét C# nyelven, Visual Studio keretrendszerben írtam meg. Miután a programból kiszedtük a fontos információkat, az elemzés következik, amelyet vezérlésifolyam- és adatfolyamgráfok segítségével oldottam meg.

A vezérlésifolyam gráf a program lefolyásának a reprezentációjára szolgál, míg az adatfolyam gráf a programban lévő változó értékek felhasználását mutatja be.

A vezérlésifolyamgráf a programállományból, az adatfolyam gráf a vezérlésifolyam gráfból kerül előállításra. Maga az elemzés a vezérlésifolyam gráfon halad végig, úgy, hogy közben az adatfolyam gráf információit használja fel.

A szakdolgozat felhasználói felülete egy webes alkalmazás, amely Angular keretrendszerben készült el. Ez az elemzéshez fontos funkciókon túl további kényelmi funkciókat is tartalmaz.

A felületen lehetőség van a P4 program feltöltésére, valamint kézzel való bevitelére, a gráfokon való interakciókra, kezdő- és végállapotok megadására, ezekből csomagok összeállítására és elemzett adatok megtekintésére.

5. Irodalomjegyzék

-
- [1] „P4 hivatalos oldala” [Online]. Available: <https://p4.org/> [Hozzáférés dátuma: 2020. 05. 31.]
- [2] „Xaml alkalmazása a WPF-ben” [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/desktop-wpf/fundamentals/xaml> [Hozzáférés dátuma: 2020. 05. 31.]
- [3] „Razor Pages” [Online]. Available: <https://docs.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-3.1&tabs=visual-studio> [Hozzáférés dátuma: 2020. 05. 31.]
- [4] „Angular hivatalos oldala” [Online]. Available: <https://angular.io/> [Hozzáférés dátuma: 2020. 05. 31.]
- [5] „Node.js hivatalos oldala” [Online]. Available: <https://nodejs.org/> [Hozzáférés dátuma: 2020. 05. 31.]
- [6] „Visual Studio letöltése” [Online]. Available: <https://visualstudio.microsoft.com/vs/> [Hozzáférés dátuma: 2020. 05. 31.]
- [7] „Angular fordító” [Online]. Available: <https://cli.angular.io/> [Hozzáférés dátuma: 2020. 05. 31.]
- [8] „SHA256 algoritmus” [Online]. Available: <https://en.wikipedia.org/wiki/SHA-2> [Hozzáférés dátuma: 2020. 05. 31.]
- [9] „D3” [Online]. Available: <https://d3js.org/> [Hozzáférés dátuma: 2020. 05. 31.]
- [10] „d3-graphviz” [Online]. Available: <https://github.com/magjac/d3-graphviz> [Hozzáférés dátuma: 2020. 05. 31.]
- [11] „Graphviz hivatalos oldala” [Online]. Available: <https://www.graphviz.org/> [Hozzáférés dátuma: 2020. 05. 31.]
- [12] „ng2-charts hivatalos oldala” [Online]. Available: <https://valor-software.com/ng2-charts/> [Hozzáférés dátuma: 2020. 05. 31.]
- [13] „ngx-store letölthetősége” [Online]. Available: <https://www.npmjs.com/package/ngx-store> [Hozzáférés dátuma: 2020. 05. 31.]
- [14] „Ásványi Tibor, algoritmusok és adatszerkezet II jegyzet” [Online]. Available: <http://aszt.inf.elte.hu/~asvanyi/ad/ad2jegyzet.pdf> [Hozzáférés dátuma: 2020. 05. 31.]