Szegedi Tudományegyetem

Informatikai Intézet

SZAKDOLGOZAT

Révész Gergő László

2022

Szegedi Tudományegyetem

Informatikai Intézet

Általános feladatütemező szolgáltatás

Szakdolgozat

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Készítette: |  | Témavezető: |  |
|  | Révész Gergő László |  | Pénzes János |  |
|  | programtervező informatikus szakos hallgató |  | asd |  |

Szeged

2022

# Feladatkiírás

A szakdolgozat keretein belül a hallgató egy általános feladat ütemező készít, amely segítségével időszakosan végrehajtandó funkciók futtathatók, előre definiált időközönként.

A hallgató szakdolgozatát a GriffSoft Informatikai Zrt.-nél készíti. A cég fő terméke a Forrás ügyviteli rendszer, amelyet főként Önkormányzatok, Költségvetési intézmények és középvállalatok használnak.

Az ügyviteli rendszer használata során többször felmerül az igény, hogy szükség van adott időközönként rendszer- vagy egyéb ügyviteli funkció futtatására. Ilyen lehet egy e-mail küldés vagy akár a NAV számára küldendő Online számla bevallás.

Ennek megoldására készít a hallgató egy központi ütemező alkalmazást, amelybe ezeket az időszakos futást igénylő funkciókat könnyen beilleszthetjük, egyedileg kezelhetjük, paraméterezhetjük.

Az ütemezőben definiált feladatok végrehajtásának figyelésére, egy böngésző alapú alkalmazás is fejlesztésre kerül, ami által egyszerűen nyomon követhetjük az aktív feladatok futási státuszát.

# Tartalmi összefoglaló

* ***A téma megnevezése:***

*Általános feladatütemező szolgáltatás*

* ***A megadott feladat megfogalmazása:***

*A feladat egy olyan alkalmazás készítése, amely dinamikusan, futásidőben tud Windows szolgáltatáshoz funkciókat hozzáadni és törölni, valamint egy felhasználói felület készítése ezen funkciók állapotainak megfigyelésére.*

* ***A megoldási mód:***

*Egy, a keretrendszer által nyújtott technológia, a Reflection segítségével a program futásidőben tud változtatni saját szerkezetén, és ezt kihasználva készítettem egy újra indítást nem igénylő, de szerkezetileg folyamatosan változó alkalmazást. Felhasználói felületnek egy web alkalmazást készítettem, ami kommunikál a háttérben futó szolgáltatással.*

* ***Alkalmazott eszközök, módszerek:***

*Az alkalmazást a Visual Studio nevű fejlesztői környezetben készítettem el, C# nyelven.*

*A fejlesztés Windows szolgáltatás részéhez .NET Core, a felhasználói felület részéhez a Blazor keretrendszert használtam. Az alkalmazást több eszközön, különböző verziószámokkal teszteltem.*

* ***Elért eredmények:***

*Az alkalmazást a Griffsoft Informatikai Zrt használja.*

* ***Kulcsszavak:***

***//TODO csak 4-5 kell***

*számítógép, program, assembly, futtatás, aszinkron, appDomain*

Tartalom

[Feladatkiírás 2](#_Toc99401624)

[Tartalmi összefoglaló 3](#_Toc99401625)

[Bevezetés 5](#_Toc99401626)

[1 Használt technológiák, eszközök bemutatása 6](#_Toc99401627)

[1.1 Programozási nyelv 6](#_Toc99401628)

[1.1.1 C# 7](#_Toc99401629)

[1.2 Keretrendszer 7](#_Toc99401630)

[1.2.1 .NET 8](#_Toc99401631)

[1.2.2 Blazor 8](#_Toc99401632)

[1.3 Fejlesztői környezet 9](#_Toc99401633)

[1.3.1 Visual Studio 9](#_Toc99401634)

[2 Megvalósítás menete 10](#_Toc99401635)

[2.1 Windows Service 10](#_Toc99401636)

[2.1.1 A Task objektum, az async és az await kulcszó 11](#_Toc99401637)

[2.1.2 FileSystemWatcher 12](#_Toc99401638)

[2.1.3 Reflection 12](#_Toc99401639)

[2.1.4. AssemblyLoadContext 13](#_Toc99401640)

[2.1.5. NLog 13](#_Toc99401641)

[2.2. Blazor 13](#_Toc99401642)

[2.2.1. Service Controller 13](#_Toc99401643)

[2.2.2. TCP Listener és TCP Client 13](#_Toc99401644)

[3. A kész alkalmazás 13](#_Toc99401645)

[4. Funkciók 13](#_Toc99401646)

[4.1. Service-ben 13](#_Toc99401647)

[4.2. UI-on 13](#_Toc99401648)

[5. Tesztelés 13](#_Toc99401649)

[6. Elért eredmények 14](#_Toc99401650)

[6.1. A kész alkalmazás 14](#_Toc99401651)

[6.2. További fejlesztési lehetőségek 14](#_Toc99401652)

[Irodalomjegyzék 15](#_Toc99401653)

[Nyilatkozat 16](#_Toc99401654)

# Bevezetés

A Griffsoft Informatikai Zrt. fő terméke a Forrás ügyviteli rendszer, amit Önkormányzatok, Költségvetési intézmények és középvállalatok is használnak adminisztrációs célokra. A Forrás használata közben számos alkalommal felmerül az igény bizonyos funkciók előre meghatározott időközönként történő lefuttatására. Az ilyen funkciók pl. egy e-mail küldés vagy a NAV számára küldendő Online számla bevallás.

Az ellátandó feladatok megnövekedett száma miatt a dolgozók munkáját megkönnyítené, ha ezeket egy számítógépes program végezné el automatikusan. Ezen probléma megoldása miatt esett a választásom egy olyan alkalmazás elkészítésére, ami ennek a feladatoknak eleget tesz.

A szakdolgozatom célja egy olyan Windows operációs rendszeren futó szolgáltatás implementálása, amit mindamellett, hogy menedzseli a szükséges funkciók futtatását, lehetőséget nyújt újabb feladatok hozzáadására és meglévők törlésére egyaránt anélkül, hogy meg kellene szakítani a program futását.

Az alkalmazás egy háttérben futó szolgáltatás, amihez egy Blazor webalkalmazást is készítek a felhasználói interakciók kezelésére.

A kész alkalmazás egy könnyen használható Windows szolgáltatás, aminek továbbfejlesztése kevés programozói tudást igényel. A dinamikus megvalósításnak köszönhetően bármennyi ideig képes futni nagyon kevés erőforrás használatával, és tudja kezelni a funkciók futtatása közben fellépő esetleges hibákat.

//TODO bővíteni

# 1 Használt technológiák, eszközök bemutatása

## 1.1 Programozási nyelv

**//TODOref**:https://www.researchgate.net/figure/A-brief-history-of-high-level-programming-languages-from-1956-to-2004\_fig1\_3248243

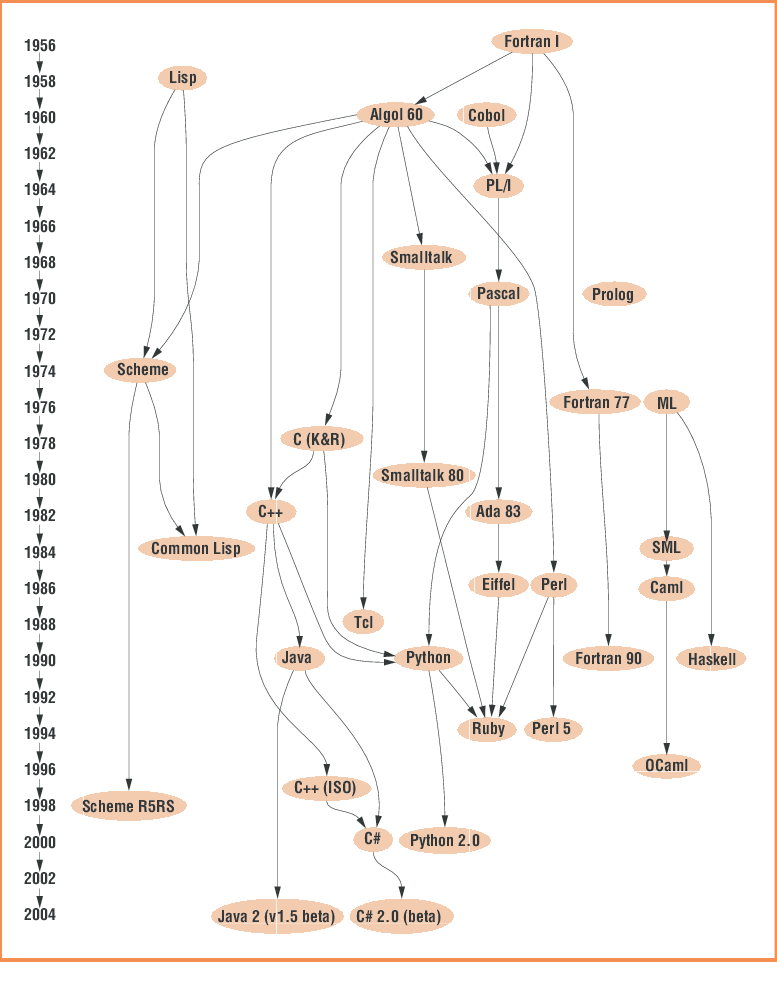
Egy mai számítógép az emberek számára követhetetlen számolási sebességgel bír, és ez miatt jó ötletnek tűnhet, ha az egyes emberi feladatokat rá tudnánk bízni egy számítógépre. Ezek „agya” egyelőre még nem ért emberi nyelven, de akkor mégis hogyan tudnánk megmondani egy számítógépnek, hogy mit csináljon?

Egy számítógép utasításokat hajt végre. Egy utasítás lehet például egy szorzás, ami szétbontható összeadások összegére. Egy összeadást is tovább lehet egyszerűsíteni több, de egyszerűbb műveletre, és ezt addig ismételni, amíg annyira elemeire bontott utasítást nem kapunk, hogy azt egymás után írt egyesekkel és nullákkal (bitekkel) reprezentálni lehet. Az így kapott bitek sorozata egy számítógép számára már értelmezhető, de az ember számára szinte érthetetlenné váló utasítás. Már a legegyszerűbb matematikai műveleteket is nagyon hosszadalmas és macerás munka ilyen kis elemi részekre bontani annak érdekében, hogy elvégeztethessük egy számítógéppel, nem beszélve a komplexebb programokról, amik az előbb említett bonyolultabb utasítások egymás utáni sorozatából állnak.

Ennek a problémának a megoldására születtek meg a programozási nyelvek. A programozási nyelvek mindegyike, csak úgy, mint egy emberi nyelv, rendelkezik saját szabályrendszerrel. Egy szabályrendszer lehetőséget ad az emberek számára egységes módon megfogalmazni egy utasítást, vagy annak sorozatát, egy programot. Egy program a nyelv szabályait követve mindamellett, hogy az más, a szabályokat szintén ismerő ember számára is olvashatóvá és érthetővé válik, a programozási nyelv a programnak el tudja készíteni a számítógép számára érthető megfelelőjét. Ezt a folyamatot, amikor egy ember által olvasható programból egy gép által olvasható, bitekből álló gépi kód készül, fordításnak nevezik. A fordítást a programozási nyelv saját fordítója csinálja, ami csakugyan egy program, ezért természetesen a legelső ilyen fordítót programozási nyelv hiányában nehéz volt „megírni”.

### 1.1.1 C#

Attól függetlenül, hogy mindegyik végeredménye egy gép számára érthető és végrehajtható utasítás sorozat, a ma létező programozási nyelvek között érdemes lehet választani az alapján, hogy milyen célra szeretnénk programot készíteni, ugyanis ettől függően változhat a program megírásának nehézsége és minősége is. Én a szakdolgozatomat a Microsoft által fejlesztett C# programozási nyelven készítettem el.



1.1. ábra: magas szintű programozási nyelvek története

## 1.2 Keretrendszer

//TODO ref: <https://hackr.io/blog/what-is-frameworks>

//TODO ref: https://hu.wikipedia.org/wiki/Szoftverkeretrendszer

Egy nagy projekt megírása sok profi programozónak is rengeteg időbe telik, azonban léteznek módszerek, amikkel a fejlesztés ideje lerövidíthető.

A programozók valamilyen hálózaton, például az interneten keresztül meg tudják osztani egymással elkészített munkáikat, ily módon ezt egy másik programozó már fel tudja használni programjában. Jó példa erre a majdnem minden programozási nyelvben megtalálható Math könyvtár, ami matematikai függvényeket tartalmaz. Ennek köszönhetően, ha egy programozó programjának fejlesztése során például hatványozni szeretne, nem kell a nulláról leprogramoznia egy hatványozás függvényt, használhatja a mások által megírt Math könyvtárat.

A könyvtáraknál még nagyobb segítséget tudnak nyújtani a keretrendszerek. Ezek szintén programozók által készített, de a könyvtáraknál jóval bővebb rendszerek, amik sok más programozást segítő eszközt is tartalmaznak, ilyenek például az API-k (//TODO RÉSZLETEZZEM??). A keretrendszerek általános eszköztárat biztosítanak, amik segítségével a fejlesztőnek nincs szükség mindent az elejéről felépíteni, helyette a lényegi részre, a funkciók implementálására fókuszálhat, nagyban lerövidítve ezzel a projekt fejlesztésének idejét. Egy web alkalmazás fejlesztése esetében például nem kell azzal foglalkozni, hogyan tudjuk megjeleníteni azt egy böngésző ablakban, helyette az időnket a lényegi részre, az alkalmazás funkcióinak fejlesztésére fordíthatjuk.

### 1.2.1 .NET

A szakdolgozatom elkészítéséhez a .NET Core keretrendszert használtam. Ez egy ingyenes, nyílt forráskódú szoftver, ami több operációs rendszerre is képes lefordítani és futtatni egy C#-ban (F# és Basic) megírt forráskódot, továbbá rengetek hasznos API és NuGet csomag használatára is lehetőséget ad.

### 1.2.2 Blazor

Nagy különbség a kettő között, hogy könyvtárak esetében a programozó munkája lesz kibővítve máséval, a keretrendszerek viszont a programozó munkájával kiegészítve építenek fel egy működő programot.

A keretrendszerek használatának elsajátítása nagyon meggyorsíthatja egy program fejlesztését, szükségtelen használata viszont csak ront a program performanciáján. A szakdolgozatomhoz a C# programozási nyelv keretrendszerét, a .NET keretrendszert használtam, ami a nagy eszköztárának köszönhetően meggyorsította munkámat.

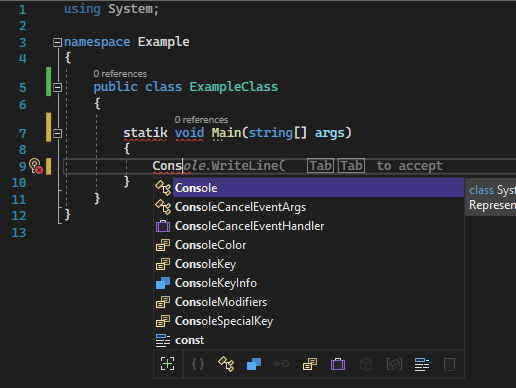
## 1.3 Fejlesztői környezet

**//TODO ref**: https://ak-akademia.hu/fejlesztoi-kornyezetek/

Programot írni bárhova tudunk, ahol lehet betűket tárolni, beleértve egy papírlapot, de még egy barlang falát is. Ahhoz viszont, hogy elérjük, hogy ezeket az utasításokat egy számítógép hajtsa végre, először is kell a programozási nyelvnek előbb említett fordítójára, másodszor pedig egyes programok esetében még szükségünk lehet egy futtatókörnyezetre is.

### 1.3.1 Visual Studio

A fejlesztői környezetek olyan programozók számára készült szoftverek, amik a fordítót, a futtatókörnyezetet és még egy szövegszerkesztőt is tartalmaznak annak érdekében, hogy a programozónak a rendelkezésére álljon minden egy helyen. Egy ilyen fejlesztői környezet nagyon sokat tud gyorsítani egy program fejlesztésén. Léteznek azonban olyan fejlesztői környezetek is, amik mindamellett, hogy tudják ezek mindegyikét, még adott programozási nyelvek formai szabályait is „ismerik”, ami miatt még több segítséget tudnak nyújtani. Képesek például a programkód gépelése közben vétett hibákat jelezni a fejlesztő számára, felugró listával segíteni befejezni a nyelv kulcs szavait, képes velük a fejlesztő a program futása közben belelátni a számítógép memóriájának bizonyos részeibe (debugolás), és még ezeken kívül is rengetek hasznos funkcióra tehet szert egy fejlesztő a használatukkal. Ezeket a fejlesztői környezeteket integrált fejlesztői környezeteknek hívják, és manapság már a leggyakrabban használt programozási nyelvek mindegyikéhez létezik ilyen.



1.2. ábra: a Visual Studio egy hibajelzése és egy kiegészítő javaslata

A fejlesztői környezet tehát egy olyan szoftver, ami a programozóknak hatalmas segítséget nyújt a programok fejlesztésében, javításában. A szakdolgozatom programkódját a C# programozási nyelv egy szintén Microsoft által készített integrált fejlesztői környezetét, a Visual Studio-t használva készítettem el.

# 2 Megvalósítás menete

## 2.1 Windows Service

Egy hagyományos alkalmazás általában rendelkezik felhasználói felülettel annak érdekében, hogy a felhasználók interakcióba tudjanak lépni az alkalmazással, kommunikálni tudjanak vele és közölni tudják szándékukat a programmal. Jó példa erre egy számítógépes játék, aminek elindulásakor megjelenik az ikonja a tálcán, a bejelentkezett felhasználónak tudomására van hozva, hogy fut-e az alkalmazás, amit elindított, és tud-e játszani vele.

Egy Windows Service alkalmazás ezekből a szempontokból különbözik egy hagyományos alkalmazástól. Ilyen alkalmazásokat általában azért készítünk, hogy bizonyos funkciókat kezeljenek hosszú időn keresztül. Ezeknek a funkcióknak nincs szükségük felhasználói interakcióra, csak futniuk kell, és tenniük a dolgukat, emiatt alapértelmezetten nincs is hozzájuk felület, és a felhasználó sok esetben nem is tud futásukról.

Ilyen alkalmazást a Visual Studio nevű fejlesztői környezetben könnyedén meg tudunk írni. Egy Worker Service típusú projektet kell létrehozni, néhány sor programkód módosítással és pár beállítással, paraméterrel el is tudunk készíttetni egy futtatható .exe kiterjesztésű alap Service alkalmazást //TODO LINK, amit a Windows szolgáltatásai közé beregisztrálva futtatni tudunk. Debug-olására csak úgy, mint fejlesztése közben, kiadás után is van lehetőség egy debugger hozzácsatolásával.

Egy Service alkalmazásból egy számítógépen maximum egy példány fut még akkor is, ha felhasználó váltás történik. Mivel az alkalmazás nem egy bejelentkezett felhasználóhoz, hanem a számítógéphez van kötve, így általában boot-olás után már el is indulnak. Szüneteltetésükre és megállításukra is van lehetőség, de csak megfelelő felhasználói jogosultsággal.

### 2.1.1 A Task objektum, az async és az await kulcszó

Egy Task objektum egy műveletet reprezentál, ami általában aszinkron módon felsorakozik a thread pool-ra, majd sorra kerülés után lefut egy szálon. Hasznos lehet, ha olyan műveletet szeretnénk végrehajtani, ami sok időbe telik, várakoztatja és blokkolja a fő szálat

Az await kulcsszó segítségével tudjuk megvárni az előbb említett Task objektum sorra kerülését és lefutását. Hasznos lehet, ha például a felhasználó várakozásának ideje alatt nem a lefagyott, hanem például egy töltő képernyőt szeretnénk megjeleníteni. Az e féle várakozás ezt lehetővé teszi, hiszen a fő szálat tehermentesíti az elvégzendő feladat alól, így az használható marad másra.

Az async kulcsszó teszi lehetővé, hogy az ezzel megjelölt metódusok belsejében használható legyen az await kulcszó, ezért az async és await kulcsszavak általában együtt használatosak.

A szakdolgozatomban cél, hogy a ciklust, ami stopperóraként funkcionál és 1 másodpercenként futtatja a dll-eket, ne blokkolják a hozzáadás és törlés metódusok, hiszen akkor a másodperc számláló lassulna, nem lenne pontos.

//TODO KEP A TASKOKROL, TASK FUGGVENYEKROL

### 2.1.2 FileSystemWatcher

A .NET Core System.IO névterében található FileSystemWatcher egy keretrendszer által biztosított osztály, ami képes egy könyvtárban, és annak alkönyvtáraiban észrevenni, ha egy megadott mintára illeszkedő névvel rendelkező file-lal változás történt, és lefuttatni az eseményhez hozzárendelt függvényt.

A szakdolgozatomban ezt az osztályt használtam arra, hogy futás közben file-ok hozzáadásakor vagy törlésekor el legyenek indítva a megfelelő eseménykezelők.

Az osztály konstruktorában példányosítok egy FileSystemWatcher objektumot, aminek első paraméternek a monitorozott mappa abszolút útvonalát adom át, amit egy publikus readonly adattagból szedek, második paraméternek pedig egy beinjektált objektum egyik property-jét.

Ez az objektum tartalmazza az összes paramétert, amik a Service alkalmazás exe file-jának mappájában található appsettings.json file-ból bind-olódnak be. //TODO KÉSŐBB A BLAZORNÉL. Az EnableRaisingEvents property-jének igazra állításával tudnak meghívódni az eseménykezelők, //TODO a notifyfiltert kiszedni; a többi porperty-t pedig alapértelmezetten hagytam. Ez után a hozzáadás és a törlés delegate-ekhez hozzáadom az eseménykezelőket. Ezek aszinkron metódusok, amiknek void a visszatérési értéke. Ez azért fontos, mert ilyen esetben nincs lehetőség arra, hogy az eseménykezelőből eldobott hibát elkapjuk.

### 2.1.3 Reflection

A Reflection egy keretrendszer által nyújtott technológia, ami segítségével dinamikusan tudunk assembly-ket és típusokat kezelni. Segítségével ezekről futás idejű információt tudunk szerezni, példányosítani tudjuk őket, az így létrehozott objektumnak meg tudjuk hívni a metódusait, és elérjük az adattagjait is. Ezt kihasználva tudok egy teljes mértékben ismeretlen dll-ről strukturális ellenőrzéseket végezni, majd ezek alapján megállapítani, hogy megfelel-e a követelményeknek, példányosítható és futtatható-e.

//TODO KÉP, EGYETLEN OSZTÁLY ÉS INTERFACE, ELLENŐRZÉS, SIKERESSÉG ELLENŐRZÉS,

### 2.1.4. AssemblyLoadContext

.NET Core-ban megszűnt a lehetőség egy folyamaton belül több AppDomain létrehozására, helyette bevezették az AssemblyLoadContext-et. Ez az osztály használható arra, hogy assembly-ket töltsünk be, amik el lesznek tárolva egy összefüggő kollekcióban. Előnye az Assembly osztály statikus Load metódusával szemben az, hogy ha el van engedve az összes referencia az AssemblyLoadContext objektumban tárolt assembly-k mindegyikéről, akkor az Unload metódus meghívásával teljesen ki is törlődnek az assembly-k az AppDomain-ből.

//TODO KÉP, BIZONYOS SOROKRŐL, MEGMAGYARÁZVA, DLLLIFTER

### 2.1.5. NLog

## 2.2. Blazor

### 2.2.1. Service Controller

### 2.2.2. TCP Listener és TCP Client

A TCP Listener osztály segítségével példányosítani tudunk egy TCP kapcsolat fogadására alkalmas objektumot. A webes felület ennek segítségével kommunikál a Service alkalmazással,

# 3. A kész alkalmazás

# 4. Funkciók

## 4.1. Service-ben

## 4.2. UI-on

# 5. Tesztelés

# 6. Elért eredmények

## 6.1. A kész alkalmazás

## 6.2. További fejlesztési lehetőségek

# Irodalomjegyzék

# Nyilatkozat

Alulírott Révész Gergő László, Programtervező informatikus szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, Programtervező informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

Szeged, 2022. január 17.

aláírás