**Fejlesztői dokumentáció**

## 0. Program Futtatása

A projekt futtatásához az alábbi lépéseket kell követni:

### 0.1 Szükséges szoftverek telepítése

A projekt futtatásához az alábbi programok telepítése szükséges:

* **Visual Studio 2022** (ajánlott a legfrissebb verzió)
* **.NET 8 SDK** (ha Visual Studio telepítés közben nem lett kiválasztva, külön is telepíthető a [hivatalos weboldalról](https://dotnet.microsoft.com/en-us/download))

### 0.2 A projekt megnyitása és futtatása

1. **A forráskód letöltése vagy klónozása**  
   Ha a projekt egy GitHub repóban található, akkor klónozható a következő paranccsal a parancssorban (Git Bash, PowerShell vagy Command Prompt):

git clone https://github.com/mskvszkyt/dusza-cluster.git

Ha nincs Git telepítve, a repó ZIP formátumban is letölthető, majd ki kell csomagolni.

1. **A projekt megnyitása**
   * Nyisd meg a **Visual Studio 2022**-t.
   * Kattints a **"Nyiss meg egy projektet vagy megoldást"** (Open a project or solution) opcióra.
   * A megnyíló fájlkezelőben keresd meg a projekt mappáját, majd válaszd ki a **.sln** kiterjesztésű fájlt.
   * Kattints a **"Megnyitás"** gombra.
2. **A projekt futtatása**
   * A Visual Studio-ban győződj meg arról, hogy a **Debug** konfiguráció van kiválasztva.
   * A felső menüsávban válaszd ki a **"Start"** (Zöld nyíl gomb) opciót, vagy nyomd meg az **F5** billentyűt.
   * Ha minden megfelelően van beállítva, a program elindul, és megnyílik az alkalmazás fő ablaka.

### 3.3 Lehetséges hibák és megoldásuk

* **"A .NET 8 nem található" hibaüzenet:**  
  Győződj meg róla, hogy a .NET 8 SDK telepítve van. Ha nincs, töltsd le és telepítsd a [hivatalos oldalról](https://dotnet.microsoft.com/en-us/download).
* **"Függőségek hiányoznak" hibaüzenet:**  
  A projekt első megnyitása után érdemes frissíteni a függőségeket. Ehhez nyisd meg a **Package Manager Console**-t a Visual Studio-ban, és futtasd az alábbi parancsot:

dotnet restore

* **"A Visual Studio nem található a rendszerben" hibaüzenet:**  
  Ellenőrizd, hogy a Visual Studio 2022 telepítve van-e. Ha nincs, töltsd le és telepítsd a [hivatalos oldalról](https://visualstudio.microsoft.com/hu/downloads/).

## 1. Fejlesztés menete

### 1.1 Célkitűzés

A projekt célja egy WPF alkalmazás fejlesztése volt, amely klasztereket és azokhoz tartozó erőforrásokat kezeli. A fejlesztés során a fő szempontok a stabilitás, az átláthatóság és a hatékony erőforráskezelés voltak.

### 1.2 Fejlesztési fázisok

1. **Tervezés**: A rendszer alapvető funkcionális és nem-funkcionális követelményeinek meghatározása.
2. **Implementáció**: Az alkalmazás megvalósítása, kódolás, funkcionális egységek kifejlesztése.
3. **Tesztelés**: Az alkalmazás működésének ellenőrzése, hibajavítás.
4. **Telepítés és élesítés**: Használatba helyezés, hibajavítás.

**2. Használt technológiák**

### 2.1 Fejlesztési környezet

A projekt fejlesztéséhez a **C#** nyelvet és a Windows Presentation Foundation (**WPF**) technológiát használtuk, amely lehetővé teszi az alkalmazás modern, dinamikus felhasználói felületének kialakítását. A fejlesztés a **.NET 8** keretrendszerben zajlott, amely biztosítja a stabilitást, a teljesítőképességet és a legfrissebb technológiai támogatást. A **Visual Studio 2022** szolgált a fejlesztés fő eszközeként, amely fejlett hibakeresési és kódszerkesztési funkcióival tette hatékonnyá a munkafolyamatot.

A projekt egyik fontos része a fájlkezelés, amelyhez a **System.IO** osztálykönyvtár eszközeit alkalmaztuk. Ez biztosítja a gyors és biztonságos adatbeolvasást, valamint a naplózási és egyéb rendszerfolyamatok hatékony kezelését.

### 2.2 Felhasznált csomagok

A fejlesztés során az alábbi **NuGet** csomagokat használtuk:

* **CommunityToolkit.MVVM (8.4.0)** – MVVM támogatás, amely egyszerűsíti a kód struktúráját és az adatkötést.
* **MahApps.Metro (2.4.10) –** Modern UI komponensek a felhasználóbarát felületek kialakításához.
* **LiveChartsCore (2.0.0-rc5.1) –** Interaktív adatok vizualizálásához.
* **ModernWpfUI.MahApps (0.9.5) –** Kiegészítés a WPF modernizálásához.
* **ModernWpfUI (0.9.6)** – Továbbfejlesztett WPF UI lehetőségek.
* **LiveChartsCore.SkiaSharpView (2.0.0-rc5.1)** – SkiaSharp alapú grafikonkezelés.
* **LiveChartsCore.SkiaSharpView.WPF (2.0.0-rc5.1)** – WPF grafikonkezelés fejlesztése.

Ezek a csomagok lehetővé tették a hatékony kódolást, a felhasználói felületek reszponzivitását és az adatok vizualizálását modern, windows stílusához hasonló megjelenésben.

### 2.3 Verziókezelés

A verziókezeléshez a **Git** rendszert használtuk, amely lehetővé tette a projekt változásainak pontos nyomon követését, valamint a csapaton belüli hatékony együttműködést. A projekt forráskódja **GitHub** repóban van tárolva, ahol a fejlesztési folyamat az alábbi branch-struktúrát követi:

* **main** – A stabil, kiadásra szánt verziót tartalmazza.
* **feature branchek** – Egyedi fejlesztések, amelyek pull request segítségével kerülnek be a fő ágba.

Ez a rendszer biztosítja, hogy a **main branch** mindig egy stabil, futtatható verziót tartalmazzon. A pull request-ek segítik a kód ellenőrzését, a hibák kiszűrését és a kódminőség fenntartását.

## 3. Program felépítése

### 3.1 Architektúra

A projekt egy hagyományos **WPF alkalmazás**, amely **nincs** elkülönített backend és frontend részre osztva. Az alkalmazás **nem** követi az MVVM mintát, hanem egy egyszerűbb struktúrát használ, ahol az adatok és a logika a UI kódban kezelhető.

### 3.2 Fő komponensek

* **Fő ablak**: Az alkalmazás kezdő ablaka, amely lehetőséget biztosít a klaszterek kezelésére.
* **Klaszterkezelés**: Az egyes klaszterek és azokhoz tartozó erőforrások vizuális megjelenítése.
* **Grafikonok**: Az erőforrás-használat elemzésére szolgáló diagramok.
* **Fájlkezelés**: Az adatok **System.IO** alapú mentése és betöltése. Ezen kívül egy FileManager osztály került alkalmazásra a kétirányú állapotlekövetés érdekében.

### 3.3 Models réteg felépítése

A program modelljei a klaszterek és azok összetevőinek logikai reprezentációját valósítják meg. Az alábbi osztályok alkotják a rendszer alapvető adatszerkezetét:

#### **1. Cluster osztály**

A klaszterek alapvető struktúráját és tulajdonságait írja le.

**Tulajdonságok:**

* Path: A klaszter fájlrendszerbeli elérési útja
* ScheduledPrograms: A klaszterben ütemezett programok listája (ScheduledProgram objektumok)
* Instances: A klaszterhez tartozó számítógépek listája (Instance objektumok)

**Feladata:**

* A klaszter teljes állapotának tárolása
* Programütemezések és számítógépek központi kezelése

#### **2. Instance osztály**

Egy számítógépet reprezentál a klaszteren belül.  
**Tulajdonságok:**

* Name: A számítógép egyedi neve
* MemoryCapacity: Teljes memóriakapacitás (GB)
* ProcessorCapacity: Teljes processzorkapacitás (%)
* Programs: A számítógépen futó programok listája (ProgInstance objektumok)

**Metódusok:**

* CalculateMemoryUsage(): Aktuális memóriahasználat számítása
* CalculateProcessorUsage(): Aktuális processzorhasználat számítása
* CanAccommodateProgram(): Ellenőrzi, hogy elfér-e egy új program

**Számított tulajdonságok:**

* MemoryUsagePercentage: Memóriakihasználtság százalékban
* ProcessorUsagePercentage: Processzorkihasználtság százalékban
* AvailableMemoryCapacity: Szabad memória
* AvailableProcessorCapacity: Szabad processzorkapacitás

#### **3. ProgInstance osztály**

Egy futó program példányt modellez.  
**Tulajdonságok:**

* ProgramName: A program egyedi azonosítója
* IsRunning: Aktív állapot jelzője
* ProcessorUsage: Processzorterhelés (%)
* MemoryUsage: Memóriahasználat (GB)
* StartDate: Indítás időpontja

**Feladata:**

* Programpéldányok erőforrásigényének nyomon követése
* Futó folyamatok állapotának tárolása

#### **4. ScheduledProgram osztály**

Ütemezett programok követelményeit definiálja.  
**Tulajdonságok:**

* ProgramName: Program azonosítója
* InstanceCount: Kötelezően futtatandó példányszám
* ProcessorRequirement: Processzorigény példányonként (%)
* MemoryRequirement: Memóriahasználat példányonként (GB)

**Feladata:**

* Klaszter-szintű erőforrásigények meghatározása
* Automatikus példánykezelés támogatása

### Modellek kapcsolata

#### Megjegyzés: A diagram szemlélteti a modellek hierarchiáját:

1. Egy Cluster több Instance-t (számítógépet) tartalmaz
2. Egy Instance több ProgInstance-t (programpéldányt) futtat
3. A ScheduledProgram a klaszter szintjén határozza meg a programkövetelményeket

A modellek a System.IO és a **Services/FileManager.cs** osztály segítségével szinkronban vannak a fizikai fájlrendszerrel, biztosítva az adatok konzisztenciáját.

## 4. Műveletek

### 4.1 Számítógépek kezelése

* **Hozzáadás**: Új számítógép felvétele egy klaszterbe
* **Eltávolítás**: Számítógép törlése
* **Módosítás**: Kapacitások vagy egyéb paraméterek frissítése

### 4.2 Programok kezelése

* **Futtatás**: Új program vagy meglévő programpéldány indítása egy számítógépen
* **Leállítás**: Futó program leállítása

### 4.3 További műveletek

### A saját ötleteink, amelyek hasznosak lehetnek klaszterek kezelése közben.

* **Egyesítés**: Két klaszter egyesítése.
* **Példány** **áthelyezése**: Program példányok átcsoportosítása klaszterek között.
* **Számítógépek** **áthelyezése**: Egyes gépek mozgatása különböző klaszterek között.

## 5. Projektmunka

A fejlesztés során a **GitHub Projects** eszközt használtuk a feladatok szervezésére és nyomon követésére. A projektmunka struktúrája lehetővé tette, hogy átlátható módon kezeljük a fejlesztési folyamatot, valamint hatékonyan osszuk fel a munkát a csapat tagjai között.

### 4.1 Feladatkezelés GitHub Projects segítségével

A fejlesztési folyamat során egy **Kanban táblát** hoztunk létre a GitHub Projects-ben, amelyen a következő oszlopokat használtuk:

* **To Do** – Az aktuálisan elvégzendő feladatok.
* **In Progress** – A folyamatban lévő fejlesztések.
* **Done** – A befejezett és az éles verzióba beolvasztott fejlesztések.

### 4.2 Issue-k és feladatok

A fejlesztés során minden új funkcióhoz vagy hibajavításhoz **külön GitHub Issue**-t hoztunk létre. Az issue-k tartalmazták:

* A feladat rövid leírását
* Az elvárt eredményt vagy funkciót
* Az esetleges problémákat vagy függőségeket
* A felelőst, aki a feladatot végzi

Minden issue egy fejlesztési ághoz (branch) kapcsolódott, amelyet a munka befejezése után **pull request** segítségével egyesítettünk a fő ággal.

### 4.3 Pull Request-ek és kódátvizsgálás

Minden új fejlesztés vagy módosítás pull request (PR) formájában került be a fő kódbázisba. A PR-k előnyei:

* Lehetőséget adtak a csapattagok számára, hogy átnézzék egymás kódját.
* Ellenőrzések futottak le a merge előtt.
* Kommenteket lehetett hagyni, ha javításokra volt szükség.

A fejlesztési folyamat során folyamatosan dokumentáltuk a pull request-eket, így könnyen visszakövethető volt, hogy melyik fejlesztés mikor és milyen formában került be az alkalmazásba.

### 4.4 Hatékonyság és csapatmunka

Ez a struktúra biztosította, hogy a fejlesztési folyamat jól szervezett, átlátható és hatékony legyen. Az issue-k és PR-k használata segített elkerülni az átfedéseket a feladatok között, valamint gyors és hatékony hibajavítást tett lehetővé.

A GitHub Projects, issue-k és pull request-ek kombinációja lehetővé tette, hogy a csapat minden tagja tisztában legyen az aktuális feladatokkal, a fejlesztés állapotával, és hatékonyan tudjunk együtt dolgozni a projekt sikeres megvalósításán.

