**Fejlesztői dokumentáció**

**1. Fejlesztés menete**

* 1. **Célkitűzés**

A projekt elsődleges célja a versenyfeladat hibátlan megoldása volt. Viszont a második fordulóban lehetőségünk volt saját képzeleteinket megvalósítani a jelenlegi szakmai tudásunkkal.

* A fejlesztés során figyelembe vett szempontok.

**1.2 Fejlesztési fázisok**

1. **Tervezés**: Követelmények meghatározása, rendszertervezés.
2. **Implementáció**: Kódolás, modulok elkészítése.
3. **Tesztelés**: Rendszer szintű tesztelések.
4. **Telepítés és élesítés**: Használatba helyezés, hibajavítás.

**2. Használt technológiák**

### Fejlesztési környezet

A fejlesztéshez **C# WPF** technológiát használtunk, a projekt fejlesztése **.NET 8** verzióban történt. A fejlesztői eszközünk **Visual Studio 2022** volt.

A projekt során a fájlkezelés kiemelt szerepet kapott, amelyhez a **System.IO könyvtár** eszközeit alkalmaztuk.

### Felhasznált csomagok

A fejlesztés során az alábbi NuGet csomagokat használtuk:

* **CommunityToolkit.MVVM** (8.4.0) – MVVM támogatás
* **MahApps.Metro** (2.4.10) – Modern UI komponensek
* **LiveChartsCore** (2.0.0-rc5.1) – Adatok vizualizálása
* **ModernWpfUI.MahApps** (0.9.5) – WPF modernizációs eszközök
* **ModernWpfUI** (0.9.6) – WPF UI fejlesztéshez
* **LiveChartsCore.SkiaSharpView** (2.0.0-rc5.1) – Grafikonok megjelenítésére
* **LiveChartsCore.SkiaSharpView.WPF** (2.0.0-rc5.1) – WPF grafikonkezelés

## Verziókezelés

A verziókezeléshez **Git**-et használunk. A projekt forráskódja **GitHub** repóban található, ahol több branch-et hoztunk létre. Minden bővítést egy új branch-en fejlesztettünk le először. Miután egy fejlesztési egység elkészült, pull request segítségével kerül be a **main** branch-be.

### A GitHub repo branch-ei

A fejlesztés során az alábbi fő branch-eket használjuk:

* **main** – A stabil, éles verziót tartalmazza.
* **feature branchek** – Egyéni fejlesztések, amelyeket pull request segítségével merge-elünk

Ez a folyamat biztosítja, hogy a **main** branch mindig egy futtatható, stabil verziót tartalmazzon. A pull request-ek lehetőséget adnak az ellenőrzésre, valamint segítenek a kódminőség fenntartásában és a hibák kiszűrésében.

### Korábbi pull request-ek

A pull request-ek dokumentálják a fejlesztési lépéseket, lehetőséget biztosítva a kódátvizsgálásra és a hibák kiszűrésére. A változtatások és a kommentek segítenek a fejlesztők közötti hatékony együttműködésben.

**3. Program felépítése**

**3.1 Architektúra**

* **Model-View-ViewModel (MVVM) minta**
* Backend REST API központú kommunikáció

**3.2 Fő komponensek**

* **Backend:**
  + Hitelesítés, felhasználói kezelés
  + Adatbázis kezelés
  + API végpontok
* **Frontend:**
  + UI komponensek
  + Adatok megjelenítése és interaktivitás

### 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 segítségével szinkronban vannak a fizikai fájlrendszerrel, biztosítva az adatok konzisztenciáját.

**4. Klaszter műveletek**

**4.1 Klaszterek kezelése**

* **Létrehozás:** Új klaszter definiálása, paraméterek beállítása.
* **Törlés:** Nem használt klaszter eltávolítása.
* **Módosítás:** Klaszter adatok frissítése.

**4.2 Egyéb műveletek**

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

**4.3 Jogosultságok**

* Csak adminisztrátorok és szervezők kezelhetik a klasztereket.
* Minden módosítás naplózásra kerül.