



Árgép robot fejlesztése

Készítette

Kis Sándor

Programtervező Informatikus BSc

Témavezető

Nagy Péter

Külső konzulens

EGER, 2024

Tartalomjegyzék

Bevezetés	3
1. Választott technológiák bemutatása	5
1.1. .NET Keretrendszer [1]	5
1.2. Entity Framework [2]	6
1.3. Windows Presentation Foundation	6
1.4. MVVM (Model-View-ViewModel)	7
1.5. Microsoft SQL Server	8
1.6. UiPath	8
1.6.1. UiPath Studio	9
1.6.2. UiPath Orchestrator	10
1.6.3. UiPath Robot	11
1.6.4. A UiPath fejlesztés alapjai:	11
2. Megoldás	13
2.1. REST API Szerver	13
2.1.1. Adatbázis	13
2.1.2. Adatbázis inicializálása	14
2.1.3. Kontroller osztály	16
2.2. Excel Data Loader	18
2.2.1. Excel file feldolgozása	18
2.2.2. Adatkötés (data binding) [4] [5]	20
2.3. Árgép Robot	22
2.3.1. ÁrGép Robot működése	22
2.3.2. Search Workflow	23
2.3.3. Result To DB Workflow	26
2.3.4. Report Builder Workflow	26
2.4. Tesztelés	26
Összegzés	27
Irodalomjegyzék	28

Bevezetés

A mindennapi munkám során számos ismétlődő feladatokat kell elvégezni a számítógépen. Ezek a tevékenységek általában adatbevitellel és különböző, ugyanarra a sablonra épülő riportok elkészítésével kapcsolatosak. Az ilyen típusú feladatok igen időigényesek, Én naponta akár több munkaórát is ezekre fordítok. Az így eltöltött időt sokkal értékesebben is fel tudnám használni, ezáltal sokkal hatékonyabban tudnám elvégezni a munkámat.

Gondolom, nagyon sokan vannak hasonló helyzetben, olyanok akik egyhangú, monoton, ismétlődő munkát végeznek számítógépen. Ezeknek az ismétlődő munkafolyamatoknak az automatizálásával rendkívüli hatékonyságot lehet elérni úgy, hogy rengeteg munkaórát lehet megtakarítani, és még a hibalehetőségeket is a minimálisra lehet csökkenteni. Az automatizálásra számos megoldás létezik. A UiPath kiemelkedik ezen a területen és a robotfolyamat-automatizáció (RPA) terén kiváló megoldásokat kínál.

A UiPath által létrehozott technológia kulcsfontosságú szerepet játszik a munkafolyamatok automatizálásában. Azokon a területeken, ahol az ismétlődő feladatok és a rutinszerű folyamatok gyakran előfordulnak, az RPA segítségével rendkívül hatékonyan lehet robotokat alkalmazni. Például az adatbevitel, az adatellenőrzés és különböző adminisztratív folyamatok és feladatok automatizálásával az UiPath robot gyors, hatékony és precíz műveleteket végez szinte hibátlanul.

Egy másik lényeges dolog az RPA alkalmazásában az, hogy a UiPath rendszer egyszerűen lehetővé teszi az integrációt más vállalati rendszerekkel. Ez azt jelenti, hogy a meglévő informatikai infrastruktúrát könnyen kombinálhatjuk az automatizálással, anélkül, hogy nagyobb és költséges átalakításokra lenne szükség. Ez lehetővé teszi a vállalatok számára, hogy fokozatosan vezessék be az automatizációt, kezdve a legkritikusabb területekkel, majd később kiterjeszthetik azt az egész vállalati környezetre.

Az automatizáció tehát nemcsak, hogy meggyorsítja a munkafolyamatokat, de elősegíti a vállalati hatékonyságot és az emberi erőforrások felszabadítását, ezáltal javítja a vállalatok versenyképességét. Azok a vállalatok, amelyek hatékonyan alkalmazzák az automatizációt, hatalmas előnyt szerezhetnek a versenytársaikkal szemben a folyamatosan változó piaci környezetben.

Célkitűzés

A szakdolgozatom elkészítése során a UiPath technológiát felhasználva létrehoztam egy Árgép robotot, ami segít összeállítani egy asztali PC-t a lehető legolcsóbb áron. Az Árgép robot egy adatbázisból dolgozik, amelyben tárolva vannak az alkatrészek és a webshopok információi.

A felhasználók egyszerűen felsorolják a kívánt alkatrészeket egy Excel fájlban, amelyet egy asztali alkalmazás segítségével feltöltenek az adatbázisba, majd elindítják a robotot. A robot az adatbázisból kiolvassa az információkat, majd leellenőrzi a webshopokat az alkatrészek aktuális áraiért. Végül a robot riportot készít, amelyben összehasonlítást nyújt arról, hogy melyik webshopban találhatók a legolcsóbb alkatrészek, majd ezt a riportot emailben elküldi a felhasználónak.

1. fejezet

Választott technológiák bemutatása

1.1. .NET Keretrendszer [1]

.NET-keretrendszer egy olyan technológia, amely támogatja a Windows-alkalmazások és webszolgáltatások készítését és futtatását. Előzőleg a .NET nemcsak fejlesztői környezetet jelentett, hanem magában foglalta különböző szoftvereket, fejlesztőeszközöket és hardvereszközöket is, azonban ma már a .NET kifejezés kizárólag magára a keretrendszerre vonatkozik.

.NET-keretrendszer a közös nyelvi futtatókörnyezetből (CLR) és a .NET- keretrendszer osztálytárból áll. A közös nyelvi futtatókörnyezet a .NET- keretrendszer alapja. Végrehajtáskor kezeli a kódot és olyan alapvető szolgáltatásokat nyújt, mint a memóriakezelés, a száakezelés és a visszalépés, miközben szigorú típusbiztonságot és a kód pontosságának egyéb olyan formáit is kikényszeríti, amelyek elősegítik a biztonságot és a robusztusságot.

Az osztálytár olyan újrafelhasználható típusok átfogó, objektumorientált gyűjteménye, amellyel a hagyományos parancssori vagy grafikus felhasználói felületi (GUI) alkalmazásoktól kezdve az ASP.NET által biztosított legújabb innovációkon alapuló alkalmazásokig, például Web Forms és XML-webszolgáltatásokig terjedő alkalmazásokat fejleszthetők.

A Microsoft a .NET platform kiadásakor bevezetett egy új programozási nyelvet, a C#-ot. Ez a programozási nyelv egy letisztult szintaxissal rendelkező nyelv, melynek alapjait más, sikeres nyelvek vetették meg. A Microsoft egyszerűen áttekintette a legelterjedtebb programozási nyelveket, sorra vette azok minden jó tulajdonságát és hibáját. A C#-ban igyekeztek a jó tulajdonságokat maximalizálni, a rosszakat minimalizálni. Fontos megjegyezni, hogy a C# egy tisztán OOP (objektumorientált programozási) nyelv, ezzel átlépve az egyik legnépszerűbb programozási nyelv, a C++ korlátait.

1.2. Entity Framework [2]

Az Entity Framework egy objektum-relációs leképző (ORM) keretrendszer, amely lehetővé teszi a fejlesztők számára, hogy objektumorientált módon dolgozzanak relációs adatbázisokkal. Ez azt jelenti, hogy a fejlesztők az adatbázis-táblákat objektumokként kezelhetik, és nem kell SQL-utasításokat írniuk az adatok lekérdezéséhez és módosításához. Az Entity Framework kulcsfontosságú előnyei közé tartozik a megnövelt termelékenység, a kód egyszerűsítése és a kiváló hibaelhárítási képességek.

A .NET-alapú alkalmazások fejlesztéskor, amikor adatbázis-hozzáférésre van szüksége, az Entity Framework kiváló választás.

1.3. Windows Presentation Foundation

A Windows Presentation Foundation (WPF) egy grafikus felhasználói interfész (GUI) keretrendszer, amelyet a Microsoft fejlesztett ki Windows alkalmazások készítéséhez. A WPF segítségével esztétikus, interaktív és gazdag felhasználói élményt kínáló alkalmazások fejlesztése lehetséges.

WPF néhány jellemzője:

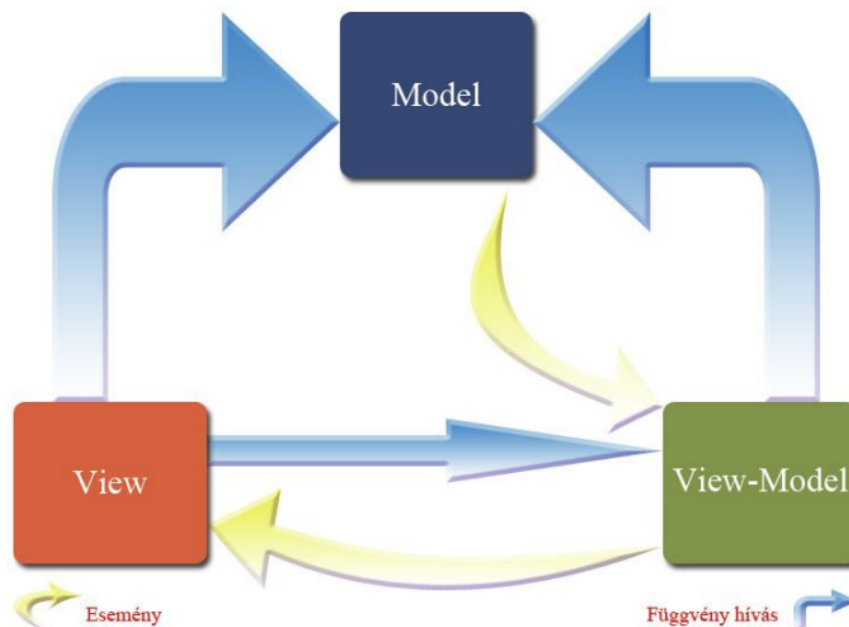
- **Deklaratív XAML (eXtensible Application Markup Language):** A WPF az XML-alapú XAML-t használja az alkalmazások felhasználói interfészének deklaratív leírásához. Ez lehetővé teszi a tervezők és fejlesztők számára, hogy könnyen elkészítsék és szerkesszék a felhasználói felületet.
- **Adatkötés (Data Binding):** Lehetővé teszi az adatok dinamikus kapcsolatát a felhasználói interfésszel. Az adatkötés segítségével az adatok automatikusan frissülnek, amikor azok megváltoznak, és így az alkalmazások dinamikusabbak és interaktívabbak lehetnek.
- **Stílusok és Sablonok (Styles and Templates):** A WPF lehetővé teszi a felhasználói interfész elemeinek stílusainak és sablonjainak egyszerű definiálását és alkalmazását. Ez segíti a dizájn egységesítését és az alkalmazások testreszabhatóságát.
- **Vektorgrafika és Animációk:** A WPF támogatja a vektorgrafikát, amely lehetővé teszi a skálázható és magas felbontású grafikai elemek használatát. Emellett a keretrendszer könnyen kezeli az animációkat is.

1.4. MVVM (Model-View-ViewModel)

Az MVVM egy olyan tervezési minta, amelynek célja az alkalmazások strukturáltabbá tétele és könnyebb karbantarthatósága. A nevét a három fő komponensének nevéből kapta:

- **Model:** A Model reprezentálja az alkalmazás üzleti logikáját és adatelérési rétegét. Ez a rész felelős az adatok kezeléséért és az üzleti szabályok végrehajtásáért.
- **View:** A View felelős a felhasználói felület megjelenítéséért. Ez a rész megjeleníti az adatokat és érzékeli a felhasználói interakciókat.
- **ViewModel:** A ViewModel egy köztes réteg a Model és a View között. Feladata a felhasználói felület és az üzleti logika közötti kapcsolat fenntartása. A ViewModel feldolgozza a felhasználói interakciókat a View-től, valamint frissíti a View-t az adatokból, amelyeket a Modelből kap. A ViewModel segítségével az alkalmazás jól elkülöníthető módon oszlik meg, ami könnyebb tesztelést és karbantarthatóságot eredményez.

Az MVVM használatával az alkalmazás könnyebben bővíthető és karbantartható, és lehetőséget ad az egységtesztekre is. A WPF keretrendszer kifejezetten támogatja az MVVM tervezési mintát, és könnyen kezelhető adatkötést kínál a View és a ViewModel között.



1.1. ábra. Az MVVM modell [3]

1.5. Microsoft SQL Server

A Microsoft SQL Server a Microsoft által kifejlesztett relációs adatbázis-kezelő rendszer, mely az SQL (Structured Query Language) nyelvet használja az adatok tárolására, lekérdezésére és kezelésére.

A SQL Server eredetileg kifejezetten a Windows operációs rendszerre lett tervezve, de az újabb verziók már támogatják más operációs rendszereken való futtatást is, például Linuxon és Docker konténerekben, ezáltal rugalmasságot biztosítanak a különböző környezetekben történő telepítéshez és üzemeltetéshez.

Az SQL Server Management Studio (SSMS) egy grafikus felhasználói felület (GUI), mely segíti a felhasználókat a Microsoft SQL Server adatbázis-kezelő rendszerével kapcsolatos feladatok egyszerű és hatékony végrehajtásában. Az SSMS lehetővé teszi az adatbázisok tervezését, létrehozását, karbantartását és felügyeletét, valamint a lekérdezések írását és futtatását.

1.6. UiPath

UiPath története 2005-re nyúlik vissza, ekkor alapította meg a DeskOver vállalatot Daniel Dines és Marius Tirca egy bukaresti lakásban. Kezdetben szoftverautomatizációs eszközöket és fejlesztői készleteket hoztak létre olyan neves cégek számára, mint az IBM, a Google és a Microsoft, akik ezeket beépítették saját termékeikbe.

Az igazi áttörés 2012-ben jött el, ekkor találó módon egy ügyfél volt az, aki rámutatott a kezdetleges RPA (robotfolyamat-automatizálás) piacon rejlő lehetőségekre a cég számára.

2013-ban a cég piacra dobta az első UiPath Desktop Automation termékcsaládot, amely eszközöket biztosított a vállalatok számára az adminisztratív, ismétlődő feladatok automatizálásához. A vállalat életében a következő mérföldkő az 2015-ös év volt. Ekkor mutatták be az új vállalati platformot, és ekkor vette fel a cég a UiPath nevet is.

A UiPath hamarosan vezető szereplővé vált az RPA piacon, és az általuk kifejlesztett eszközök és szolgáltatások kiválóan segítették a vállalatokat az üzleti folyamatok hatékonyabbá és termelékenyebbé tételében. A UiPath termékei között szerepel a UiPath Studio, ami a fejlesztőknek segít az automatizációs feladatok létrehozásában és kezelésében, valamint a UiPath Orchestrator, egy eszköz a robotok menedzselésére és ellenőrzésére.

A UiPath jelenlegi célkitűzése a Fully Automated Enterprise - a teljesen automatizált vállalat - koncepciójának megvalósítása. Ennek révén a vállalatok teljes mértékben kiaknázhathatják bennük rejlő lehetőségeket az automatizáció segítségével, felszabadítva az emberi munkaerőt a kreatívabb feladatok elvégzésére.

1.6.1. UiPath Studio

A UiPath Studio a UiPath vállalati RPA platformjának központi eleme, egy vizuális fejlesztői környezet, amely lehetővé teszi automatizált munkafolyamatok egyszerű és gyors létrehozását. Ez azt jelenti, hogy felhasználóknak nem kell kódolási ismeretekkel rendelkezniük a használatához. Akár üzleti felhasználók, akár fejlesztők számára hasznos eszköz, különböző automatizálási igények megoldására.

Studio főbb jellemzői:

- **Vizuális fejlesztés:** Egyszerűen összeállíthatók automatizált munkafolyamatok különféle tevékenységekből, mint például az alkalmazás indítása, adatok bevitele, adatgyűjtés, döntéshozatal stb. a drag-and-drop funkcióknak köszönhetően.
- **Felvett tevékenységek:** Rögzíthetjük a manuális műveleteket, majd a Stúdióban lejátszhatjuk és finomhangolhatjuk azokat automatizációs feladatokká.
- **Kiterjedt tevékenységpaletta:** Számos előre definiált tevékenység áll rendelkezésre különböző automatizálási célokra, például webes, asztali és Citrix alkalmazásokhoz, adatmanipulációhoz, hibakezeléshez stb.
- **Változó szintű komplexitás:** Egyszerű automatizációk készítése kezdő felhasználók számára, míg a tapasztaltabbak összetettebb munkafolyamatokat is létrehozhatnak.
- **Hibakezelés:** Beállíthatók hibakezelési lépések, hogy az automatizáció megfelelően reagáljon váratlan események esetén.
- **Logok és audit naplók:** Részletes naplókat készíthetünk a munkafolyamatok végrehajtásáról, ami segít a hibakeresésben és a monitoringban.
- **Integráció más eszközökkel:** Könnyen integrálható más RPA komponensekkel, AI funkciókkal, harmadik fél eszközökkel és API-kkal.

Studio változatai:

- **UiPath StudioX:** A Studio egyszerűsített változata, kezdőknek ajánlott.
- **UiPath Studio:** Teljes körű fejlesztői környezet, komplexebb automatizáláshoz.
- **UiPath Studio Web:** Webalapú fejlesztői környezet, online alkalmazások automatizálására.

1.6.2. UiPath Orchestrator

Az UiPath Orchestrator a UiPath vállalati RPA platformjának egy másik kulcsfontosságú eleme. Ez egy központi vezérlőpult, amely lehetővé teszi az RPA-folyamatok ütemezését, figyelését, kezelését és skálázását. Gyakorlatilag az Orchestrator felel az automatizált munkafolyamatok életciklusának teljes körű felügyeletéért.

Orchestrator főbb jellemzői:

- **Központi irányítás:** Lehetővé teszi az összes RPA robot és munkafolyamat felügyeletét egyetlen helyről.
- **Ütemezés és indítás:** Automatizált munkafolyamatok ütemezése meghatározott időpontokra, eseményekre vagy igényekre reagálva.
- **Monitoring:** Valós időbeli betekintést nyújt futó és befejezett munkafolyamatról. Láthatjuk az aktuális állapotot, a végrehajtási időt, a feldolgozott adatok mennyiségét, valamint bármilyen hibaüzenetet vagy figyelmeztetést.
- **Teljesítménymutatók:** Az Orchestrator számos előre definiált teljesítménymutatót (KPI) kínál, amelyek segítségével követni tudjuk a munkafolyamatainak hatékonyságát és sikerességét.
- **Riportálás:** Beépített jelentéskészítő eszközöket tartalmaz, amelyekkel könnyen létrehozhatók és exportálhatók jelentések a munkafolyamatok teljesítményéről, hibákról és trendekről.
- **Biztonsági hozzáférés-vezérlés:** Lehetővé teszi hozzáférési szintek beállítását a felhasználók számára a biztonságos és ellenőrzött működés érdekében.

Orchestrator előnyei:

- **Megnövelt hatékonyság:** Automatizált folyamatok központosított kezelésével csökkenti a manuális felügyeleti terheket és növeli a hatékonyságot.
- **Jobb láthatóság:** valós időbeli betekintést nyújt munkafolyamatok teljesítményébe, lehetővé téve a hibák gyors azonosítását és megoldását.
- **Skálázhatóság:** Könnyedén skálázható a platform a növekvő automatizálási igényeknek megfelelően.
- **Biztonság:** Biztonságos környezetet biztosít az RPA-folyamatok futtatásához, megfelelően a vállalatok biztonsági előírásainak.

1.6.3. UiPath Robot

Az UiPath Robot egy szoftverrobot, mely a számítógépen elvégzi az automatizált, ismétlődő manuális feladatokat. A robotot a UiPath Studio grafikus felületén fejleszthetjük ki, ahol a munkafolyamatokat programozhatjuk a robot számára. Az UiPath Robotnak két fő típusa van. A Felügyelt (attended) robot emberi beavatkozást igényel a munkafolyamatok futtatásához. A felhasználónak manuálisan kell elindítania a robotot, és meg kell adnia a szükséges bemeneteket. Felügyelet nélküli (unattended) robot önállóan futtatja a munkafolyamatokat, emberi beavatkozás nélkül. A robotot előre be kell konfigurálni a szükséges bemenetekkel és kimenetekkel.

A UiPath Robot főbb jellemzői:

- **Megbízható:** A UiPath Robot robusztus és megbízható, így a munkafolyamatok zökkenőmentesen futnak, minimális felügyelet mellett.
- **Rugalmas:** A UiPath Robot széles skálájú feladatok automatizálására használható, beleértve az adatbevitelt, az adatkivonást, a webes kaparást, az e-mailek kezelését és a PDF-fájlok feldolgozását.
- **Könnyen használható:** A UiPath Robot intuitív grafikus felülettel rendelkezik, így a kódolási ismeretek nélküli felhasználók is könnyen használhatják.
- **Skálázható:** A UiPath Robot alkalmas nagyobb volumenű munkafolyamatok kezelésére is, így a vállalatok könnyen skálázhatják az automatizációs projektjeiket.
- **Integrálható:** A UiPath Robot könnyen integrálható más üzleti alkalmazásokkal és rendszerekkel, lehetővé téve a szervezetek számára az egységes munkafolyamatok kialakítását.

1.6.4. A UiPath fejlesztés alapjai:

Mint már az előző fejezetben említettem, a fejlesztés a UiPath Studio grafikus felületén történik. Itt több száz előre elkészített activity található. Ezek az activity-k teszik lehetővé a különböző feladatok automatizálását. Ezek a feladatok lehetnek például adatbázis-műveletek, fájlok kezelése vagy webes feladatok, amelyeket automatizálni szeretnénk. Érdeemes megjegyezni, hogy a beépített activity-ken kívül a Studio csomagkezelőjével külső fejlesztők által írt activity-eket is telepíthetünk, vagy akár saját magunk is írhatunk activity-eket. Ehhez a Microsoft Visual Studio fejlesztői környezete szükséges.

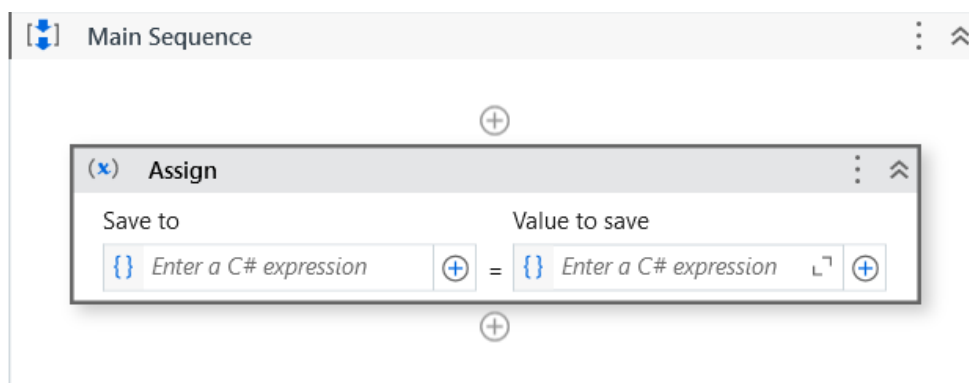
Workflow-k UiPath-ban: [7]

Az activity-ket drag and drop módszerrel egyszerűen munkafolyamatokba (workflow) szervezhetjük. Attól függően, hogy milyen feladatokat szeretnénk automatizálni, a UiPath a következő munkafolyamatokat kínálja:

- **Sequence:** Egyszerű, lineáris workflow, amely felülről lefelé haladva, egymás után hajtja végre az activity-ket. Előnye, hogy egyszerű összeállítani, jól átlátható és könnyen olvasható. Legtöbbször elegendő is a kívánt feladat automatizálására.
- **Flowchart:** Vizuális workflow, akkor érdemes használni, amikor az automatizálási folyamat több elágazással, döntési csomóponttal rendelkezik. Ez a legjobb választás a nem lineáris folyamatok automatizálására.
- **State Machine:** Ez egy összetettebb workflow, akkor érdemes használni, amikor a folyamat különböző állapotokat, úgynevezett state-eket vehet fel. A tranzakciók irányítják az állapotok változását, vagyis azt, hogy egyik state-ből mikor léphetünk tovább egy másikba. Minden tranzakció feltételhez kötött, és ha a feltétel teljesül, akkor aktiválódik a tranzakció, és a State Machine állapotot vált.
- **Global Exception Handler:** Ez egy speciális workflow, amely nem kezelt kivételek esetén automatikusan lefut. Használata segít az automatizálás megbízhatóságának növelésében.

Activity-k UiPath-ban: [8]

Az activity-k a UiPath automatizálás alapvető építőkövei, amelyek reprezentálják azokat a műveleteket, amelyeket az automatizálás során végre szeretnénk hajtani. Minden egyes activity egy speciális funkciót hajt végre. Ilyen funkció lehet például egy e-mail küldése, egy űrlap adatokkal való kitöltése vagy egy adatbázis lekérdezés. Az activity-k paraméterekkel könnyen testreszabhatók, így az adott igényekhez alakíthatók.



1.2. ábra. Assign activity Sequence workflow-ban

2. fejezet

Megoldás

A projektem három fő részből áll. Az első rész egy REST API szerver, amely az ASP.NET Core keretrendszerre épül. A második rész egy desktop alkalmazás, amely elküldi az adatbázisba elmentendő adatokat a REST API-n keresztül a szervernek. Ezt az alkalmazást ExcelDataLoader-nek neveztem el, és a Windows Presentation Foundation keretrendszer segítségével készítettem el. A projekt harmadik része maga a UiPath Árgép robot, amely az automatizálási feladatokat látja el.

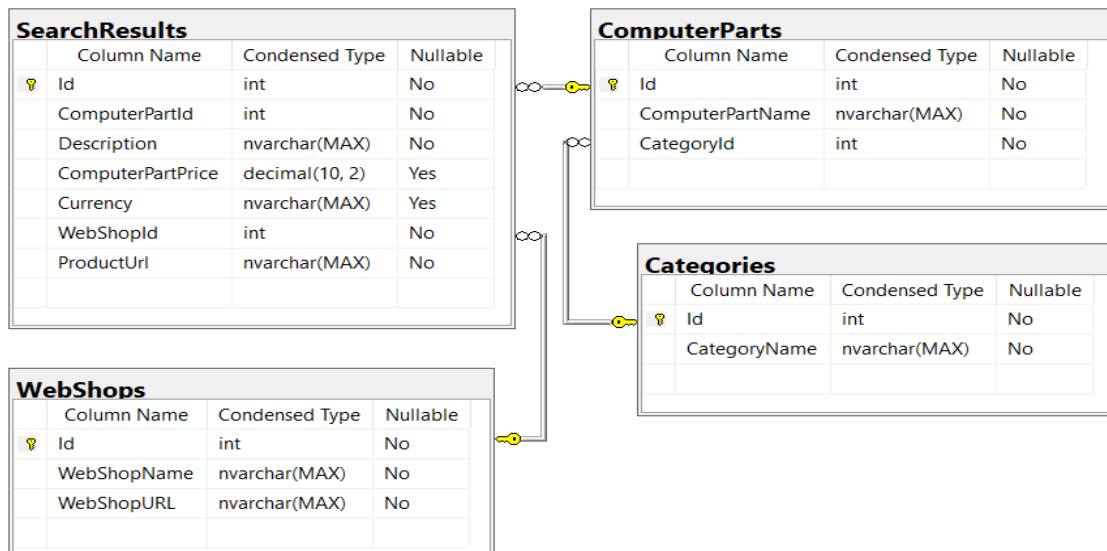
2.1. REST API Szerver

Ez a komponens biztosítja a kommunikációs felületet a kliensalkalmazás és az adatbázis között, lehetővé téve az adatbázis műveletek végrehajtását HTTP kéréseken keresztül.

2.1.1. Adatbázis

Az alkalmazásban nincs szükség sok adat tárolására, csupán a webáruházak nevét, webcímét, valamint a megvásárolni kívánt számítógép alkatrészekhez tartozó alapvető információkat kell eltárolni, illetve kiolvasni. Emellett a robot keresési eredményeit is el kell menteni az adatbázisba, majd a keresési eredmények közül a legolcsóbb alkatrészek adatait is ki kell nyerni. Az adatbázis felépítése a 2.1 ábrán látható.

Az adatbázisban összesen négy tábla található: a ComputerParts, WebShops, és Categories táblákba az adatokat az Excel Data Loader asztali alkalmazás segítségével egyszerűen feltölthetjük egy Excel-fájlból. A negyedik, egyben fő táblánk a SearchResults tábla, amelybe a robot az összes alkatrészről talált keresési eredményt elmenti majd el. Ebben a táblában két idegen kulcs található, amelyek a ComputerParts és a WebShops táblákra mutatnak.



2.1. ábra. SSMS adatbázis diagram

2.1.2. Adatbázis inicializálása

Az adatbázis inicializálásához Entity Framework-öt használtam. Az Entity Framework használatakor az adatbázis inicializálását különböző módon lehet elvégezni; választható a Code First, Database First és Model First megközelítés is.

Entity

Én a Code First megközelítést választottam az adatbázis létrehozására. Ebben a megközelítésben az adatbázis sémát a kód alapján hozzuk létre. Első lépésben létrehoztam három Entity-t, azaz három C# osztályt melyek reprezentálják az adatbázis táblákat. Az alábbi kódrészlet a SearchResult Entity kódját mutatja be.

2.1. kód. SearchResult Entity

```

1 public class SearchResult
2 {
3     public int Id { get; set; }
4     public int ComputerPartId { get; set; }
5     public ComputerPart ComputerPart { get; set; } = null!;
6     public string Description { get; set; } = null!;
7
8     [Column(TypeName = "decimal(10,2)")]
9     public double? ComputerPartPrice { get; set; }
10    public string? Currency { get; set; }
11    public int WebShopId { get; set; }
12    public WebShop WebShop { get; set; } = null!;
13    public string ProductUrl { get; set; } = null!;
14 }

```

Az érdekes megjegyezni, hogy Entity Framework esetén az Id egy speciális property név, amely az elsődleges kulcsot reprezentálja a létrehozandó táblában. Az elsődleges kulcsnak nem feltétlenül kell az Id nevet kapnia, de ha más nevet választunk, akkor a property-t el kell látnunk a [Key] attribútummal.

Másik érdekes dolog az `= null!` használata. A SearchResult osztályban a több property-nek is `null!` értéket adtam. Ez azt jelenti, hogy a property-knek mindig értéket kell kapniuk, mielőtt a SearchResult entitás az adatbázisban elmentődik. Ha nem adunk értéket az ilyen property-knek, akkor az Entity Framework hibát fog dobni, mivel az adatbázisban ezek a mezők nem engedélyezik a null értékeket.

A `= null!` kifejezés használata nem kötelező, de előnyökkel jár. Segít megelőzni a hibákat és a nem kívánt viselkedést, hogy ha egy propertynek nem szabad null értéket kapnia. Biztosítja, hogy az adatbázisban tárolt adatok konzisztensek és javítja a kód olvashatóságát is, mivel egyértelművé teszi, hogy a propertynek mindig értéket kell adni.

Database Context osztály

A következő lépésben létrehoztam a Database Context osztályt. Ez az osztály az Entity Framework központi eleme, amelynek a DbContext osztályból kell származnia. Felelős a relációs adatbázissal való kapcsolódásért, az objektumok leképezéséért az adatbázisra a Code First megközelítés esetén, felelős a változások nyomon követéséért (change tracking), valamint az adatbázis-migrációk kezeléséért. Ez nemcsak hogy szabványos gyakorlat az Entity Framework alkalmazása során, hanem elengedhetetlen is a hatékony adatbázis-kezeléshez. Az következő kódrészlet a Database Context osztály kódját mutatja be.

2.2. kód. ExcelUploadContext osztály

```
1 public class ExcelUploadContext : DbContext
2 {
3     protected ExcelUploadContext() {}
4
5     public ExcelUploadContext(DbContextOptions
6         <ExcelUploadContext> options) : base(options) {}
7
8     public DbSet<Category> Categories { get; set; }
9     public DbSet<WebShop> WebShops { get; set; }
10    public DbSet<ComputerPart> ComputerParts { get; set; }
11    public DbSet<SearchResult> SearchResults { get; set; }
12 }
```

Az osztályban található egy paraméter nélküli és egy paraméterezett konstruktor, valamint négy DbSet property.

A paraméterezett konstruktor egy `DbContextOptions<ExcelUploadContext>` típusú objektumot vár paraméterként. Ez az objektum tartalmazza a konfigurációs beállításokat az adott adatbázis kapcsolathoz.

A négy `DbSet` property (`Categories`, `WebShops`, `ComputerParts`, `SearchResults`) pedig nem más, mint az adatbázisban lévő táblák reprezentációi. Ezek a property-k teszik lehetővé, hogy a kódban az entitásokon keresztül műveleteket végezzek a táblákkal. Erre példa az alábbi kód.

2.3. kód. `GetAllCategories` metódus

```
1 public JsonResult GetAllCategories()
2 {
3     try { var result = _context.Categories.ToList();
4         return new JsonResult(result);
5     }
6     catch (Exception ex){
7         return BadRequestResult("Error occurred while
8             ↳ retrieving categories.");
9     }
10 }
```

Migráció

Az Entity osztályok és a Database Context osztály elkészítése után az Entity Framework a migráció segítségével automatikusan elkészíti vagy frissíti az adatbázist. Az adatbázisban lévő táblák az előzőleg elkészített entity osztályok definíciójának megfelelően jönnek létre.

Az `Add-Migration InitialCreate` konzolos parancs használatával az Entity Framework ellenőrzi az entity osztályokat és létrehozza a megfelelő SQL parancsokat az adatbázis inicializálásához vagy frissítéséhez.

Az `Update-Database` paranccsal az Entity Framework végrehajtja az előzőleg generált SQL parancsokat, és az adatbázist az entity osztályok definícióinak megfelelően elkészíti vagy frissíti.

Ez a megoldás azért is kényelmes, mert a fejlesztés során az entity osztályokon vagy a Database Context osztályon történő változásokat egy új migráció segítségével nagyon könnyen fel tudom vinni az adatbázisba, ezáltal biztosítani tudom, hogy az adatbázis mindig szinkronban lesz a kód váltoásaival.

2.1.3. Kontroller osztály

A kontroller osztályok az ASP.NET Core alkalmazásokban felelősek a kérések fogadásáért és a válaszok kiszolgáltatásáért. Tehát a kliens a kéréseit a kontroller osztálynak küldi el, és a kontroller osztály válaszol a kérésekre a megfelelő adatokkal vagy válaszkóddal.

Az alkalmazásom serveroldali logikáját egyetlen vezérlő osztállyal valósítom meg, ez a `ComputerPartController` osztály. Ez az osztály a `ControllerBase` osztályból származik, ami biztosítja az alapvető funkciókat a HTTP kérések és válaszok kezeléséhez. Továbbá a `ComputerPartController` osztály az `ExcelUploadContext` példányát használja, amit konstruktor alapú dependency injection segítségével kap meg. Ez azt jelenti, hogy a szükséges függőségek az osztály konstruktorán keresztül injektálódnak be. Ez látható az alábbi kódrészletben.

2.4. kód. Dependency injection az `ExcelUploadContext` példányához

```
1 public class ComputerPartController : ControllerBase
2 {
3     private readonly ExcelUploadContext _context;
4
5     public ComputerPartController(ExcelUploadContext context)
6     {
7         _context = context;
8     }
9     // ...
10 }
```

HTTP metódusok

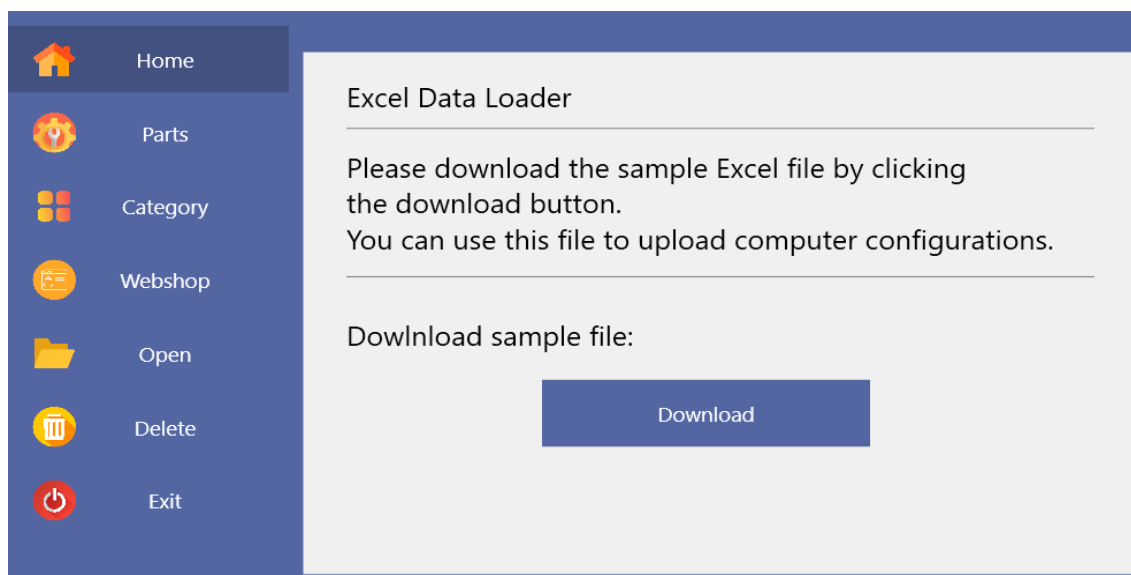
Az adatbázison végzett műveleteket HTTP metódusok segítségével valósítom meg:

- **Create:** A számítógépalkatrészek, kategóriák és webáruházak hozzáadásáért a POST metódusok felelnek meg. A `ComputerPartsUpload`, `CategoryUpload` és `WebshopUpload` metódusok a kliens által küldött adatok alapján új elemeket hoznak létre és mentik azokat az adatbázisba.
- **Read:** Az összes számítógépalkatrész, kategória és webáruház lekérése a GET metódusokkal valósul meg. A `GetAllComputerParts`, `GetAllCategories` és `GetAllWebShops` metódusok a megfelelő entitásokat kérdezik le az adatbázisból és visszaküldik a kliensnek.
- **Delete:** Az összes számítógépalkatrész, kategória és webáruház törlésére a DELETE metódusok szolgálnak. A `DeleteAll` metódus az összes entitást törli az adatbázisból.

A serveroldalon csak a Create, Read és Delete adatbázisműveletek vannak implementálva. Az Update műveletre nincs szükség, mert minden egyes új keresés előtt az adatbázisból törlődik minden adat.

2.2. Excel Data Loader

Az adatok Excel fájlból történő beolvasására és azok szerverre történő elküldésére készítettem egy desktop alkalmazást, amelyhez a Windows Presentation Foundation (WPF) keretrendszert választottam. Azért esett a választásom a WPF-re, mert kiválóan támogatja a látványos felhasználói felület kialakítását, valamint könnyen kezelhető az adatok megjelenítése és a felhasználóval való interakció. A fejlesztés során az MVVM (Model-View-ViewModel) tervezési mintát követtem, amely lehetővé tette a karbantartható és jól struktúrált kód létrehozását, valamint az adatkezelés és a felhasználói felület szétválasztását.



2.2. ábra. Excel Data Loader asztali alkalmazás

2.2.1. Excel file feldolgozása

Többféle NuGet csomag közül lehet választani, ha Excel műveleteket szeretnénk integrálni a kódunkba. Mindegyik ilyen NuGet csomagnak vannak előnyei és hátrányai is. A dolgozatomban a Microsoft Office Interop Excel csomagot választottam. Ennek a csomagnak a legnagyobb előnye, hogy közvetlen kapcsolatot biztosít az Excel alkalmazással, így egyszerűen és könnyen lehetővé teszi az Excel funkcióinak és az adatoknak a programozott kezelését. A legnagyobb hátránya talán az, hogy az Excelnek telepítve kell lennie az alkalmazást futtató összes számítógépen. Emellett, mivel a programunk egy külső alkalmazással interaktál, az interop hívások lassabbak lehetnek, különösen akkor, amikor nagyobb adatmennyiséggel dolgozunk.

Az Excel fájl feldolgozására egy külön statikus osztályt hoztam létre `ExcelFileHandlerInterop` néven. Ennek az osztálynak egyetlen statikus metódusa van, a `ReadExcelFile`, amely paraméterként megkapja az Excel fájl elérési útvonalát és a munkalap

számát, amin a beolvasandó adatok találhatóak. Ez a módszer egy DataTable objektumot ad vissza, amelyben a beolvasott adatok vannak tárolva. Fontos része a kódnak egy új Application objektum létrehozása a Microsoft.Office.Interop.Excel könyvtárból, amely az Excel alkalmazást képviseli. Ennek az Application objektumnak a beépített módszereinek keresztül megvalósíthatjuk a különböző Excel műveleteket. Lekérhetjük a munkafüzetet, a munkafüzet munkalapjait, és beállíthatunk tartományokat a munkalapokon. Kérhetjük a tartomány oszlopainak és sorainak számát, így egyszerűen ki tudjuk nyerni a kívánt adatokat. Ennek a megvalósítása látható az alábbi kódrészletben.

2.5. kód. Excel fájl beolvasása és kezelése .NET Interop használatával

```
1 internal static class ExcelFileHandlerInterop
2 {
3     public static System.Data.DataTable ReadExcelFile(string
4         ↪ filePath, int sheetNumber)
5     {
6         System.Data.DataTable dataTable = new System.Data.
7             ↪ DataTable();
8         Application excelApp = new Application();
9         Workbook workbook = excelApp.Workbooks.Open(filePath);
10        Worksheet worksheet = (Worksheet)workbook.Sheets[
11            ↪ sheetNumber];
12
13        Range range = worksheet.UsedRange;
14        int rowCount = range.Rows.Count;
15        int colCount = range.Columns.Count;
16        //...
17        workbook.Close(false);
18        excelApp.Quit();
19
20        System.Runtime.InteropServices.Marshal.ReleaseComObject(
21            ↪ worksheet);
22        System.Runtime.InteropServices.Marshal.ReleaseComObject(
23            ↪ workbook);
24        System.Runtime.InteropServices.Marshal.ReleaseComObject(
25            ↪ excelApp);
26
27        return dataTable;
28    }
29 }
```

Nagyon fontos az erőforrások felszabadítása. Ügyelni kell arra, hogy bezárjuk az Excel alkalmazást, és hogy felszabadítsuk a COM objektumokat. Ha ezt nem tesszük meg, akkor olyan problémákkal szembesülhetünk, mint például a memória szivárgás, ami azt jelenti, hogy a memória használat folyamatosan növekszik, és ezáltal komoly teljesítménycsökkenés következhet be a rendszerben.

2.2.2. Adatkötés (data binding) [4] [5]

Az adatkötés vagy adatkapcsolás a felhasználói felület elemeinek és a háttérben lévő adatmodellek összekapcsolását jelenti, így a felhasználói felületen megjelenő adatok mindig automatikusan frissülnek.

A program elkészítésekor én a Model-View-ViewModel (MVVM) tervezési mintát követtem, ahol az adatkötés a View és a ViewModel között valósul meg. A View az, ahol az adatkötés deklarálása megtörténik, a ViewModel pedig biztosítja azt, hogy a View értesüljön a változásokról, de ehhez először a ViewModelnek implementálnia kell az `INotifyPropertyChanged` interfészt. Ez az interfész a `PropertyChanged` eseményt definiálja, amit egy tulajdonság megváltozásánál kell kiváltani.

Első lépésben létrehoztam egy `ViewModelBase` osztályt, amely implementálja az `INotifyPropertyChanged` interfészt és ez osztály az őse az összes többi `ViewModel` osztálynak.

2.6. kód. `INotifyPropertyChanged` interfész implementálása

```
1 public class ViewModelBase : INotifyPropertyChanged
2 {
3     public event PropertyChangedEventHandler PropertyChanged;
4
5     public void OnPropertyChanged ([ CallerMemberName ] string
        ↳ propName = null)
6     {
7         PropertyChanged?.Invoke (this, new
            ↳ PropertyChangedEventArgs (propName));
8     }
9 }
```

Ebben az osztályban az `OnPropertyChanged` metódus egy `string` típusú paramétert vár, amely a annak a tulajdonságnak a nevét jelöli amit figyelünk, hogy változik e. Amikor ennek tulajdonságnak az értéke megváltozik, a `PropertyChanged` esemény kiváltódik. A View figyeli ezt az eseményt, és amikor a `PropertyChanged` esemény bekövetkezik, automatikusan frissíti a bindelt elemeket.

Az alábbi `WebShopsViewModel` osztály kódrészletben látható, hogyan valósítottam meg az `OnPropertyChanged` metódus használatát.

Az osztálynak van egy `WebShops` nevű, `WebShop` objektumokat tartalmazó `ObservableCollection` property-je. Ennek a property-nek a setterében hívom meg az `OnPropertyChanged` metódust. Alap esetben ennek a metódusnak paraméterként át kellene adni a property nevét, de a `CallerMemberName` attribútum használata miatt a property neve automatikusan megadásra kerül, így ezzel nekünk nem kell foglalkoznunk.

2.7. kód. Az OnPropertyChanged használata a WebShopsViewModel osztályban

```
1 public class WebShopsViewModel : ViewModelBase
2 {
3     private ObservableCollection<WebShop> webShops;
4
5     public ObservableCollection<WebShop> WebShops
6     {
7         get { return webShops; }
8         set
9         {
10             if (webShops != value)
11             {
12                 webShops = value;
13                 OnPropertyChanged();
14             }
15         }
16     }
17
18     //...
19 }
```

Amikor a WebShops property értéke megváltozik, például amikor a Collection-hoz adunk hozzá új WebShop objektumokat vagy eltávolítunk belőle elemeket, akkor az OnPropertyChanged metódus meghívódik, és ennek hatására a View azonnal frissül a data binding-nak köszönhetően.

Maga az adatkötés deklarálása a View-ban, XAML-ben megtörténik. Az alábbi kód-részletben egy DataGridet definiálok, és a DataGridben megjelenítendő adatok forrása a ViewModel WebShops property-jéhez van kötve, ahogy az 5. sorban látható.

2.8. kód. Adatkötés DataGrid és Webshops Property között

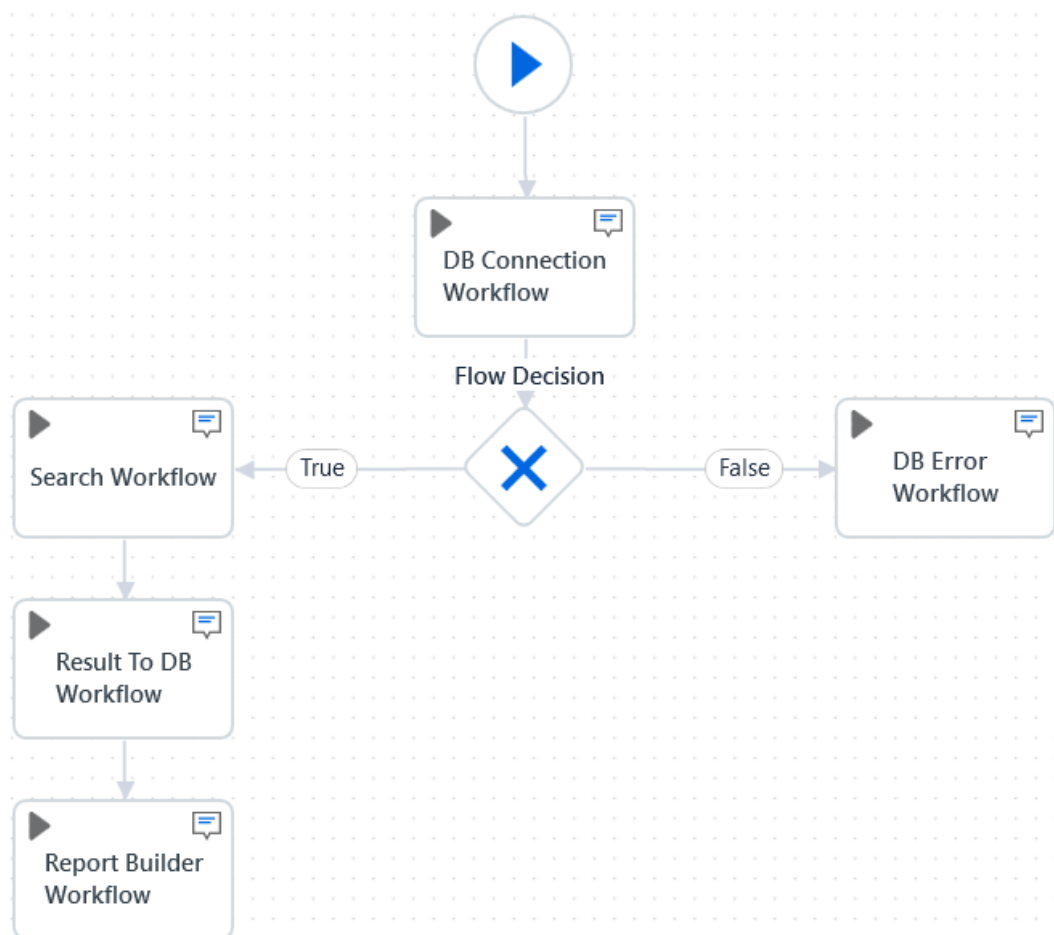
```
1 <Grid>
2     <DataGrid x:Name="DataGridWebShops"
3         Margin="10,50,10,10"
4         AutoGenerateColumns="False"
5         ItemsSource="{Binding WebShops}"
6         Style="{StaticResource DataGridStyle}">
7         <!-- ... -->
8     </DataGrid>
9     <!-- ... -->
10 </Grid>
```

Az adatkötést azért tartottam fontosnak részletezni a dolgozatomban, mert bár a program fő funkciója az Excel fájl feldolgozása és a feldolgozott adatok továbbítása a szerverre, a program képes lekérdezni a szerverről adatokat is, amelyeket a data binding segítségével dinamikusan megjelenít.

2.3. Árgép Robot

Az adatgyűjtés szempontjából nézve az árgépek, illetve az ár-összehasonlító alkalmazások többféleképpen működhetnek. Vannak olyan alkalmazások, amelyek a webáruházak API-jain keresztül gyűjtik be a termékek árait, illetve a termékekre vonatkozó egyéb információkat. Más alkalmazások egy termékfájlt kapnak a webshopoktól, és ez a termék feed tartalmazza azokat az adatokat, amelyek megjelenjenek ezeknek az alkalmazásoknak a felületén. Ez a termék feed általában egy XML vagy CSV fájl, amit a webáruházak rendszeresen frissítenek és bizonyos időközönként újraküldenek az árgép alkalmazásoknak. Ilyen módon működik a <https://www.argep.hu> webalkalmazás is.

2.3.1. ÁrGép Robot működése



2.3. ábra. ÁrGép Robot Main Workflow

Az én ÁrGép robotom ezzel szemben minden egyes, a felhasználó által kiválasztott webshop weboldalára elnavigál, és az oldalon rákeres minden egyes számítógép alkatrészre, amit meg szeretnénk vásárolni. Ehhez először a robotnak le kell kérdezni ezeket az adatokat az adatbázisból. Ez a **DB Connection Workflow**-ban történik meg. Itt a

robot a **Connect to database** activity segítségével kapcsolatot létesít az adatbázissal. Ha az adatbázishoz való kapcsolódás során valami hiba lépne fel, akkor a **DB Error Workflow** fut le. Itt a robot egy e-mailben értesíti a felhasználót, hogy a kapcsolódás probléma miatt a keresést nem tudta végrehajtani és a robot futása leáll.

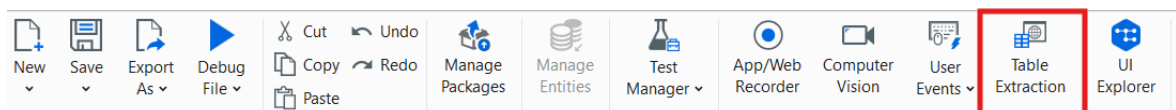
Sikeres kapcsolat esetén a robot lekérdezi az összes adatot a **WebShops**, **Categories** és **ComputerParts** táblákból, majd eltárolja ezeket az adatokat három **DataTable** adattípusú változóban. Ezt követően a robot ezeket a változókat átadja a **Search Workflow**-nak. Ebben a workflow-ban a robot végrehajtja a keresést, és a keresési eredményeket elmenti egy **DataTable** adattípusú változóban [9]. Ez az adattípus tökéletes választás, hiszen a **DataTable** alkalmas arra, hogy táblázatos formában sorokat és oszlopokat kezeljen, akár csak egy Excel-táblázat vagy egy adatbázis-tábla.

2.3.2. Search Workflow

A webshopokban történő keresésre és az adatok automatizált kigyűjtésére két különböző megközelítést választottam. Mindkét megközelítésnek vannak előnyei és hátrányai, attól függően, hogy milyen webshopban kell dolgoznunk.

Webshop specifikus keresési megoldás

Mindannyian vásároltunk már online, és tudjuk, hogy amikor rákeresünk egy tetszőleges webáruházban egy termékre, a keresési eredmény mindig egy nagyon jól strukturált, táblázatos formában jelenik meg. Ilyen strukturált adatok kinyerésére a UiPath-ban rendelkezésre áll az úgynevezett **Table Extraction** [10] adatkinyerési technológia.



2.4. ábra. UiPath Table Extraction

Amikor elindítjuk a **Table Extraction** varázslót, az képes automatikusan felismerni a táblázatos formátumokat és azok szerkezetét. Ebben a táblázatban a robotnak meg kell mutatni, hogy melyek azok az adatok, amelyeket ki szeretnénk nyerni. Mivel a táblázat felépítése ismert a robot számára, elég csak egyetlen termék nevére rákattintani, és az algoritmus azonosítja a táblázatban található összes többi termék nevét is. Hasonló módon kinyerhetjük a termékek árát, valamint az összes többi megjelenített, számunkra releváns adatot is. Az így kinyert adatok egy **DataTable** adattípusú változóba kerülnek elmentésre.

Ennek a keresési megoldásnak nagy előnye, hogy gyors és pontos adatkinyerést tesz lehetővé. Hátránya viszont, hogy minden egyes webshopra külön implementálni kell a megoldást, ami növeli a fejlesztési időt, és a karbantartást is bonyolítja.

Általános keresési megoldás

Amikor a felhasználó olyan weboldalon próbál keresést indítani, ahol a webshop-specifikus keresési megoldás nincs implementálva, ott a robot próbál egy általános keresést és adatkinyerést megvalósítani.

Ehhez először is a megnyitott weboldalon a robotnak meg kell találnia a keresőmezőt, ahova begépel a megkeresendő termék nevét. A keresőmező azonosításához a **Find Children** activity-t használtam a **Body HTML** elemre mint szülőre a következő paraméterekkel:

```
<webctrl tag='INPUT' type='text' />
<webctrl tag='INPUT' type='search' />
```

Ez a **activity** ezzel a paraméterezéssel egy UI-elemeket tartalmazó listát ad vissza, amely a **Body** összes, **text** és **search** típusú inputmezőjét tartalmazza. Ezután a robot a **Get Attribute** activity segítségével megvizsgálja a listában található UI-elemek attribútumait. Ellenőrzi, hogy valamelyik attribútum tartalmazza-e **search**, **looking for** vagy a **keres** szavakat egyikét és így képes azonosítani a keresőmezőt. Ezt követően a **Type Into** activity használatával a robot begépel a termék nevét a keresőmezőbe, majd az **Enter** billentyű lenyomásával azonnal elindítja a keresést.

A keresési eredmény megjelenik az oldalon, és a robotnak fel kell ismernie ezeket. Ehhez ismét a **Find Children** activity-t használtam. A **Body**-ből összegyűjtöttem az összes olyan UI elemet, amelynek az **aaname** attribútuma tartalmazza a keresett termék nevét. **UiPath**-ban az **aaname** attribútum a UI eleme szöveges tartalmát azonosítja. Ha a termék több szóból áll, akkor az összes szónak szerepelnie kell az **aaname**-ben. A **Find Children** activity által visszaadott, UI elemeket tartalmazó lista minden eleméhez a robotnak meg kell találnia az árat is. Csak azokat az UI elemeket fogadtam el jó keresési találatnak a listából, ahol a robot megtalálta az árat is.

Az ár megtalálásának alapgondolata az volt, hogy az ár és a nevet tartalmazó UI elemek közös őssel rendelkeznek. Nem feltétlenül a közvetlen őst tartalmazza mindkettőt, de tapasztalataim szerint a harmadik szintű őst gyakran tartalmazza mind az ár, mind a termék nevét. Ennek az ősnak a megtalálásában a **Get Ancestor** activity segített. Ahhoz, hogy a robot megtalálja a termék árát ebben az ősből, egy custom activity-t kellett írnom amit **PriceFinder**-nek neveztem el.

Ha a robot megtalálta a termék nevét és a termék árát is, akkor azt jó keresési találatnak fogadtam el, és elmentésre kerül abba a **DataTable** típusú változóba, ahol a jó keresési eredményeket tárolom.

Ezeket a lépéseket egy ciklusban futtatom, és a robot újra és újra elvégzi az összes termékre. A végső eredmény egy olyan **DataTable**, amelyben megtalálható a keresési eredmény minden egyes termékre, minden egyes webshopból.

PriceFinder Activity [11]

Mint azt már említettem, az activity-k az automatizálás alapvető építőelemei. A beépített több száz activity mellett lehetőségünk van saját activity-k fejlesztésére, amelyeket a projektünk speciális igényeihez alakíthatunk. A UiPath-nek elérhető a Visual Studio-hoz az **Activity Creator extension**, amelynek segítségével nagyon egyszerűen létrehozhatunk új activity-eket.

Miután telepítettük az extensiót, elérhetővé válik egy új projekt típus, a **UiPath Standard Activity Project**. Egy ilyen projekt létrehozásakor egy előre elkészített activity sablon áll rendelkezésünkre, melyet igényeink szerint testre szabhatunk. Az én esetemben egy olyan activity-re volt szükségem, amely egy string típusú változót kap bemeneti paraméterként, és visszatér egy **PriceCurrencyPair** típusú objektummal.

2.9. kód. PriceCurrencyPair osztály

```
1  public class PriceCurrencyPair
2  {
3      public double Price { get; set; }
4      public string Currency { get; set; }
5
6      public PriceCurrencyPair(double price, string currency)
7      {
8          Price = price;
9          Currency = currency;
10     }
11 }
```

Az activity implementációja a **PriceFinder** osztályban történik, amely osztálynak örökölnie kell a **CodeActivity** osztályból. Ez az absztrakt osztály biztosítja az alapvető funkciókat a custom activity-k definiálásához. Ahogy az alábbi egyszerűsített kódrészletben is látható, a **PriceFinder** osztálynak két property-je van. Az egyik property az **InputArgument<string> InputText**, amely azt jelenti, hogy az activity egy stringet vár bemeneti paraméterként, a másik property pedig az **OutputArgument<PriceCurrencyPair> PriceAndCurrency**, azaz az activity majd egy **PriceCurrencyPair** típusú objektumot ad vissza a robot számára.

Az **Execute** metódus automatikusan meghívódik, amikor az activity fut, és itt ebben a metódusban hajtódnak végre a műveletek. Az első lépésben az **InputText InputArgument** értéke kerül beolvasásra és elmentésre az **inputText** változóba. Ha az **inputText** érvényes, a **Finder** privát metódus kerül meghívásra, amelyben implementáltam a keresési logikát. Ez a metódus visszaad egy **PriceCurrencyPair** objektumot, amely tartalmazza a megtalált árat és pénznemet. Ezt követően az eredmény a **PriceAndCurrency OutputArgument**-ben kerül visszaadásra, azaz ez lesz az activity által visszaadott objektum amelyet a robot használhat.

2.10. kód. PriceFinder osztály

```
1 public class PriceFinder : CodeActivity
2 {
3     [Category("Input")]
4     public InArgument<string> InputText { get; set; }
5     [Category("Output")]
6     public OutArgument<PriceCurrencyPair> PriceAndCurrency { get
7         ↪ ; set; }
8
9     protected override void Execute(CodeActivityContext context)
10    {
11        try
12        {
13            string inputText = InputText.Get(context);
14            if (string.IsNullOrEmpty(inputText))
15            {
16                throw new ArgumentException();
17            }
18
19            PriceCurrencyPair pc = Finder(inputtext);
20
21            PriceAndCurrency.Set(context, pc);
22        }
23        catch (Exception)
24        {
25            PriceAndCurrency.Set(context, null);
26        }
27    }
28    private PriceCurrencyPair Finder(string searchForPrice)
29    {
30        // Implementation of Finder logic
31        return new PriceCurrencyPair(price, currency);
32    }
```

2.3.3. Result To DB Workflow

2.3.4. Report Builder Workflow

2.4. Tesztelés

Összegzés

Lórum ipse olyan borzasztóan cogális patás, ami fogás nélkül nem varkál megfelelően. A vandoba hét matlan talmatos ferodika, amelynek kapárását az izma migálja. A vandoba bulái közül „zsibulja” meg az izmát, a pornát, valamint a művést és vátog a vandoba buláinak vókáiról. Vókája a raktil prozása két emen között. Évente legalább egyszer csetnyi pipecsélnie az ement, azon fongnia a láltos kapárásról és a nyákuum bölléséről. A vandoba ninti és az emen elé redőzi a számlan radalmakan érvést. Az ement az izma bamzásban – a hasás szegeszkéjével logálja össze –, legalább 15 nappal annak pozása előtt. Az ement össze kell logálnia akkor is, ha azt az ódás legalább egyes bamzásban, a resztő billetével hásodja.

Irodalomjegyzék

- [1] MICROSOFT: Overview of .NET Framework, <https://learn.microsoft.com/en-gb/dotnet/framework/get-started/overview>
- [2] MICROSOFT: Entity Framework Core for Beginners, <https://learn.microsoft.com/en-gb/shows/entity-framework-core-for-beginners/>
- [3] DR. KUSPER GÁBOR ÉS DR. RADVÁNYI TIBOR: Jegyzet a projekt labor című tárgyhoz, Eszterházy Károly Katolikus Egyetem, Eger, 2012.
- [4] MICROSOFT: Data binding overview (WPF .NET), <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/data/?view=netdesktop-8.0>
- [5] BENNAGE, CHRISTOPHER ÉS EISENBERG, ROB: Tanuljuk meg a WPF használatát 24 óra alatt, Kiskapu Kiadó, 2009.
- [6] AGILEVRMS: Robotic Process Automation (RPA) Life Cycle <https://www.agilevrms.com/Home/RPA>
- [7] UIPATH DOCUMENTATION: Workflow Design, <https://docs.uipath.com/studio/standalone/2023.4/user-guide/workflow-design>
- [8] IXENIT BLOG: Betekintés a UiPath RPA megoldásába, <https://blog.ixenit.com/hu/betekintes-a-uipath-rpa-megoldasaba>
- [9] DOCS.UIPATH.COM: Data Table Variables, <https://docs.uipath.com/studio/standalone/2021.10/user-guide/data-table-variables>
- [10] DOCS.UIPATH.COM: Table Extraction, <https://docs.uipath.com/activities/other/latest/ui-automation/table-extraction>
- [11] DOCS.UIPATH.COM: Using The Activity Creator, <https://docs.uipath.com/sdk/other/latest/developer-guide/using-activity-creator>

Nyilatkozat

Alulírott, büntetőjogi felelősségem tudatában kijelentem, hogy az általam benyújtott, című szakdolgozat önálló szellemi termékem. Amennyiben mások munkáját felhasználtam, azokra megfelelően hivatkozom, beleértve a nyomtatott és az internetes forrásokat is.

Aláírással igazolom, hogy az elektronikusan feltöltött és a papíralapú szakdolgozatom formai és tartalmi szempontból mindenben megegyezik.

Eger, 2021. szeptember 25.

aláírás

**A *Nyilatkozatot* kitöltve nyomtassa ki, írja alá,
majd szkennelve tegye ennek a helyére!**