



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

**OE-NIK
2022**

Hallgató neve:
Hallgató törzskönyvi száma:

**Kun Szabolcs István
T/006632/FI12904/N**

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Kun Szabolcs István**
Törzskönyvi száma: T/006632/FI12904/N
Neptun kódja: 

A dolgozat címe:

**Adatvezérelt folyamatos ügyfél-elégedettségi platform fejlesztése és
egységesítése az üzleti döntéshozatal támogatásához
a Nokia Solutions and Networks Kft.-nél
Developing a consolidated data-driven continuous customer feedback
platform to assist business decision-makers
at Nokia Solutions and Networks Kft.**

Intézményi konzulens: Dr. Nagy Enikő
Külső konzulens: Jambricska Éva
IT Advanced Analytics Specialist - NOKIA

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Számítógép architektúrák
Big Data és üzleti intelligencia
specializáció

A feladat

A szakdolgozat kapcsán a hallgató feladata egy olyan riportalkalmazás-együttes fejlesztése, amely támogatja a Nokia Solutions and Networks Kft. „customer feedback driven” azaz ügyfél-elégedettségen alapuló üzleti stratégiáját. Az alkalmazásnak különböző formulák segítségével értékmutatókat és kulcsindikátorokat kell tudnia előállítania és vizualizálnia a vállalat szervezeti egységei számára. Az alkalmazás tulajdonosainak pedig olyan adminisztrációs felületet kell biztosítani, ami lehetővé teszi a különböző régiók alkalmazotti hozzáféréseinek szűkítését vagy lazítását. A hallgató feladata a projekt kapcsán, hogy megtervezze az architektúrát, részt vegyen az adatpiacok kialakításában és ETL folyamatok segítségével megvalósítsa az adatátemeléseket, és transzformációkat, ezen felül pedig előállítsa a különböző metrikákat. További feladata, hogy kiszolgáló-nélküli (serverless) és backend alkalmazásokat fejlesszen formula specifikus metrikák előállításához, Microsoft Graph integrációhoz és a hozzáférés menedzsment biztosításához. Az elkészült metrikákból pedig külön adatpiacokat állítson elő a Power BI adatvizualizációs réteg számára.

A dolgozatnak tartalmaznia kell:

- az üzleti igény rövid összefoglalását,
- az architektúra tervezés különböző aspektusait, folyamatát és a végeredményt,
- az adatbázis tervezését és megvalósítását,
- az ETL folyamatok megtervezését és megalkotását,
- az adatpiacok kialakításának módját,
- a backend és kiszolgáló-nélküli alkalmazások tervezését és fejlesztését,
- és a BI alkalmazás integrációjának megvalósítását.



Fleiner Rita
.....
Dr. Fleiner Rita
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

Sandorcska Rita
.....
külső konzulens

Dei
.....
intézményi konzulens

Titoktartási megállapodás

amely létrejött egyrészről a

Nokia Solutions and Networks Kft.

Székhelye: H-1083 Budapest, Bókay János u. 36-42.
Cégjegyzékszám: **01-09-875759**
Adószám: **13835400-2-44**
Képviseli: **László Linda (Country HR Manager)**
(a továbbiakban: **Társaság**),

másrészről a

Óbudai Egyetem

Székhelye: 1034 Budapest, Bécsi út 96/b
Intézményi azonosító: OM-FI12904
Adószáma: 15773063-2-41
Képviseli: Dr. Póser Valéria
(a továbbiakban: **Egyetem**)

együttesen Felek (a továbbiakban: **Felek**)

között az alább jelzett napon és helyen, a következő határidőkhöz és feltételekhez igazodva.

1. Előzmények

- 1.1. Kun Szabolcs István (tanulmányi azonosító: XOUEFI) (a továbbiakban: Hallgató), az Óbudai Egyetem Neumann János Informatikai Karának BSc szakos hallgatója a „**Adatvezérelt folyamatos ügyfél-elégedettségi platform fejlesztése és egységesítése az üzleti döntéshozatal támogatásához a Nokia Solutions and Networks Kft.-nél**” című szakdolgozatában (továbbiakban: Szakdolgozat) olyan bizalmasnak minősülő információkat, adatokat rögzít, amelyek zártkörű kezelése (a továbbiakban: titkosítása) indokolt a Társaság üzleti érdekeinek védelme érdekében.
- 1.2. Felek rögzítik, hogy az Óbudai Egyetem Tanulmányi és Vizsgaszabályzatának és Tanulmányi Ügyrend szabályzatának (a továbbiakban együtt: Szabályzat), megfelelően a Hallgató és a Társaság kérelmezték a Szakdolgozat titkosítását.

2. A megállapodás tárgya

- 2.1. Az Egyetem jelen megállapodás aláírásával tudomásul veszi, hogy a Szakdolgozatban szereplő, a Szakdolgozattal összefüggésben a Társaságtól származó személyes, informatikai, üzleti, műszaki és egyéb információ (a továbbiakban együttesen: Információ) bizalmas természetű és a Társaság tulajdonát képezi. Felek jelen megállapodás aláírásával tudomásul veszik és kijelentik, hogy az elkészült Szakdolgozat és mellékletei jelen titoktartási megállapodás hatálya alá tartoznak.

3. Felek jogai és kötelezettségei

- 3.1. A 2.1. pontban foglaltakra tekintettel az Egyetem a Szakdolgozatot a titoktartás szabályai és a Szabályzatban foglaltak szerint köteles kezelni. Az Egyetem vállalja, hogy a Szakdolgozatot üzleti titokként kezeli, azt a szerződés időtartama alatt harmadik személyek tudomására nem hozza, illetve semmilyen módon nem teszi hozzáférhetővé a Társaság előzetes írásbeli engedélye nélkül.
- 3.2. Társaság kijelenti, hogy az Óbudai Egyetem Szabályzatának a titkosított szakdolgozatok kezelésének eljárási rendjében foglaltakat megismerte és az abban foglalt rendelkezéseket a jelen megállapodás alkalmazásának vonatkozásában elfogadja.
- 3.3. Az Egyetem vállalja, hogy a Szabályzat szerinti eljárásrend alkalmazásával kerül sor a titkosított Szakdolgozat megvédésére. A 3.1. pontban előírt titoktartási kötelezettség alól kivételt képez a Szakdolgozat megvédésének folyamata, ahol az értékelő bizottság tagjai – előzetes titoktartási nyilatkozat aláírását követően – megismerik a Szakdolgozat tartalmát.
- 3.4. Az Egyetem kötelezettséget vállal arra, hogy kizárólag azok a közalkalmazottai ismerhetik meg a Szakdolgozat tartalmát, akik tekintetében ez feltétlenül szükséges a Szakdolgozat intézményen belüli, szabályos kezelése céljából. Az Egyetem köteles gondoskodni arról, hogy a titoktartási kötelezettség kiterjed az Egyetem közalkalmazottaira.
- 3.5. Társaság tudomásul veszi, hogy a Szakdolgozat elektronikus formátumú változata a Szabályzat értelmében feltöltésre kerül az intézményi könyvtárakba, ahol lehetőség van a titkosításhoz szükséges elérési jogosultsági szint beállítására.
- 3.6. Felek megállapodnak, hogy a titkosított Szakdolgozat kapcsán az alábbi adatok nyilvánosak, azaz nem képezik jelen megállapodás tárgyát:
 - a) a szakdolgozat címe, a szerző és témavezető neve és a védés időpontja
 - b) a titkosítás ténye és a titkosítás határidejének várható lejárta.
- 3.7. Felek megállapodnak, hogy jelen megállapodás időtartama az aláírásának napjától számított **5 év** időtartamra szól. Ennek megfelelően a Szakdolgozatra vonatkozóan az Egyetem titoktartási kötelezettsége **5 év** elteltével megszűnik és Társaság hozzájárul, hogy ezen idő leteltét követően a Szakdolgozat nyilvánosságra kerüljön.
- 3.8. Felek rögzítik, hogy az üzleti titok védelméről szóló 2018. évi LIV. törvény alapján az üzleti titok akkor is törvényes védelem alatt áll, ha külön szerzői jogi védelemben, szabadalmi oltalomban, használati mintaoltalomban vagy egyéb, jogszabályokban meghatározott, szellemi alkotásokra vonatkozó jogi védelemben nem részesül.
- 3.9. Felek rögzítik, hogy a jelen megállapodás szerinti titoktartási kötelezettség nem érvényesíthető államigazgatási (így különösen adóügyi) és bírósági eljárásban, továbbá azokban az esetekben, amikor jogszabály írja elő, hogy az információt a jogszabályban megjelölt személlyel közölni kell (pl. közérdekű, vagy közérdekből nyilvános adatok közzétevése), ezért ezekre nézve kölcsönösen és előzetesen mentesítik egymást a titoktartási kötelezettség alól, azzal a feltétellel, hogy Felek kötelesek előzetesen értesíteni egymást a jogszabályi kötelezettségről, illetve az eljárások tényéről és jogszabály alapján, illetve az eljárás során átadandó információk mértékéről.
- 3.10. Nem állapítható meg titoktartási kötelezettség továbbá az információk következő csoportjára:
 - (i) azon információ, amely nem az Egyetem hibájából vagy szerződésszegésének következtében került nyilvánosságra;

(ii) azon információ, amely a Társaság tudtán kívül már az átadás időpontjában közismert vagy bárki számára megismerhető volt;

(iii) azon információ, amely olyan személy által jutott nyilvánosságra, akiért Felek nem felelnek. Felek a következő kapcsolattartó személyeken keresztül értesítik egymást jelen szerződéssel kapcsolatosan.

Az Egyetem részéről: Oktatási dékánhelyettes

Név, beosztás: Dr. Póser Valéria

Telefon: 0616665585

E-mail: poser.valeria@nik.uni-obuda.hu

A Társaság részéről:

Név, beosztás: László Linda (Country HR Manager)

Telefon: [REDACTED]

E-mail: [REDACTED]

Felek a kapcsolattartók személyében bekövetkezett változásokról kötelesek egymást haladéktalanul, de legfeljebb három munkanapon belül írásban tájékoztatni.

4. Egyéb rendelkezések

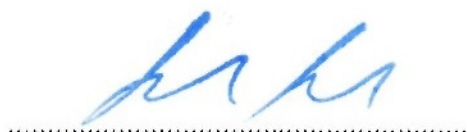
4.1. Felek a jelen megállapodásból eredő vitás kérdéseket elsődlegesen tárgyalás útján egymás között rendezik, amennyiben ez 60 napon belül nem vezet eredményre, úgy Felek a Polgári Perrendtartásról szóló törvény mindenkorai szabályai szerint járnak el.

4.2. A jelen megállapodásban nem szabályozott kérdések tekintetében a Ptk. rendelkezései, a hatályos magyar jogszabályok és a Szabályzat rendelkezései az irányadók.

4.3. Jelen megállapodás 4 (azaz négy) egymással szó szerint egyező eredeti példányban készült.

Felek jelen szerződést, mint akaratukkal mindenben megegyezőt, jóváhagyólag és saját kezűleg irták alá.

A Nokia Solutions and Networks Kft.
(Társaság) képviseletében:



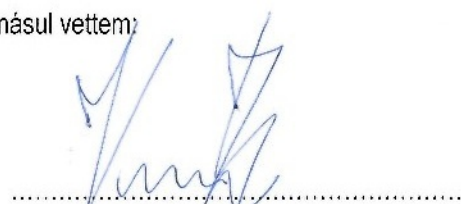
László Linda
Country HR Manager

Az Óbudai Egyetem
képviseletében:



A fenti megállapodásban foglaltakat megismertem és tudomásul vettem;

Budapest, 2021.11.16.



Kun Szabolcs István (hallgató)

Nokia Solutions and Networks Kft.
H-1083 Budapest, Bókay János u. 36-42.
Adószám: 13835400-2-44
Cégjegyzékszám: 01-09-875759



HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem.

Budapest, 2022.05.09.

.....
hallgató aláírása



KONZULTÁCIÓS NAPLÓ

Hallgató neve:

Kun Szabolcs István

Neptun kód:

[REDACTED]

Tagozat:

Mérnökinformatikus BSc

Telefon:

Levelezési cím (pl: lakcím):

[REDACTED]

Szakdolgozat címe magyarul:

Adatvezérelt folyamatos ügyfél-elégedettségi platform fejlesztése és egységesítése az üzleti döntéshozatal támogatásához

Szakdolgozat címe angolul:

Developing a consolidated data-driven continuous customer feedback platform to assist business decision-makers

Intézményi konzulens:

Dr. Nagy Enikő

Külső konzulens:

Jambricska Éva
IT Advanced Analytics Specialist - NOKIA

Alk.	Dátum	Tartalom	Aláírás
1.	2021.09.28.	Szakdolgozat témájának megbeszélése	
2.	2021.10.19.	Feladatlap megbeszélése	
3.	2021.11.30.	Irodalomkutatás megbeszélése	
4.	2021.12.07.	Dokumentumok beadásának megbeszélése	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. (BSc) tantárgy követelményét teljesítette, beszámolóra bocsátható.

Budapest, 2021.12.07.

intézményi konzulens



KONZULTÁCIÓS NAPLÓ

Hallgató neve:

Kun Szabolcs István

Neptun kód:



Tagozat:

Mérnökinformatikus BSc

Telefon:

Levelezési cím (pl: lakcím):



Szakdolgozat címe magyarul:

Adatvezérelt folyamatos ügyfél-elégedettségi platform fejlesztése és egységesítése az üzleti döntéshozatal támogatásához

Szakdolgozat címe angolul:

Developing a consolidated data-driven continuous customer feedback platform to assist business decision-makers

Intézményi konzulens:

Dr. Nagy Enikő

Külső konzulens:

Jambricska Éva
IT Advanced Analytics Specialist – NOKIA

Alk.	Dátum	Tartalom	Aláírás
1.	2022.02.22.	Irodalomkutatás véglegesítése	
2.	2022.03.28.	Funkcionális specifikáció és rendszerterv	
3.	2022.04.12.	Implementáció és eredmények összegzése	
4.	2022.05.09.	Dokumentumok beadásának megbeszélése	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat II. (BSc) tantárgy követelményét teljesítette, védésre bocsátható.

Budapest, 2022.05.09.

intézményi konzulens

TARTALOMJEGYZÉK

1.	Bevezetés	13
2.	A projekt bemutatása	14
2.1.	Ügyfél hangja platform	14
2.2.	Ügyfél által érzékelt érték	15
2.3.	Üzleti igények	16
2.4.	A projekt szervezése	17
3.	Felhőalapú számítástechnika	18
3.1.	Felhőalapú számítástechnika definiálása [2]	18
3.2.	Szolgáltatási modellek	19
3.3.	Költségek kezelése	20
3.4.	Felhő vagy lokális infrastruktúra?	20
3.5.	Alkalmazás portfólió realizálása és migráció	22
3.6.	Szolgáltatói függőség kérdése	25
4.	Funkcionális specifikáció	26
4.1.	Felhasználói felületek	26
4.2.	Felhasználói profilok	26
4.3.	Forrásadatok	27
4.4.	Adatok töltése és feldolgozása	27
4.5.	Számításokat végző komponens	27
4.6.	Monitorozás	27
5.	Nem funkcionális követelmények	28
5.1.	Performancia	28
5.2.	Biztonság	28
5.3.	Karbantarthatóság	28
5.3.1.	Infrastruktúra	28
5.3.2.	Kódbázis	29
5.4.	DevOps	29
5.5.	Ügyféltámogatás	30
5.6.	Rendelkezésre állás	30
6.	Megközelítési módok és megoldási lehetőségek áttekintése	31

6.1.	Microsoft Azure specifikus szolgáltatások kiválasztása	31
6.1.1.	ETL eszköz megválasztása	32
6.1.2.	Számítási szolgáltatások megválasztása	33
6.1.3.	Adatbázis szolgáltatás megválasztása.....	36
6.1.4.	BI vizualizációs eszköz megválasztása.....	36
7.	Architektúra tervezés	37
7.1.	Az infrastruktúra architektúrája	37
	A rendszerterv ábrájának magyarázata	38
7.2.	Az adatbázis architektúrája	39
8.	Implementáció folyamata	41
8.1.	Felhős hoszting környezetek provizionálása	41
8.2.	Kódbázis	41
8.3.	A CPV kalkulációk alapelvei	43
8.4.	Adattárház modul implementálása.....	43
8.5.	Adatcsatornák implementálása	47
8.5.1.	A PL_DATA_CPV_BASE adatcsatorna megvalósítása	47
8.5.2.	A PL_DATA_APP_BIAS_CPV_CALCULATION adatcsatorna megvalósítása.....	50
8.5.3.	A PL_DATA_APP_CPVI_CALCULATION adatcsatorna megvalósítása	51
8.5.4.	A PL_DATA_CPV_AAD_TO_DB_SYNC pipeline megvalósítása	51
8.5.5.	A PL_DATA_CPV_DB_TO_AAD_SYNC adatcsatorna megvalósítása	52
8.6.	Adatfolyamok implementálása	52
8.6.1.	A DF_DATA_CPV_COMPETITOR adatfolyam megvalósítása	53
8.6.2.	A DF_CPV_CUSTOMER_AND_REGION_LEVEL_ACCESS_MANAGEMENT adatfolyam megvalósítása	54
8.6.3.	A DF_DATA_CPV_SURVEY_RESPONDENT_NAME adatfolyam megvalósítása.....	55
8.6.4.	A DF_DATA_CPV_SURVEY_RESPOND adatfolyam megvalósítása .	56
8.6.5.	A DF_DATA_CPV_CALCULATION_CONTROL_BIAS adatfolyam megvalósítása.....	57
8.6.6.	A DF_DATA_CPV_CALCULATION_CONTROL_CPV adatfolyam megvalósítása.....	57

8.6.7.	A DF_DATA_CPV_SURVEY_RESPOND_CALCULATION adatfolyam megvalósítása.....	59
8.6.8.	A DF_DATA_CPV_CUSTOMER_CALCULATION adatfolyam megvalósítása.....	59
8.7.	A számításokat végző és a címtár szinkronizációs modulok implementálása	61
8.7.1.	A BIAS kalkuláció megvalósítása	61
8.7.2.	Címtár szinkronizáció	62
8.7.3.	Címtárból az adatbázisba történő szinkronizáció megvalósítása.....	63
8.8.	Hozzáférés menedzsment modul implementálása	65
8.8.1.	Végpontok.....	66
8.8.2.	A GetAllUsers adatszolgáltató végpont megvalósítása	66
8.8.3.	Az AddUsersToGroup adatmanipulációs végpont megvalósítása	67
8.8.4.	Autentikáció és autorizáció.....	68
8.8.5.	Frontend.....	69
8.9.	BI integráció	69
9.	Tesztelés.....	71
10.	Elért eredmények	72
11.	Továbbfejlesztési lehetőségek.....	73
12.	Összegzés.....	74
13.	Summary.....	75
14.	Köszönetnyilvánítás.....	76
15.	Rövidítések	77
16.	Irodalomjegyzék	79
17.	Mellékletek	81
18.	Ábrajegyzék.....	82

1. BEVEZETÉS

A Nokia számára mindig is fontos volt, hogy nagyvállalati ügyfelei számára innovatív, megbízható, és iparágvezető telekommunikációs, illetve IoT megoldásokat biztosítson. Elengedhetetlen, hogy tisztában legyünk azzal, hogy ügyfeleink mit gondolnak rólunk, hogyan értékelik a múltbéli teljesítményünket és miként látják a közös jövőt. Stratégiánk részévé vált, hogy működésünket ügyfeleink visszajelzéseire alapozzuk, és ezek mentén szervezzük folyamatainkat a termék fejlesztéstől a gyártáson át az értékesítésig. A siker eléréséhez pedig egy rögzös út vezet, mely nem csak stratégia- és kultúraváltást igényel, hanem ezen felül jól megtervezett, belsős IT rendszerek kiépítését is megköveteli.

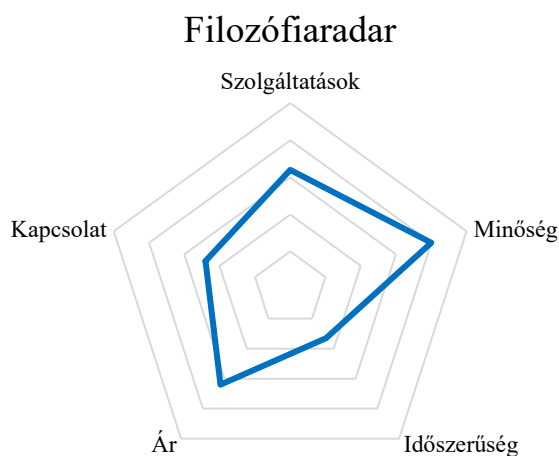
Szakedolgozatomban a Nokia Voice of Customer („ügyfél hangja”) stratégiájának Customer Perceived Value („ügyfél által érzékelt érték” - CPV) alkalmazásáról, annak újratervezéséről és lefejlesztéséről értekezem. Górcső alá kerül továbbá, hogy milyen kihívásokkal járt e projekt külsős beszállító által fejlesztett alkalmazásának beszerzése (insourcing) és az új alkalmazások architektúráinak tervezése és fejlesztése.

2. A PROJEKT BEMUTATÁSA

Ebben a fejezet a projekt tárgyáról és az elvárt követelményekről fogok beszélni. Részletezem az üzleti igényeket, illetve hogy miért fontos a Nokia számára, hogy az ügyfelek hangot adhassanak a véleményeiknek, továbbá milyen üzleti értékeket tartanak fontosnak a jelenlegi piaci helyzetet tekintve.

2.1. Ügyfél hangja platform

Általánosságban véve üzleti évenként mérjük ügyfeleink elégedettségét, hogy megértsük, mit gondolnak az általunk nyújtott szolgáltatásokról, azok minőségéről, időszerűségükről és árainkról, valamint a velük kialakított általános üzleti kapcsolatunkról. (lásd: 1. ábra) Célunk, hogy a számukra fontos dolgokra összpontosítsunk, ezért az ügyfelek értékeléseit két irányban mérjük - a múltbéli tapasztalatok és jövőbeli elvárások irányából. Fontos hangsúlyozni, hogy értéket mérünk, és nem pedig azt, hogy egyes ügyfeleink mennyire lojálisak a Nokiához. (Értékek alatt olyan mérhető paraméterekre gondolunk, mint például üzleti teljesítmény, árazás, ügyféltámogatás minősége stb.) Tehát a Voice of Customer platform célja, hogy különböző módszerekkel gyűjtsön adatokat ügyfél visszajelzések mentén, és ezekből értékes KPI-okat¹ állítson elő különböző üzletágak számára.



1. ábra: A Nokia értékeinek filozófiaradarja (Forrás: saját szerkesztés)

¹ A KPI a key performance indicators angol kifejezés rövidítése, melynek jelentése kulcs hatékonysági mutató.

A platform több különböző célalkalmazást és alrendszert foglal magába, mint például kérdőív rendszerek, kimutatások vizualizációjára szolgáló alkalmazások, elemző és eset-virtualizációs rendszerek. Az említett célalkalmazások és alrendszerek összessége biztosítja az alkalmazottak számára, hogy rövid idő alatt juthassanak közel valós idejű információhoz, és ezek alapján tudjanak meghozni üzletkritikus döntéseket, mint például termék árak módosítása, reklámkampányok bevezetése, értékesítési stratégiák változtatása, stb.

2.2. Ügyfél által érzékelt érték

A Customer Perceived Value alkalmazás a Voice of Customer platform magja. Feladata, hogy adatot emeljen át az ügyfél visszajelzéseit kezelő alrendszerből (Customer Experience Survey - CES), majd pedig ebből statisztikákat és KPI-okat állítson elő, ehhez különböző múltbéli értékeket és formulákat felhasználva. Feladata továbbá az is, hogy biztosítsa ezen információk vizualizációját és elérhetőségét.

Az ügyfelek elégedettségét általában évente egyszer mérjük kérdőívek segítségével. Ez alapján ezután a CPV értékeket minden ügyfél entitásra és vállalatra vonatkozóan évente újra kalkuláljuk, mikor új kitöltések érkeznek az ügyféltől a CES alrendszeren keresztül. Ilyenkor nyolc személyt keresünk meg a szervezettől, hogy válaszoljon egy sor kérdésre, melyekre egy és tíz közötti számot tud válaszul adni. A CPV-t a négy kérdésre adott 1 és 10 közötti pontszámokból vezetjük le. Fontos megemlíteni, hogy a kérdőívek során törekszünk arra, hogy ügyfél-elégedettséget mérjünk és nem pedig lojalitást², habár a kérdőív kérdései részben hasonlítanak a hagyományos értelemben vett ügyfél lojalitás programok során megszokott kérdésekhez.

A teljesség igénye nélkül az alábbi kardinális kérdésekre kaphatunk választ az alkalmazás segítségével:

- Hogyan rangsorolják ügyfeleink a Nokiát és a többi versenytársat ár, innováció és ügyfélélmény szempontjából?
- Mely földrajzi régiókban tölt be piacvezető szerepet a Nokia?
- Hogyan alakultak az ügyfélelégedettség mutatók különböző idő intervallumokat tekintve?

² A lojalitás definíciója a fogyasztás értelmében: "A lojalitást, mint az újrávásárlás pillanatához viszonyított ex-ante állapotot értelmezem, mely állapotban szociális, egyéni, és termékhez kapcsolható tényezők hatásainak együttes eredménye a vizsgált termék áldozathozatallal járó újrávásárlásához vezet." [18]

Számos alkalmazást, mint ezt is, a Voice of Customer platformon belül egy külsős beszállító cég fejlesztette le, és üzemelteti a Nokia számára 2006 óta.

2.3. Üzleti igények

A nagyvállalatok többsége a profit maximalizálásának érdekében és a folyamatosan változó piaci helyzetek miatt rákényszerül arra, hogy folyamatos stratégiaváltozásokat eszközöljenek és törekedjenek folyamataik optimalizálására. Ez alól a Nokia sem kivétel, jó példa erre a Microsoft és a Nokia között létrejött stratégia-szerződés 2015-ben, melynek célja, hogy a Nokiában 2017-ig bevezetésre kerüljenek az Office 365 és Microsoft Azure szolgáltatások. A bevezetést követően folyamatosan törekszünk arra, hogy meglévő belső alkalmazásainkat és termékeinket a felhőbe migráljuk, míg az újakat felhő natív alkalmazásként fejlesszük le.

A felhősödés tendenciája mellett 2019-ben megjelent egy új irányzat a vállalaton belül, amely arra törekszik, hogy az egyes beszállítókra kiszervezett belső alkalmazások fejlesztését és üzemeltetését visszaszervezze vállalaton belülről a szakmai tudással együtt. Egy esetleges visszaszervezés nem csak csökkenti a függőséget esetleges külső entitásoktól, hanem visszaadja a teljeskörű rendelkezést a vállalat saját alkalmazásai felett, illetve ezzel együtt a szaktudást is házon belülről helyezi.

A legtöbb multinacionális vállalat életében jelen van egy „insourcing-outsourcing hullám”. Ez reprezentálja, hogy egy vállalat stratégiája jelenleg kiszervezésen vagy beszerzésen alapszik. Egy ilyen hullám hossza és amplitúdója attól függ, hogy a jelenlegi költségvetés mennyire követeli meg az esetleges be- illetve kiszervezést. [1]

A fent említett két irányzat alapján született meg az üzleti igény arra, hogy a jelenlegi külsős beszállító cég által fejlesztett CPV alkalmazás beszerzésre kerüljön, és a belső IT szervezet teljesen új alapokra helyezve felhő natív alkalmazásként fejlessze le.

Az újradolgozott alkalmazásnak nem csak az eredeti feladatait kell tudnia ellátni, hanem ezen felül további követelményeknek is eleget kell tennie, a teljesség igénye nélkül:

- Az alkalmazásnak illeszkednie kell a „Nokia Enterprise Application Landscape” arculatába, ami annyit tesz, hogy olyan eszköztárral kell elkészülnie az alkalmazásnak, amely segítségével végeredményben képes elérni a landscape (alkalmazás portfólió) alkalmazásainak színvonalát és az összképet nem befolyásolja negatívan.

- Minden komponense felhő alapú legyen, és ne használja egyik Nokia Data Center-t (Private Cloud³) sem. Ha mégis elkerülhetetlen valamelyik Nokia Data Center használata, akkor dekompozícióra és a függőségek lazítására kell törekedni.
- Az alkalmazást csak intranetről lehessen elérni és a felhasználók autentikációjához Nokia Single Sign-On⁴ interfészt, pontosabban Azure Active Directory-t használjon.
- Az alkalmazás front-end komponensei mind webes alapúak legyenek és megfeleljenek a legújabb webes tendenciáknak.
- Az adatvizualizációhoz Microsoft Power BI kerüljön felhasználásra.

2.4. A projekt szervezése

A projekt megrendelését követően egy szakértői csapat került felállításra, ami 6 főből tevődött össze. A projekt követelményeinek követéséért, a feladatok szervezéséért és a határidők betartatásért a projekt menedzser és terméktulajdonos volt felelős. A fejlesztésen 5 mérnök dolgozott, akik különböző területre szakosodtak: egy Power BI fejlesztő, egy frontend fejlesztő, egy adattárház és adatmodell fejlesztő, és jómagam adattárház, adatmodell és backend fejlesztőként.

A projekt során részt vettem az adatmodell és adattárház megtervezésében és fejlesztésében, ezentúl a backend és démon⁵ alkalmazások tervezése illetve fejlesztése teljesen az én hatáskörömbe tartoztak.

³ Private cloud alatt a privát felhős környezetet értjük, mikor egy szervezet saját felhőt épített ki és üzemeltet egy beszállító által.

⁴ A Single Sign-On (SSO) egy webes egyszeri bejelentkezést lehetővé tevő architektúráis minta, melynek célja, hogy a felhasználó számára leegyszerűsítse egy célalkalmazásba vagy rendszerbe történő belépéskor az azonosítást. Az adott alkalmazásban mindössze csak egyszer kell elvégeznie az azonosítást, majd a későbbiekben pedig automatikusan megtörténik az azonosítás más SSO-t használó alkalmazásokban.

⁵ Daemon („démon”) alkalmazás alatt olyan szerveres környezetben üzemelő számítógépes programot értünk, mely háttérfolyamatként fut, nem pedig egy interaktív felhasználó közvetlen irányításától függ. Ezen alkalmazások bizonyos események, illetve időzítés alapján futtatják le a logikáikat.

3. FELHŐALAPÚ SZÁMÍTÁSTECHNIKA

A projekt egyik követelménye, hogy az alkalmazások komponensei felhőalapú megoldásokat használjanak, továbbá a projekt célja a jelenlegi alkalmazás beszerzése és modernizálása. Első lépésként elmélyültem a rendelkezésre álló szakirodalomban a felhőalapú számítástechnika területén, hogy a legjobb döntést hozhassam majd meg az architektúra tervezése során. Részletezem majd a szolgáltatási modellek tulajdonságait, költségkezelést, infrastruktúra környezetek típusait. Továbbá górcső alá veszem a különböző migrációs stratégiákat és a piacvezető felhőszolgáltatóit.

3.1. Felhőalapú számítástechnika definiálása [2]

A felhőalapú számítástechnika egy olyan modell, amely lehetővé teszi a mindenütt elérhető, kényelmes, igény alapú hálózati hozzáférést a konfigurálható számítástechnikai erőforrások megosztott állományához (pl. hálózatok, szerverek, tárolók, alkalmazások és egyéb szolgáltatások). Ezen kívül ez a modell gyorsan igénybevehető és felszabadítható, minimális kezelési erőfeszítéssel vagy szolgáltatói interakcióval.

A National Institute of Standards and Technology öt alapvető jellemzőt fogalmaz meg a modellel kapcsolatban:

- **Igény szerinti önkiszolgálás:** Lényege, hogy az ügyfél önkiszolgáló módon képes szolgáltatásokat létrehozni anélkül, hogy bármilyen interakció lenne szükséges a szolgáltatóval, illetve emberi erőforrást venne igénybe.
- **Hálózati hozzáférés:** Az erőforrások és egyéb szolgáltatások hálózaton keresztül érhetők el szabványos protollokon keresztül.
- **Megosztott erőforrások:** A szolgáltató által biztosított erőforrások nem feltétlenül dedikált szerverek segítségével, hanem megosztott erőforrás csoportokkal szolgálják ki a számítási feladatokat a terhelésnek megfelelően. Ebből a csoportból a szolgáltató több egymástól független fogyasztó számára is kioszthat erőforrásokat. Ez egy több bérlőre épülő modell, amely olcsóbb számítási kapacitást tesz elérhetővé.
- **Gyors és rugalmas skálázhatóság:** Az igényeknek megfelelően az erőforrások könnyedén és gyorsan skálázhatóak kétirányúan, horizontálisan és vertikálisan. A skálázhatóság megvalósítható automatikusan és/vagy manuálisan.
- **Mérhető szolgáltatás:** A felhasználó által generált fogyasztás mérésre kerül és ezek mentén a felhasználó pontosan annyit fizet amennyi szolgáltatást igénybe vett.

3.2. Szolgáltatási modellek

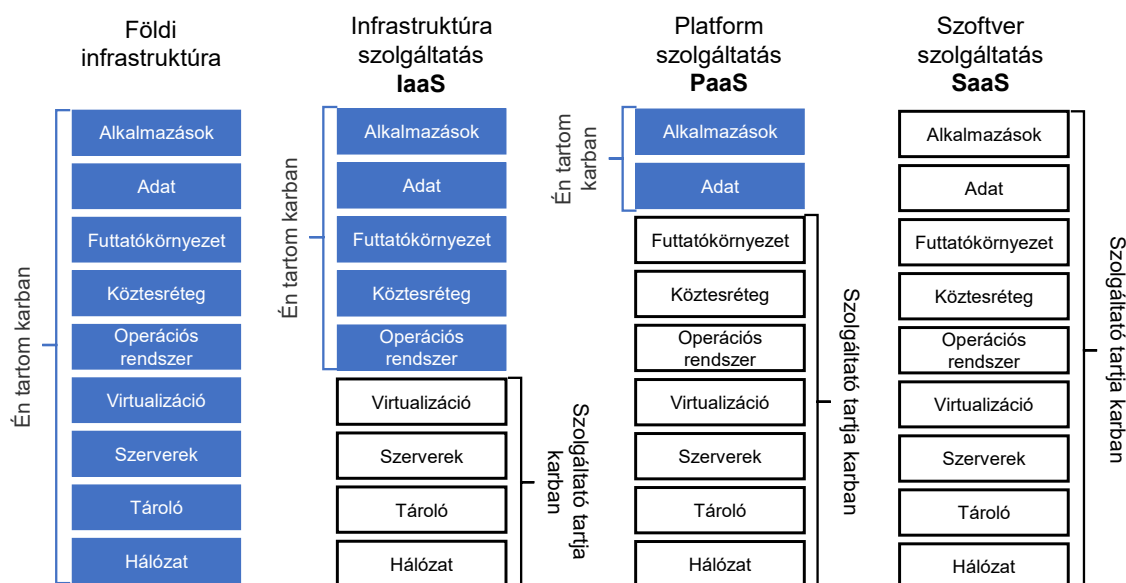
A jelenlegi alkalmazás hagyományos, földi szerveres környezetben fut, valamint célunk, hogy felhő natív alkalmazásként kerüljön lefejlesztésre a modernizált változata. Éppen ezért az ideális szolgáltatások megválasztásához elengedhetetlen, hogy megvizsgáljam a rendelkezésre álló szolgáltatási modelleket. A megfelelő szolgáltatási modellek megválasztásával nem csak költséget takaríthatunk meg, hanem időt és ráfordított munkaórát is a különböző modellek eltérő üzemeltetési igényeinek köszönhetően.

Ahhoz, hogy a projekt megvalósítása során a legjobb döntést hozhassam majd meg, alább nem csak elemzem a szolgáltatásokat, hanem összehasonlításukat is megteszem.

Felhős környezetben három alap modellt tudunk megkülönböztetni:

- az infrastruktúra szolgáltatásokat (Infrastructure as a Service – IaaS),
- a platform szolgáltatásokat (Platform as a Service – PaaS),
- illetve a szoftver szolgáltatásokat (Software as a Service – SaaS).

A modellek közti különbségeket jól szemlélteti a 2. ábra. A lényegi különbség a felelősségi szintekben mutatkozik meg: míg az infrastruktúraszolgáltatásoknál én felelek a szoftverek licenszeléséért és egyéb szoftveres karbantartásokért, addig a platformszolgáltatások esetében ezt a szolgáltató vállalja át. Szoftver szolgáltatásoknál mindent a gyártó szállít, a szoftvertől a futtatókörnyezeten át a hardverig.



2. ábra: A szolgáltatás modellek összehasonlítása felelősség szempontjából (Forrás: saját szerkesztés)

3.3. Költségek kezelése

A különböző kiadások szabályozása minden nagyvállalat költségoptimalizálásának sarokköve. Egy-egy nagyobb beruházás megvalósítása különböző kihívásokat gördíthet a vállalkozások pénzügyi vezetői elé, hiszen egy nagyobb mértékű beruházás könnyen felboríthatja az adott üzleti év pénzáramlását. Ezen kívül nagyban befolyásolja a vállalkozás tőkehelyzetét és más innovációs befektetések megvalósulását. Tőkebefektetésről (CAPEX – capital expenditure) beszélünk akkor, mikor egy hagyományos IT on-premise szerverparkot tervezünk kiépíteni vagy egy új telephelyet (irodát) vásárlunk a vállalkozásunk számára. Ez jelentős kiadásokat vonhat maga után még azelőtt, hogy egyáltalán használatba tudnánk venni a szervereinket és egyéb informatikai eszközeinket. Ezen kívül nem elhanyagolhatók a később fellépő költségek sem, mint például a szerverek üzemeltetéshez szükséges közüzemi fogyasztásból keletkező költségek, szakképzett munkaerő fenttartása, és így tovább. [3]

Mint ahogy feljebb is részleteztem a szolgáltatási modellek során, ahogy egyre jobban átruházzuk az üzemeltetés felelősségét a szolgáltatóra, úgy a mellékes költségek kezelésének felelőssége is egyre jobban átkerül hozzá. Ezáltal hónap végén a fogyasztó csak egy számlát kap a használt erőforrások után, amit működési költségként (OPEX – operational expenditure) tud elszámolni. Erre jó példa az a helyzet, mikor egy cég kibérel egy irodát a vállalkozása számára. Ezáltal a fogyasztónak csak egy költsége keletkezik, míg feljebb nem csak tőkebefektetéssel, hanem belső működéssel kapcsolatos költségek könyvelésével is számolnia kell. [3]

Ezek alapján a felhőalapú megoldások használata és ilyesfajta infrastruktúrák kiépítése jelentősen költséghatékonyabbnak minősülhet hagyományos módszerekkel szemben. Nem csak az üzemeltetés kihívásait csökkenti, de tovább könnyíti a költségoptimalizálást is, hogy szükségleteinkhez képest valós időben méretezhessük fogyasztásunkat.

3.4. Felhő vagy lokális infrastruktúra?

Annak ellenére, hogy a projekt elköteleződött a felhős megoldások használata mellett, mégis fontosnak tartom, hogy megvizsgáljam a rendelkezésre álló hosztolási opciókat, hiszen ezáltal részben megérthetjük, hogy Nokia számára miért kardinális a felhő alkalmazása.

Az infrastruktúra kiválasztása során két fontos faktort vehetünk figyelembe, a deployment módját és a szolgáltatói függőséget. Többféle paradigma közül választhatunk, mint például magán-, közösségi-, nyilvános-, és hibrid felhős környezetek, illetve helyszíni infrastruktúra.

A helyszíni infrastruktúra az egyik legrégebbi hoszting módszer; a szerverek a vállalat telephelyén találhatóak és a szervezet tulajdonát képezik. Az infrastruktúra és a berendezések kezelése a vállalaton belül megkönnyíti a jogszabályi előírások betartását és magas szabadsági fokot tesz lehetővé. Ellene szól viszont, hogy szinte lehetetlen skálázhatóságot biztosítani, komplikált a karbantartása, illetve szakképzett IT csapat szükséges az üzemeltetéshez. [4]

A privát felhős környezet alatt azt értjük, mikor egy szervezet saját felhőt építtet ki és üzemeltet egy beszállító által. Ennek a megoldásnak többek között előnye, hogy a szervezetnek teljes autoritása van az izolált szerverek felett, ezeket saját üzleti igényeire szabhatja. Ezáltal nagyobb fokú skálázhatóságot érhet el, mint helyszíni infrastruktúrák segítségével. Hátránya a módszernek szemben az eddigiekkel a magas üzemeltetési költség, ezen kívül jelentős előzetes tőkekiadások léphetnek fel és külsős beszállító szükséges a fenntartásához. A hardverek elhelyezése lehet lokális (on-premise), esetleg hoszting szolgáltatónál elhelyezett saját, illetve kizárólagos használatra bérelt szerverek VPN eléréssel. [4]

A publikus felhőszolgáltatások megjelenésével és térnyerésével új távlatok nyíltak meg a hoszting megoldások világában, jellemzőbbé vált a magas rendelkezésre állás, globális elérhetőség, skálázhatóság és stb. A felhő publikusnak nevezhető, mikor a szolgáltatások olyan nyilvános hálózatokon, illetve kiszolgálói erőforrásokon keresztül jönnek létre, amelyek mások számára is elérhetőek. Ilyen szolgáltatásnak minősül például a Microsoft Azure, Google Cloud Platform, Amazon Web Services és Microsoft 365. A publikus felhőszolgáltatások ideálisak lehetnek számunkra, ha fontos, hogy költséghatékony, skálázható és rugalmasan hozzáférhető rendszert kapjunk, illetve ha a cél az üzemeltetésre szánt erőforrás minimalizálása. Ebben az esetben fontos továbbá, hogy igény szerint gyorsan és könnyedén tudjunk erőforrásokat provizionálni, emellett hangsúlyt fektessünk arra, hogy a mindig rendelkezésre álló legújabb technológiákból versenyelőnyt kovácsolhassunk. Publikus szolgáltatások esetében alapos fejtorést okozhat az, hogy szenzitív projektek által támasztott adatkezelési előírásoknak teljes mértékben megfeleljünk, mivel az erőforrások felett nincs közvetlen kontrollunk. Éppen emiatt komoly figyelmet kell fordítanunk a lehetséges szolgáltatás-kiesésekre és a helyreállításra. [4]

A közösségi felhőszolgáltatás inkább tekinthető modellnek, mint egzakt deployment módszernek. Ezt azzal magyarázhatjuk, hogy a többi módszernél eggyel magasabb absztrakciós szinten helyezkedik el, hiszen több felhőszolgáltatásból állhat össze. Megvalósulhat tisztán magán vagy publikus szolgáltatásokból, de akár ezek hibrid kombinációja is előfordulhat. Célja, hogy azonos érdekeltségi körbe tartozó szervezetek közös platformon dolgozhassanak együtt. A közösségi felhőszolgáltatásra akkor lehet szükség, mikor valamilyen speciális biztonsági előírásnak kell megfelelni, vagy ezt valamilyen szerver-specifikus alkalmazás, illetve fejlesztés megköveteli. [4]

A hibrid felhőszolgáltatás – a közösségi szolgáltatáshoz hasonlóan – inkább modellnek tekinthető deployment módszer helyett, mivel célja, hogy kombináljon egy/több publikus szolgáltatást a szervezet meglévő, vagy tervezett privát és/vagy helyszíni infrastruktúrájával. Erre jó bizonyíték a legtöbb nagyvállalat, melyek előszeretettel használnak valamiféle címtár⁶ megoldást. A legelterjedtebb ilyesfajta megoldás a Microsoft Active Directory (AD) terméke. Ha egy vállalat rendelkezik már egy meglévő on-premise AD deploymenttel és szeretne felhőt is igénybe venni, akkor a bevett eljárás szerint megtartja az on-premise AD környezetét, és ezzel együtt valamilyen AD szinkronizációs megoldással köti össze a felhős AD tenantjával. Az AD szinkronizációs alkalmazás felel azért, hogy az identitások mindkét környezetben elérhetőek legyenek, így ha egy kliens az on-premise, illetve felhő végponthoz kapcsolódik, akkor is képes lesz az identitás használatára. [4]

3.5. Alkalmazás portfólió realizálása és migráció

Felhős környezetre történő migráció során elengedhetetlen, hogy a megfelelő stratégiát válasszuk ki, hiszen ez akár a siker záloga is lehet. Azt gondolnánk, hogy új megoldások használatához mindig új eljárásokra lehet szükségünk, pedig ez nem mindig igaz. A felhőre történő migráció általános stratégiájának megalkotása során egy már létező stratégiát dolgoztak tovább, az alkalmazás portfólió racionalizálás (Application Portfolio Rationalization - APR) nevezetű stratégiát.

Az APR stratégiát elsősorban az on-premises környezet racionalizálásra alkották meg, de természetesen így önmagában is alkalmazható felhős környezetekre is. Vegyünk egy igen gyakori scenáriót, mint például, hogy egy vállalat két vállalatirányítási információs rendszert (Enterprise Resource Planning – ERP) használ. Eredetileg csak egy ERP-alkalmazással rendelkezett, a másodikat akkor vezették be, mikor a vállalat felvásárolt egy másik szervezetet, amely relatív hasonló alkalmazást használt. Két ERP rendszer fenntartása nem csak magas üzemeltetési költségeket eredményez, de vállalat adatpiacát is megosztja, továbbá szimultán üzemeltetésük erőforrásigényes. Hasonlóképpen, ez az alkalmazáskészlet idővel “örökséggé” válik, ami szükségessé teszi az alkalmazások racionalizálását.

Ilyen forgatókönyvek esetében ajánlott alkalmazni az APR stratégiát következők elérése érdekében:

- A duplikált alkalmazások és funkciók megszüntetése.

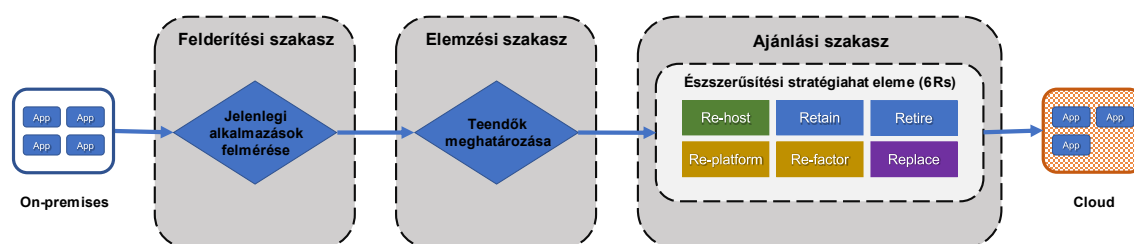
⁶ A címtár egy olyan szolgáltatás, ami a hálózat minden erőforrását azonosítja, és hierarchikus rendszerben tárolja.

- Azon alkalmazások eliminálása, melyek már használaton kívüliek, de még mindig üzemelnek.
- A magas üzemeltetési és kezelési költségekkel járó alkalmazások lecserélése.
- Az örökséggé vált és skálázásra képtelen alkalmazások átírása vagy újratervezése.

Célok melyekre törekszünk egy APR folyamat során:

- IT költségek csökkentése.
- A rendelkezésre álló költségvetés optimalizálására és a felesleges költségek csökkentésére.
- Komplexitás csökkentésére.
- Redundáns alkalmazások és rendszerek csökkentésére.

A felhőre történő migráció megtámogatásához specializált APR stratégia alább látható a 3. ábrán. Ennek a stratégiának elsődleges célja a migráció megvalósítása, míg csak másodlagos szerepet kap az alkalmazás portfólió realizálása. Fontos kiemelni, hogy ez nem azt jelenti, hogy a realizálás folyamata elhanyagolható, hiszen költségmegtakarítás szempontjából igenis lényeges, viszont a folyamatokat első sorban a migráció szempontjából nézzük és mérlegeljük.



3. ábra: Az APR stratégia felhőre tovább dolgozott változatának folyamatábrája (Forrás: [5] alapján saját szerkesztés)

Mérvadó különbség az on-premises-es és felhős APR között, hogy míg az előbbinél hangsúlyt fektetünk a teendők analizálásra, az utóbbinál már nem tesszük ezt, mivel a felhőre történő migráció magában hordozza a teendőket is. Továbbá az ésszerűsítési stratégiák halmaza kibővült a „Rehost” akció elemmel, vagyis az alkalmazás átmozgatásával úgy, ahogy van. Az ábrán jelölésre került, hogy az egyes akció elemek mely szolgáltatás-modellre történő migrációra illenek. A való életben sajnos nem lehet ilyen egyszerűen kategorizálni szolgáltatás szintekre az akció elemeket.

- **Re-host (átmozgatás eredeti állapotban):** A re-host során technikailag a „lift and shift” migrációs stratégiát valósítjuk meg. Az alkalmazást minimális erőfeszítéssel új virtuális hardverkörnyezetbe mozgatjuk át a felhőben. Az alkalmazás adatai úgy kerülnek át a felhőbe, hogy nem végzünk semmilyen átalakítást az adathalmazon, így az alkalmazás zökkenőmentesen működhet tovább. [6]

Háromféleképpen történhet meg egy ilyen átmozgatás:

- a virtuálisgép képállományának átmozgatásával,
- a futó infrastruktúra virtualizálásával és az elkészült képállomány mozgatásával,
- az alkalmazás konténerizációjával és a konténer képállomány mozgatásával.
- **Retain:** Azokat az alkalmazásokat, melyek nem állnak készen a felhőbe költözésre, megtartjuk. Ezen alkalmazásokat később felülvizsgáljuk, és adott esetben, amikor a felülvizsgálat megtörtént, átmozgatjuk a célkörnyezetbe. [6]
A következő esetekben érdemes megtartani egy alkalmazást a teljesség igénye nélkül:
 - Mikor az alkalmazás nem támogatott az adott környezetben.
 - Amikor egyes jogi és adatvédelmi szabályozások megkövetelik, hogy az alkalmazás on-premises környezetben fusson tovább.
 - Az alkalmazás dobozos vagy kész szoftver, melynek licenz szerződése nem teszi lehetővé a migrációt.
- **Retire:** A feleslegessé vált és felhős migrációra alkalmatlan alkalmazások nyugdíjazása. Ilyenkor nem csak hagyjuk a migrációra váró alkalmazások listájáról, de egyben feltesszük a „leszerelésre” (decommission) váró alkalmazások listájára. [6]
- **Re-platform:** Elsősorban platform szolgáltatásra történő áthelyezésre törekszünk, például Microsoft IIS-ről Azure Web Apps-re migrálás. Az alkalmazás alapvető architektúrája a legtöbb esetben nem változik. [6]
- **Re-factor:** Mikor egy magasabb szolgáltatási szintet szeretnénk igénybe venni, mint például PaaS szolgáltatások, előfordulhat, hogy használatba kell vennünk valamilyen gyártói SDK-t, vagy újra fejleszteni az alkalmazást a magasabb szintű hatékonyság érdekében. [6]
- **Replace:** Az alkalmazás lecserélése első sorban SaaS szolgáltatásra. Erre jó példa az a helyzet, mikor egy cég nem szeretne továbbra is Microsoft Exchange és Active Directory szervereket üzemeltetni, ezért átáll Microsoft 365-re és Azure Active Directory-ra. Előbbiekből mindkettő SaaS szolgáltatás, így nem csak egy modernebb ökoszisztémára áll át a szervezet, hanem egy magasabb szolgáltatási szintet is vesz igénybe, ahol a továbbiakban csak az adminisztrációs feladatokra kell fókuszálnia. [6]

Lényeges megemlíteni, hogy mindkét APR verzió során mikor re-platform vagy re-factor akciót végzünk, akkor egyben az alkalmazásunk modernizálását is végezzük. Habár az APR-nak nem elsődleges célja az alkalmazások modernizálása, magát az alkalmazást kontextusba helyezve az akció végrehajtása során olyan megoldások megválasztására kell törekednünk, amiktől egy hatékonyabb alkalmazást kapunk vissza. Egy alkalmazást többféleképpen is korszerűsíthetünk, például a kódrendszer részleges vagy teljes átírásával, néhány szolgáltatás cseréjével, az alkalmazás feletti szolgáltatási réteg újráválasztásával vagy architektúra optimalizálásával.

3.6. Szolgáltatói függőség kérdése

Jogosan felmerülhet a kérdés, hogy mi célravezetőbb? Egyetlen publikus szolgáltatót választani az egyre bővülő kínálatból, vagy több szolgáltatót is felvenni a palettára, majd pedig az igények mentén egy vagy több gyártó megoldására fejleszteni az alkalmazást? Multi-cloud stratégiának nevezzük azt a scenáriót, mikor több gyártó publikus felhőszolgáltatását vesszük igénybe és ezeket különböző arányban használjuk infrastruktúránk üzemeltetése során. Általánosságban elmondható, hogy különböző szolgáltatók egyidejű használata alapvetően szélesebb tudással rendelkező IT csapatot igényel. Ezen kívül valamilyen harmadik féltől származó szoftveres megoldás beszerzésére is szükség lehet, mely képes egységesített felületet biztosítani a különböző platformok üzemeltetéséhez és monitorozásához. Sokkal kézenfekvőbb megoldásnak tűnhet egyetlen szolgáltatót preferálni, viszont ez könnyedén előidézheti a vendor lock-in⁷ jelenségét. [7]

A vendor lock-in elkerülésének érdekében különböző lehetőségek közül választhatunk. Sokat alkalmazott módszer például a multi-cloud stratégia alkalmazása, illetve a szerződéskötés esetén a kilépési feltételek (exit stratégia) precíz és pontos definiálása. Magasabb szinteken érdemes elkerülni a fejlesztés során a platform specifikus megoldások és SDK-k használatát, illetve ajánlott olyan architektúrák fejlesztésére törekedni, melyek lazán kötődő komponenseket tartalmaznak. Ha minimalizálni tudjuk a vendor lock-in valószínűségét a cégünk stratégiájában, akkor egy esetleges szolgáltatóváltás esetén, illetve on-premises környezetre való visszatérésnél sokkal egyszerűbbé válhat a migrációs folyamat. [7]

A Nokiánál a hibrid multi-cloud stratégia érvényesült 2021 elejéig. Ez a gyakorlatban úgy valósul meg, hogy a vállalat saját adatközpontjainak üzemeltetésén felül igénybe veszi az Amazon, Google és Microsoft felhőszolgáltatásait is. A Nokia továbbra is propagálja a multi-cloud stratégiát, viszont jelen helyzetben már az infrastruktúra üzemeltetését teljes mértékben publikus felhőszolgáltatókra bízta. Ezzel a lépéssel megszünteti az összes jelenlegi privát adatközpontját és ezeket a Google Cloud Platformra migrálja át. Ezzel együtt viszont továbbra is folytatja a Microsoft és Amazon szolgáltatásainak igénybevételét.

⁷ A vendor lock-in egy közgazdaságtanban használt kifejezés, mely az jelenti, hogy a felhasználó függ egy gyártó vagy terjesztő termékeitől vagy szolgáltatásuktól, és csak nagyobb költségek árán tud váltani.

4. FUNKCIONÁLIS SPECIFIKÁCIÓ

Az alábbi fejezetben részletezem a projekt során megfogalmazott magasan absztrahált követelménylistát, amelyet a megrendelő fogalmazott meg a fejlesztő csapat számára. Ez a lista magában foglalja a felhasználói interfészek és backend komponensek követelményeit, továbbá ajánlásokat is tartalmaz egyes komponensek lehetséges megvalósításra.

4.1. Felhasználói felületek

A rendszernek két felhasználói felülettel kell rendelkeznie. Ezek közül az elsődleges felület a folyamatosan frissített riportok vizualizációjáért felelős BI dashboard, amelynek valamilyen SaaS szolgáltatáson kell alapulnia. A másodlagos egy egyénileg fejlesztett web alapú felhasználói felület, amely a felhasználók előbbi felülethez való hozzáféréseinek kezelését teszi lehetővé.

A BI felület fejlesztéséhez és hosztolásához a Microsoft Power BI-t ajánlom, míg a hozzáférés menedzsment megvalósításához egy kliens-szerver architektúrán alapuló alkalmazás fejlesztését tanácsolom. A front-end fejlesztéséhez a React keretrendszer, míg a backend fejlesztéséhez a .NET Core OData keretrendszer előnyösebb véleményem szerint.

A felületek irányába támasztott elvárások:

- A felhasználói interfészek megjelenésének meg kell felelnie a Nokia márkakövetelményeinek, mint például a logó használatának specifikus módja, Nokia által jóváhagyott színpaletta, betűtípusok stb.
- Reszponzív, modern megjelenés, mely lehetővé teszi az alkalmazás kényelmes használatát számítógépen, táblagépen vagy okostelefonon egyaránt.
- Csak intraneten keresztül legyen elérhető.

4.2. Felhasználói profilok

A rendszer felhasználói körét két csoportra oszthatjuk a korábban már említett két felhasználói felület mentén. A riportok vizualizációját minden Nokia alkalmazott elérheti, ez alól kivételt képeznek a sanghaji Nokia Bell Labs alkalmazottak. Az utóbbi kivétel alól felmentést kaphatnak egyes dolgozók. E folyamatok adminisztrációját a hozzáférés menedzsment alrendszeren keresztül lehet elvégezni, ezt a rendszert csak meghatározott adminisztrációs jogkörökkel rendelkező alkalmazottak érhetik el.

4.3. Forrásadatok

A rendszerhez szükséges adatokat két API biztosítja, ezek a következők:

- A rendszer adatokkal való feltöltéséhez egy beszállító által fejlesztett és üzemeltetett REST API áll rendelkezésre. Ez az API biztosítja a rendszer számára a kérdőív adatainak elérhetőségét.
- A hozzáférés menedzsment alrendszer megvalósításához szükséges alkalmazotti adatok feltöltéséhez az Azure Active Directory-t használjuk. Az adatok töltéséhez a Microsoft Graph API-t ajánlom.

4.4. Adatok töltése és feldolgozása

A forrás adatok feldolgozásáért és töltéséért ETL⁸ folyamatok felelnek, továbbá a különböző logikák megvalósítása és komponensek vezérlése is az ETL eszköz segítségével történik, így ez az eszköz részben orkesztratori szerepet is betölt.

4.5. Számításokat végző komponens

A különböző mutatók és statisztikák előállításáért szintén az ETL eszköznek kell felelnie, míg a különösen szenzitív, üzleti titoknak minősülő formulákkal való számítások elvégzése egy erre a célra fejlesztett alrendszernek a feladata. A formulákat egy erre kijelölt fejlesztő implementálja megfelelő titoktartási szerződések aláírása után egy .NET Core osztálykönyvtár formájában, és végül ezt elérhetővé teszi a CPV fejlesztő csapat számára egy fordított DLL állományként.

4.6. Monitorozás

Az esetleges hibákat és különböző történéseket szigorúan naplóznia kell minden egyes alrendszernek. Ha a rendszerek működésében valamiféle anomália lép fel, rendkívül fontos, hogy arról a fejlesztő automatikusan értesítést kapjon e-mailben a monitorozásért felelős interfésztől.

⁸ Az extract, transform és load angol szóhármass rövidítése, mely meghatározza az adatáramlás folyamatát és az adaton elvégzett feladatokat. Első lépésként extraktumokat veszünk át a forrásrendszerből valamilyen előre definiált módszer mentén, majd ezeket transzformáljuk a kívánt formára, végül pedig betöltjük a célrendszer adatbázisába vagy adattárházába.

5. NEM FUNKCIONÁLIS KÖVETELMÉNYEK

Ebben a fejezetben feltételeket fogalmazok meg a rendszer szolgáltatásaira és funkcióira a nem funkcionális specifikáció keretein belül, illetve kitérek a fejlesztésre vonatkozó elvárásokra és sztenderdekre is.

5.1. Performancia

Teljesítmény szempontjából fontos elvárás, hogy a riportok vizualizációi gyorsan betöltsenek, illetve hogy a vizualizációk minimális várakoztatással frissüljenek a különböző paraméterek és filterek módosítását követően.

A webes felületektől elvárt teljesítménybeli követelmények:

- Böngészőverziók közti kompatibilitás
- Platform-függetlenség
- Rövid betöltési idő
- Alacsony erőforrásigény
- Jól tömörített állományok

5.2. Biztonság

Fontos, hogy a rendszer komponensei kizárólag intraneten keresztül legyenek elérhetőek és a felhasználók számára nem releváns végpontokat tűzfal védje, illetve a komponensek az egymással való kommunikációt privát hálózaton keresztül bonyolítsák le. A különböző webes komponensek alkalmazzanak HTTPS protokollt, míg a titkosítási protokoll minimum TLS 1.2 verziójú legyen.

A felhasználói felületek minden esetben azonosítsák a felhasználót és erre az egységesített Nokia webes egyszeri bejelentkezési (Single Sign-On) interfészt használják.

5.3. Karbantarthatóság

5.3.1. Infrastruktúra

A projekttel kapcsolatos egyik elvárás, hogy az alkalmazás újbóli lefejlesztésre kerüljön felhős platform-szolgáltatások használatával, ebből adódóan a felhőszolgáltató az üzemeltetés terhének nagy részét leveszi a vállunkról. Különböző finomhangolásokra ettől függetlenül továbbra is lehetősége van a fejlesztő és ügyfél támogatásért felelős csapatoknak, mint például alkalmazások újraindítása, portok módosítása, stb.

Esetleges nem várt anomáliák esetén alkalmazható a környezet újbóli provizionálása, mely erőforrás leírók (ARM template) segítségével valósítható meg.

5.3.2. Kódbázis

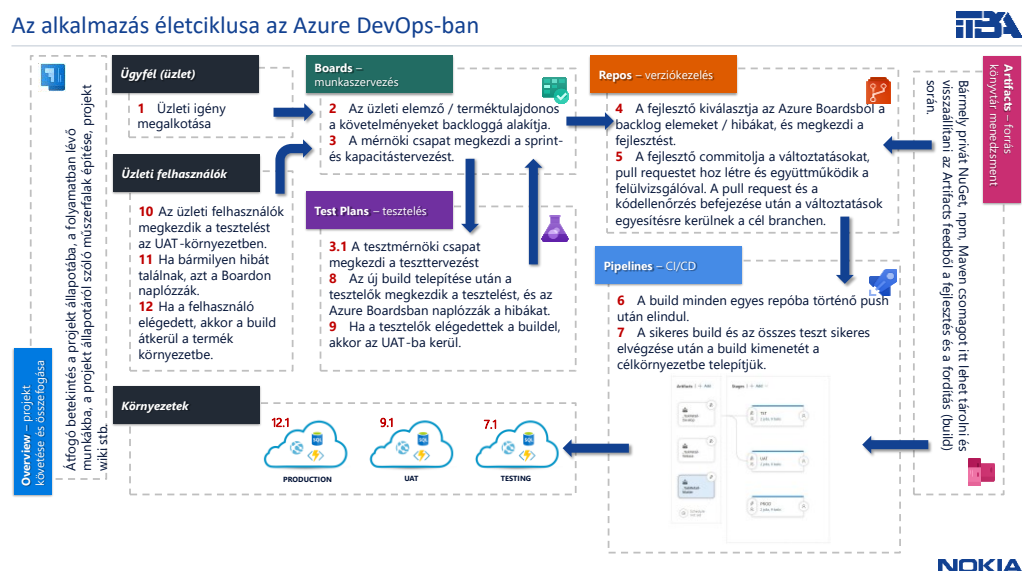
Minden fejlesztő vagy platform által generált kódot egy kijelölt git repostiroy-ban kell verzió kezelni. Ide tartoznak például a .NET Core alkalmazások, ARM templatek, Azure Data Factory ARM templatek, stb.

Elvárt teendők a kódbázis karbantartása terén:

- Kódbázis folyamatos refaktorációja. (Tiszta kód (Clean-Code) alapelveinek követése.)
- Futtatókörnyezetből és külső könyvtárakból eredő hibák és sérülékenységek kiküszöbölése frissítésekkel.
- Nagyobb futtatókörnyezet-frissítés esetén a kódbázis részleges vagy akár teljes újraírása.
- Előre meghatározott könyvtárstruktúra rendszerezetten tartása.
- Kód dokumentáció frissen tartása, mint például kód kommentek, generált és kézzel írt dokumentumok.

5.4. DevOps

Az egyik legfontosabb szempont, hogy a fejlesztés Agile-Scrum metodológia alapján kerüljön lebonyolításra, míg a szoftver kiadás Azure DevOps segítségével valósuljon meg (lásd: 4. ábra).



4. ábra: Alkalmazások életciklusa az Azure DevOps-ban (Forrás: saját szerkesztés)

Egy szoftverkiadásnak 3 fázison kell átesnie. Először be kell vezetni a fejlesztői környezeten, majd a minőségbiztosítási környezeten és végül a szoftver kikerülhet a végleges termék környezetre. A különböző környezetek között minden esetben egy kijelölt személynek jóvá kell hagynia az adott bevezetést, amelyet automatikusan az erre kijelölt céleszköz bonyolít le.

A kódbázis fordításáért, ellenőrzéséért és teszteléséért különböző automatizált, folyamatos integrációs folyamatok felelnek, továbbá ezek publikálják az eredményt a szoftverkönyvtárba. Ha elérhetővé válnak új verziók a szoftverkönyvtárban, akkor a folyamatos bevezetésért felelős folyamatok automatikusan elindulnak. Ezek ellenőrzik az adott környezet konfigurációit majd pedig elvégzik a bevezetést, illetve szükség esetén az infrastruktúra provizionálását is. Manuális bevezetés kizárólag szükséges esetekben történik, ilyen eset lehet például a rendkívüli hibajavítás.

5.5. Ügyféltámogatás

Az elkészült rendszer támogatásáért két csapat felelős, egy dedikált, folyamatosan rendelkezésre álló ügyféltámogatói csapat, illetve egy, az alkalmazást fejlesztő csapat. Az ügyféltámogatói csapathoz érkeznek be a különböző hibajegyek a felhasználóktól, ezenkívül ők oldják meg azokat a problémákat, amelyek nem igényelnek magas szintű technikai támogatást. A fejlesztői csapathoz egyedül akkor eszaklálódik hibajegy, hogyha a fejlesztők bevonása elengedhetetlen a megoldáshoz. A legtöbb esetben ilyenkor szükségessé válhat az infrastruktúra konfigurációjának finomhangolása, de akár a kódbázis módosítása is.

5.6. Rendelkezésre állás

Az alkalmazás nem lát el olyan üzletileg kritikus feladatokat, amelyek megkövetelnék a magasabb szintű rendelkezésre állást. Ebből kifolyólag elegendő a tőle elvárt rendszernek a 99.9% rendelkezésre állásra törekednie (Service-level agreement - SLA).

Az Azure szolgáltatások szolgáltatás szinten biztosítják a különböző rendelkezésre állási szinteket, ezekről a Microsoft Service Level Agreement szerződése rendelkezik.

A felhasználói felületek az alábbi Azure és Microsoft szolgáltatásokra épülnek:

- Azure SQL Database General Purpose – 99,99% elérhetőség [8]
- Azure App Service – 99,95% elérhetőség [9]
- Power BI Pro – 99,9% elérhetőség [10]

6. MEGKÖZELÍTÉSI MÓDOK ÉS MEGOLDÁSI LEHETŐSÉGEK ÁTTEKINTÉSE

A következőkben áttekintem azokat a megoldási lehetőségeket, melyek a korábban már felsorolt funkcionális specifikációban megfogalmazott elvárásoknak eleget tesznek. Mivel a megrendelő előre meghatározta a felhőszolgáltatót, így dolgozatomban kizárólag a Microsoft Azure és a Microsoft ökoszisztémába illő szolgáltatásokra összpontosítok. Nem részletezem a szolgáltatóválasztás eredményeit, mert a Microsoft Azure szolgáltatása már átesett az átvilágítási szempontokon és elfogadott a használata a Nokián belül. Ezen felül megvizsgálom, hogy mely keretrendszerek és futtatókörnyezetek lehetnek alkalmasak az egyedi fejlesztésű webes felhasználói interfészek, illetve számításokat és címtár szinkronizációt végző komponensek fejlesztésére.

Tekintettel arra, hogy CPV platform több különböző partikulumból és felhasználói interfészekből tevődik össze, így a könnyebb rendszerezés érdekében logikai modulokra bontom az alkalmazás egységeit a feladataik alapján, a későbbiekben így fogok rájuk hivatkozni.

A rendszer az alábbi modulokból tevődik össze:

- **Adattárház modul:** Az ETL eszköz által végzett adattöltéseket és kalkulációkat valósítja meg, továbbá az adatok tárolására használt adatbázis karbantartásáért is felelős.
- **Számításokat végző modul:** a szenzitív formulák implementációjáért és kalkulációk elvégzésért felel.
- **BI modul:** egy felhasználói interfészt foglal magába, az adattárház modul által elvégzett kalkulációk és adatok vizualizációjáért felel.
- **Hozzáférés menedzsment modul:** egy felhasználói interfészt és egy backend API-t foglal magába, célja a különböző sorszintű és oszlopszintű hozzáférések kezelése és jogosultságok kiosztása címtárból származó adatok mentén. A modulon keresztül felvitt korlátozások a BI modul által vizualizálható adathalmazt szűkítik és a felhasználó jogköreinek függvényében jelenítik meg azokat.
- **Címtár szinkronizációs modul:** célja az adatok töltése és szinkronizálása az adattárház modul adatbázisa és a címtár között.

6.1. Microsoft Azure specifikus szolgáltatások kiválasztása

A szolgáltatások kiválasztását modulspecifikusan végzem, ehhez pedig több különböző szolgáltatást választok ki az Azure portfóliójából. A későbbiekben döntöm el összehasonlítás mentén, hogy melyik szolgáltatás látná el a célfeladatot a lehető legjobban. A döntéshozatal során a fő szempont a performancia, illetve a célfeladat

ellátására való alkalmasság. A szolgáltatások kiválasztása során természetesen figyelembe kellett vennem azok költségvonzatait is, ez – a belső üzleti érdekek hatására – jelentős tényező volt a döntések meghozatalában.

6.1.1. ETL eszköz megválasztása

Az adattárház-modul megvalósításához szükséges egy olyan ETL eszköz, amely biztosítani tudja az adatok extrakcióját a forrás rendszerekből, ezen kívül rövid idő alatt képes nagy mennyiségű adat transzformációjára és töltésére. Fontos továbbá, hogy az ETL eszköz feladatkörébe tartozik a különböző alkalmazások szervezése és orkesztrálása. Az eszköznek képesnek kell lennie mind az ETL folyamatokat, mind a daemon alkalmazások vezérlését automatizált módon végrehajtani.

Az Azure szolgáltatás portfóliója két adatintegrációs szolgáltatást tartalmaz, az Azure Data Factory-t (ADF) és Azure Synapse Analytics-et (SYN).

Az ADF egy ETL-, ELT- és adatintegrációs projektek kezelésért felelős platformszolgáltatás. A fejlesztéshez egy kódolásmentes felületet kapunk, amely alatt különböző kapcsolt big data alapú rendszerek húzódnak meg és látnak el feladatokat. Az adatok feldolgozásért egy Spark klaszter, pontosabban az Azure HDInsight szolgáltatás felel. Az HDInsight nem csak Apache Spark klaszterek használatát teszi lehetővé, hanem Hadoop, R Server, HBase és Storm klaszterek felhasználását is. Az ADF harmadik gyártóktól származó kapcsolók széles választékával rendelkezik, mint például Oracle SQL Database, MySQL, Amazon Redshift és SAP Hana, továbbá képes on-premises és felhős adatforrásokhoz is kapcsolódni. Természetesen natívan támogatja a más Azure szolgáltatásokkal való kommunikációt, kapcsolók és feladatszervezés szintjén egyaránt. Abban az esetben, ha előzetesen is rendelkezünk meglévő SQL Server Integration Services (SSIS) környezettel, akkor azt integrálni tudjuk az ADF példányunkkal, így képesek lehetünk SSIS csomagok futtatására Azure-os környezetből. Említésre méltó, hogy lehetőségünk van git-tel integrálni az ADF példányunkat, így CI/CD folyamatokkal automatizálhatjuk bevezetéseinket. Az ADF egy ideális választás lehet, hogyha hagyományos adattárházat szeretnénk építeni, ahol nem lényeges a gyors, valós idejű feldolgozás és tervezhető a várható terhelés. [11]

A SYN az Azure Data Warehouse szolgáltatás továbbfejlesztett és átnevezett verziója, melyet a Microsoft az első számú adattárház megoldásaként tart számon. Rendelkezik ugyanazokkal a funkciókkal is, mint az ADF, viszont az egyik legnagyobb eltérés közöttük, hogy ezen kívül beépített T-SQL alapú relációs és nem relációs massively parallel

processing (MPP)⁹ SQL adatbázis klaszterekkel is fel van vértézve. Ily módon lehetőségünk van hatékonyan feldolgozni kötegelt adathalmazokat (batch) és folyamatosan változó adatfolyamokat (streaming) is közel valós időben. Ezen kívül még beépített gépi tanulási (machine learning) funkciókat, alternatív Azure DataBricks felületet, és teljes értékű Apache Spark és Azure Data Lake integrációkat is tartalmaz. A SYN ideális választás lehet, ha nagy mennyiségű adatokat kell feldolgoznunk komplex lekérdezésekkel, analitikus feldolgozásokat szeretnénk végrehajtani streaming adatokon, vagy ha gépi tanulást szükséges az adathalmazunk kielemezésére. [12]

A projektben megfogalmazott igények és az adatok mérete alapján az ADF ideális választás, hiszen a projektben csak orkesztrációra és ETL folyamatok levezénylésére van szükségem, melyeknek az ADF teljes mértékben eleget tesz.

6.1.2. Számítási szolgáltatások megválasztása

A számításokat végző modulnak egy olyan szolgáltatásra van szüksége, amely képes rövid idő alatt skálázódni, példány szintű párhuzamos végrehajtást tesz lehetővé és szolgáltatás szinten támogatja az esemény alapú indításokat. A hozzáférés menedzsment modulnak egy olyan komponense van, ami kliens-szerver architektúrára épül, így számítási kapacitást igényel, pontosabban ez a komponens a backend alkalmazás.

Az Azure szolgáltatás portfóliója öt olyan platformszolgáltatást tartalmaz, amely alkalmas lehet a modulok igényeinek teljesítésére, ezek pedig a következők:

- Azure App Service
- Azure Functions
- Azure Kubernetes Service (AKS)
- Azure Container Instances (ACI)
- Azure Service Fabric (ASF)

A projekt alkalmazásai nem rendelkeznek olyan szükségletekkel, amelyek megkövetelnék az alkalmazás-konténerizációt, így a korábban írt listából kiesik az alábbi 3 szolgáltatás: Kubernetes Service, Container Instances és Service Fabric. Az AKS és ACI szolgáltatások között a fő különbség, hogy az ACI egyetlen konténer példány futtatását teszi lehetővé, míg az AKS a Google Kubernetes technológiáján alapszik és konténer példányok klaszterekben szervezett futtatását biztosítja. Az ASF abból a szempontból lóg ki a sorból, hogy kifejezetten mikroszolgáltatások (microservices) fejlesztésére és

⁹ A massively parallel processing egy olyan adatbázis technológia, amely lehetővé teszi az analitikai munkafolyamatok kezelését, valamint nagy mennyiségű adat hatékony aggregálását és feldolgozását elosztott hálózati környezetben.

konténerizált futtatására fejlesztették ki. Az utóbbi szolgáltatás tökéletes választás lehet a számításokat végző modul számára, hiszen stateless¹⁰ és stateful¹¹ alkalmazások fejlesztését és példány szintű párhuzamos végrehajtást tesz lehetővé. Habár az ASF konténerizáción felül rendelkezik saját modell alapú fejlesztői nyelvvel, ez számomra még is kizáró ok, mivel a projekt megvalósítása során a daemon és backend alkalmazások fejlesztésekor egységes .NET kódbázisra törekszem.

Az App Service egy olyan szolgáltatás melynek célja, hogy a fejlesztőknek kizárólag az alkalmazásaik fejlesztésre kelljen fókuszálniuk a hosztingolás terhei nélkül. Ebből kifolyólag széles futtatókörnyezet támogatással rendelkezik, támogatja a mobilalkalmazások, webalkalmazások, APIok és függvény alkalmazások fejlesztését. Az App Service egy gyűjtő szolgáltatás, amelyre az Azure Web Apps az Azure Mobile Apps és az Azure Function Apps szolgáltatások ráépülnek. Összefoglalva, a korábbiakban már említett Azure Functions egy különálló szolgáltatásként listázott, de App Service-re épülő megoldás, és az ebből létrehozott példányt hívjuk Function App-nak. A már említett App Service-en alapú szolgáltatások közötti különbség az App Service-re ráépülő architektúrában és futtatókörnyezetekben mutatkozik meg. [13]

Az Azure Functions saját SDK-val rendelkező, esemény-vezérelt, igény szerinti számítási műveletek elvégzésre kifejlesztett megoldás. Lehetőségünk nyílik Azure, harmadik féltől származó, vagy on-premises rendszerekben bekövetkező események által triggerelt kódok végrehajtására serverless és dedikált szerveres környezetben. Natív integrációval rendelkezik Azureos szolgáltatások eseményeihez, ezen kívül opciónk van HTTP alapú API-t is fejleszteni. [14] A Functions alapesetben stateless függvények fejlesztését teszi lehetővé, viszont a durable function kiterjesztésnek köszönhetően megvalósíthatóak stateful függvények is, továbbá olyan tervezési minták is, mint munkafolyamatok monitorozása, párhuzamos feladatok orkesztrációja és feladatok soros végrehajtása. [15] Az Azure Function SDK támogatja a C#, JavaScript, F#, Java, PowerShell, Python és

¹⁰ A stateless („állapot nélküli”) protokoll egy olyan hálózati protokoll, amely során a kliens kérést küld a kiszolgálónak, és a kiszolgáló a megadott állapotnak megfelelően válaszol. A stateless szervernek ilyenkor nyilván kell tartania kommunikáció (session) részleteit, továbbá a feladat végrehajtása függ az interaktív felhasználó közvetlen irányításától. Ilyen protokoll például az FTP és Telnet protokollok.

¹¹ A stateful („állapot függő”) protokoll egy olyan hálózati protokoll, amely során a kliens kérést küld a kiszolgálónak, majd pedig valamiféle választ vár. Ha nem érkezik válasz, akkor újra elküldi a kérést. A stateful szerver nem követi a kommunikáció (session) részleteit, továbbá a feladat végrehajtása nem igényli az interaktív felhasználó közvetlen irányítását, tőle függetlenül üzemel. Ilyen protokoll például a HTTP, UDP és DNS protokollok.

TypeScript nyelveket, továbbá amennyiben egy függvény alkalmazást szeretnénk fejleszteni, ezt a webböngészőnkben az Azure portálon keresztül megtehetjük.

Az Azure Web Apps egy platformszolgáltatás, mely webalkalmazások és API-ok hosztolására hivatott. Széles körben támogat különböző futtatókörnyezeteket, mint például .NET, Python, Node.js és PHP. Olyan harmadik féltől származó kész alkalmazásokkal is kompatibilis, mint a WordPress, Joomla és Drupal tartalomkezelő rendszerek. Képes konténerizált, fordított és szkript alkalmazások hosztolására is. Automatikus horizontális skálázhatósággal rendelkezik, melyet akár manuálisan is finomhangolhatunk igényeink szerint. Amennyiben a vertikális skálázhatóságra keresünk automatikus megoldást, ezt az Azure Automation kiegészítő szolgáltatás segítségével tehetjük meg. Könnyedén integrálható DevOps és CI/CD megoldásokkal. Mindkét szolgáltatás rendelkezik különböző platformspecifikus funkciókkal, mint például szolgáltatás szintű Azure Active Directory autentikáció, Cross-Origin Resource Sharing (CORS) menedzsment, tűzfal, biztonsági mentés, skálázhatóság és forgalom menedzsment.

A számításokat végző, illetve a címtár-szinkronizációs modulok igényei alapján a legjobb választásnak az Azure Function-t tartom hiszen platform szinten implementál olyan funkciókat, mint feladat orkesztráció és stateful alkalmazások támogatása. Az utóbbi két funkció azért is kiemelt fontosságú, mert a daemon alkalmazásnak felkészültnek kell lennie olyan scenáriók bekövetkezése esetén is, mint a feldolgozandó adatok mennyiségének hirtelen változása. Ilyen helyzetben minimális futásidő növekedés mellett kell biztosítani a közel konzisztens teljesítményt.

A hozzáférés-menedzsment modul jelenlegi követelményei alapján az alkalmazás célja az alkalmazottak Active Directory biztonsági csoportokhoz rendelése. Ebből adódóan erre a feladatra az Azure Function Apps lenne az optimális szolgáltatás egyszerűség és alacsony költség szempontjából, mégis az Azure Web Apps szolgáltatást tartom a jobb választásnak. Ennek az egyetlen oka, hogy az esetleges jövőbeli funkcióváltozások, illetve funkcióbővülések esetén az Azure Web Apps nagyobb rugalmasságot biztosíthat.

A frontend alkalmazás hosztolására szintén a Web Apps szolgáltatást választottam, mert a Nokia kiberbiztonsági csapata belső biztonsági sztemderdekre hivatkozva nem engedélyezi az Azure Blob Storage-ból történő kliensalkalmazások hosztolását. Ha nem lennének ilyen megkötéseim, akkor inkább a Blob Storage szolgáltatást választanám, mivel lényegesen olcsóbban tudnám hosztolni a kliens statikus fájljait, kiváló teljesítmény mellett. Még ennél is gyorsabb teljesítményt lehetne elérni, ha a Blob Storage mellé egy Azure Content Delivery Network szolgáltatást is igénybe vennék. Ebben az esetben a Blob Storage csak egy tároló szerepét látná el, míg a Content Delivery Network lenne a tényleges kiszolgáló. Ez a formáció rendkívül gyors globális elérést biztosítana a kliens fájlok számára.

6.1.3. Adatbázis szolgáltatás megválasztása

Az előzőekben már felsorolt modulok mindegyike valamilyen módon ugyanazt az adatbázist használja, így egy olyan szolgáltatás után néztem, amely minimális mennyiségű kompromisszum mellett képes ellátni a feladatot.

Az adatbázis-szolgáltatás kiválasztásánál az alábbi szempontokat éreztem a legfontosabbnak:

- Relációs adatbázis legyen
- Platform szolgáltatás legyen
- Részben automatikusan skálázható legyen
- Rendelkezzen beépített audit funkcióval
- Jól integrálható legyen Azure Data Factory-vel
- Entity Framework (EF) Core kompatibilis legyen
- Hatékonyan és rövid idő alatt futtasson le analitikus lekérdezéseket
- Azure Active Directory autentikációt biztosítson
- Beépített adatmaszkolást biztosítson
- Tűzfallal rendelkezzen

Az előbb felsoroltakon kívül azt is figyelembe vettem, hogy a fejlesztői csapatunk milyen szakmai tudással rendelkezik az adatbázis megoldások terén, így két termék került fel a listára. Ezek a termékek a Microsoft SQL Server és az Oracle Database. Mindkét szolgáltatás on-premises és IaaS szolgáltatásként érhető el, viszont az SQL Server rendelkezik Azure-os platform szolgáltatás alternatívával. Ez funkcionalitását véve minimális eltérésekkel bár, de megegyezik az SQL Server-rel és teljesíti a feljebb megfogalmazott követelményeket. Mivel platform szolgáltatásról beszélünk, így kifejezetten nagy figyelmet kell majd fordítanom arra, hogy a különböző lekérdezéseket megfelelően optimalizáljam költségmegtakarítás és hatékonyság szempontjából. [16]

6.1.4. BI vizualizációs eszköz megválasztása

A projekt megrendelése során kifejezett ügyféligényként került megfogalmazásra a BI vizualizációk Microsoft Power BI-ban történő megvalósítása, mert az ügyfeleknek már van meglévő tapasztalatuk az említett szoftverrel. Habár ez elsősorban nem egy technikai döntés volt, részükről ez egy tapasztalat alapú döntésnek minősül. Mivel rendelkezem Power BI-os tapasztalatokkal és nem fedeztem fel bármilyen inkompatibilitást a kiválasztott technológiai eszköztárral, így elfogadtam az igényt.

7. ARCHITEKTÚRA TERVEZÉS

Az architektúra megtervezése és validálása az egyik legszenciálisabb lépés egy informatikai projekt életciklusa során. Egy pontatlanul és nem eléggé részletekbe menően megtervezett rendszerterv a későbbiekben a projekt késéséhez és végeláthatatlan menet közbeni tervezéshez (on the fly engineering) vezethet, arról nem is beszélve, hogy kihatással lehet a végleges termék minőségére. Hivatalos „recepteskönyv” nem létezik felhő alapú rendszerek tervezéséhez, viszont a Microsoft biztosít egy Azure Architecture Center nevezetű ingyenesen elérhető internetes rendszerterv tárat. Itt valós példákat találhatunk Azure alapú megoldás architektúra tervekre, ezzel is megkönnyítve a tervezés folyamatát.

A felhőszolgáltatások előfizetés alapú konstrukciójából adódóan a várható költségekkel is terveznünk kell, erre pedig az Azure Pricing Calculator lehet a segítségünkre. Használatával könnyedén tudunk az előfizetési módunk vagy a Microsoft-tal kötött szerződésünk alapján költségtervezetet készíteni különböző rendszerterveinkhez. Így a költség szempontjából, több rendszerterv közül történő választás nem jár komoly erőfeszítéssel.

Az architektúra megtervezése során az előzőekben említett két eszköz lesz segítségemre.

7.1. Az infrastruktúra architektúrája

A rendszerterv komponensei egymással virtuális hálózaton keresztül kommunikálnak. Az adatbázis, daemon- és backend alkalmazások közötti hitelesítést részben alkalmazás, részben szolgáltatás szinten fogom implementálni. Ezt, ahol lehetett MSI¹² vagy OAuth 2.0 client credentials flow¹³ hitelesítéssel valósítottam meg (lásd: 1. sz. melléklet).

¹² A Managed Service Identity, másnéven „menedzselt identitások” olyan szolgáltatás identitást (service principal) biztosítanak az alkalmazások számára, amely segítségével Azure Active Directory hitelesítést támogató erőforrásokhoz hitelesítve tudok csatlakozni. Ezek az Azure szolgáltatások közvetlenül kérnek token-t az AAD-tól és használják egymás között autentikációra, anélkül, hogy ez a folyamat nyílt hálózatokon keresztül történne. Ezzel is minimalizálva a token illetéktelen kezekbe történő kerülését. Az Azure automatikusan karbantartja és újítja meg ezeket az identitásokat. [19]

¹³ OAuth 2.0 client credentials flow egy nyílt szabvány, amely működésében hasonlít a feljebb már említett MSI autentikációhoz annyi különbséggel, hogy az identitást manuálisan kell létrehoznom és karbantartanom, továbbá a token-t a szolgáltató nyílt API-ján keresztül tudom kérvényezni. Token-ek igényléséhez egy kliens azonosítóra és egy kliens kulcsra van szükségem, amit idővel mindig meg kell újítanom. Ha az utóbbi két kliens paraméter idegen kezekbe kerülne, akkor a behatoló korlátlanul tudna token-eket igényelni és hozzáférni a célrendszerekhez. Ebből kifolyólag körültekintően kell token-t kérvényeznem és kezelnem. [20]

Minden esetben igyekeztem az MSI hitelesítés használatára törekedni, mivel kényelmesebb és biztonságosabb a client credentials flow-val szemben.

A rendszerterv ábrájának magyarázata

Adattárház modul részei:

1. A kérdőív adatokat a külső beszállító API-án keresztül nyerek ki, amihez egy GET végpontot biztosítanak. Ezzel a végponttal történő kommunikáció során a beszállító által biztosított API kulccsal tudom hitelesíteni a rendszereimet.
2. Az API-t egy tűzfal védi, ami az interneten keresztül csak a Nokia publikus IP címeiről érkező kéréseket fogadja és engedi tovább az API szervere felé.
3. Az Azure Data Factory nyeri ki a beszállító API-járól az adatokat, amihez inkrementális töltést alkalmaz. URL paraméterek segítségével meg tudja határozni az API felé küldött kérésben, hogy milyen időintervallumból hiányoznak adatok, így célzottan azokat nyeri csak ki.
4. Az ADF a kinyert és transzformált adatokat betölti egy Azure SQL Database-be. Továbbá elvégzi a különböző statisztikai számításokat a betöltött adathalmazokon.
5. Az ADF egy http trigger segítségével elindítja a különböző Azure Function-öket, mint például a formula kalkulációs motor és a címtár szinkronizációnak függvényeit. A függvények hosztolására elegendő egy funkció példány, mivel több függvényimplementációt is tud egyidejűleg hosztolni.

Számításokat végző és címtár szinkronizációs modulok részei:

6. A címtár szinkronizációs modul függvényei Microsoft Graph-al való autentikációhoz token-t igényelnek az Azure Active Directory-től.
7. A címtár szinkronizációs modul függvényei az adatokat a Microsoft Graph API HTTP-GET végpontjain keresztül nyerek ki, hogy azokat később majd adatbázisba szinkronizálják.
8. A számításokat végző és címtár szinkronizációs modulok függvényei a SQL Database-ből nyerek az adatokat a logikáik futtatásához és az eredményeket ugyanebbe az adatbázisba írják vissza.

Hozzáférés menedzsment modul részei:

9. A hozzáférés menedzsment modul backend alkalmazása az adatokat az SQL Database-ből tölti be és manipulálja.
10. A felhasználó ahhoz, hogy kommunikálhasson a backend alkalmazással és a frontend hitelesítést igénylő részeivel, érvényes token-nel kell rendelkezzen. Ehhez a frontend először átirányítja a Nokia SSO interfészére, hogy ott token-t igényelhessen a további interakciókhoz.
11. A felhasználó számára a frontend statikus fájljait egy Azure Web App szolgáltatás hosztolja.

12. A hitelesítést és a frontend fájljainak böngészőbe történő betöltését követően a felhasználó közvetlenül kommunikál a backend-et hosztoló Azure Web App szolgáltatással.

BI modul részei:

13. A Microsoft Power BI az adatokat az SQL Database-ből szerzi meg.

A backend Wep App-ja és az Azure Function Apps példány külön-külön rendelkezik egy Azure Application Insights kiegészítő szolgáltatással, amelyek adatokat gyűjtenek a szolgáltatásokról. Ilyen adatok lehetnek például a rendszer hibák, logok, kérések száma, hibás kérések és telemetria adatok. A felhasználó a Power BI felületet a Microsoft 365 fiókjába integrálva tudja elérni.

7.2. Az adatbázis architektúrája

Az adatbázisba kerülő adatokat funkcióik mentén három különálló sémába szerveztem egy adatbázison belül (lásd: 2. sz. melléklet). A beszállító API-járól kinyerhető adatokat egy staging (köztes) nevezetű séma tábláiba töltöttem Data Factory segítségével. Az application (alkalmazás) séma tábláit szintén a Data Factory és a címtár szinkronizációs és számításokat végző modulok függvényei töltik és módosítják. A hozzáférés menedzsment és BI modulok egyaránt használják e réteg tábláinak adatait. A reporting (riportolás) sémában az ADF tölti fel a táblákat és hozza létre a nézeteket. Ennek a rétegnek az adatait a Power BI vizualizálja a felhasználó számára.

Tárgyalt táblák és leírásuk:

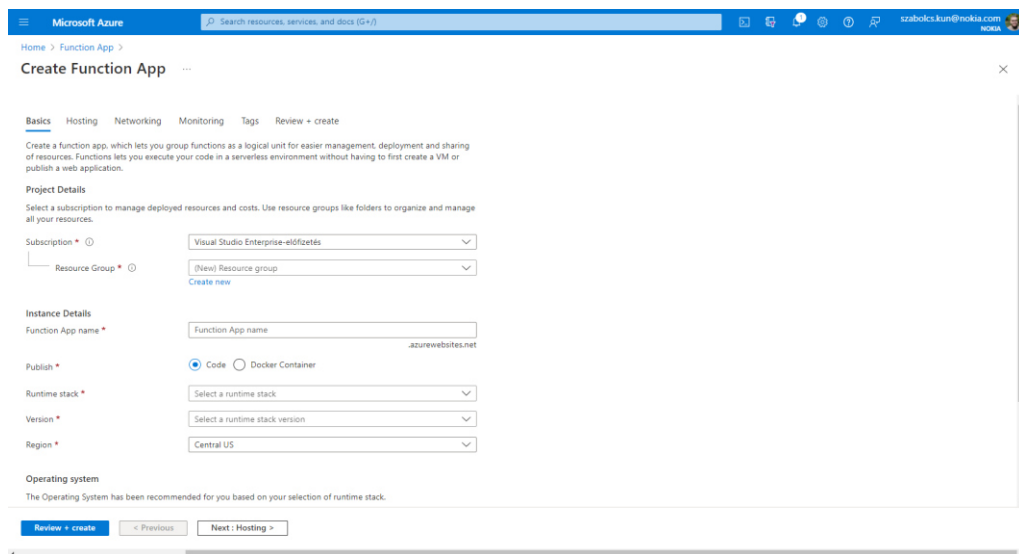
- CPV_SURVEY_RESPOND: A kérdőív kitöltéseket és egyes kalkulációk eredményeit tartalmazza.
- CPV_SURVEY_RESPONDENT_NAME: A kitöltők neveit tartalmazza.
- CPV_SURVEY_RESPONDENT_NAME_HIST: A kitöltők neveinek változtatásait tartalmazza.
- CPV_COMPETITOR: A versenytársak adatait tárolja.
- CPV_COMPETITOR_HIST: A versenytárs adatok változtatásait tartalmazza.
- CPV_CUSTOMER: Az ügyfelek adatait tartalmazza.
- CPV_CUSTOMER_HIST: Az ügyfél adatok változásait tárolja.
- CPV_CUSTOMER_X_CUSTOMER: Kapcsolótábla, amely az ügyfelek közötti kapcsolatot reprezentálja.
- CPV_SURVEY_RESPOND_CALCULATION: A válasz szintű kalkulációk eredményeit tartalmazza.
- CPV_CUSTOMER_CALCULATION: Az ügyfél szintű kalkulációk eredményeit tartalmazza.

- CPVI_COMPETITOR_ALL_LEVEL_CALCULATION: Versenytársakra vetített különböző szintű kalkulációk eredményeit tárolja.
- CPVI_NOKIA_ALL_LEVEL_CALCULATION: Nokia-ra vetített különböző szintű kalkulációk eredményeit tárolja.
- CPV_CALCULATION_CONTROL: A kalkuláció végrehajtásához szükséges vezérlőtábla.
- ACCESS_MANAGEMENT_USER: Felhasználók adatait tartalmazza.
- ACCESS_MANAGEMENT_REGION_LEVEL_RLS: Felhasználók adat szintű megszorításait tartalmazza.
- ACCESS_MANAGEMENT_GROUP: AAD csoportok listája.
- ACCESS_MANAGEMENT_GROUP_X_USER: Felhasználók csoporttagságának listáját tartalmazza.
- CPV_SURVEY_DAILY: Köztes tábla, amely a futás előtt mindig a legfrissebb másolatát tartalmazza a forrás API adathalmazának.

8. IMPLEMENTÁCIÓ FOLYAMATA

8.1. Felhős hoszting környezetek provizionálása

Azure-ban a különböző szolgáltatásokat manuálisan telepítettem az Azure portál webes felületén keresztül a szolgáltatás specifikus telepítő varázslók segítségével (lásd: 5. ábra). Két különböző környezetet telepítettem, egyet a fejlesztés és tesztelés támogatásához, míg a másikat az éles környezet hosztolására hoztam létre. Végül pedig az Azure Resource Manager (ARM) szolgáltatás telepítést leíró sablonokat pedig exportáltam a kódbázisomba a későbbi esetleges újra telepítések megkönnyítésének céljából.



5. ábra: Példa Azure Function létrehozására a felhasználói felületen keresztül a szolgáltatás varázsló segítségével (Forrás: saját szerkesztés)

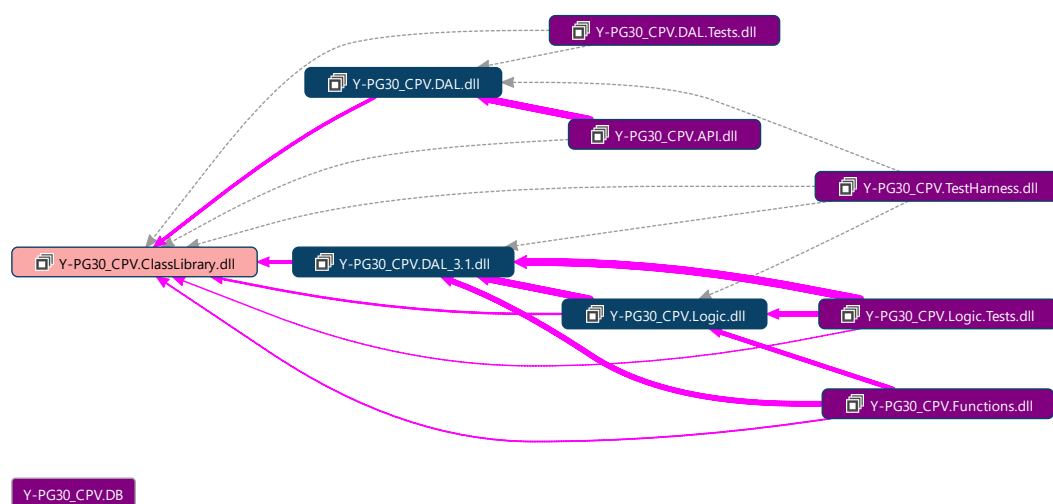
8.2. Kódbázis

A projekt kódbázisa két részre osztható: egy .NET-es kódbázisra és egy ADF által generált sablon halmazra. A kódbázis projektjeit a 1. táblázat részletezi, míg a köztük lévő függőségeket a 6. ábra szemlélteti. A backend és a démon alkalmazások .NET Core 3.1 futtatókörnyezetben és C# 8.0 nyelven készültek el. A fejlesztéshez használt IDE a Microsoft Visual Studio 2019.

Könyvtár	Specifikáció
Y-PG30_CPV.API	A hozzáférés menedzsment modul implementációját tartalmazza.

Y-PG30_CPV.ClassLibrary	Általános osztály kiterjesztéseket, konstansokat és megosztott kódrészleteket implementál, amelyeket igénybe vehet bármelyik másik könyvtár.
Y-PG30_CPV.DAL	Adathozzáférési réteg implementációja, amely Entity Framework Core 5.0-t használ.
Y-PG30_CPV.DAL_3.1	Adathozzáférési réteg implementációjának, ideiglenes Entity Framework Core 3.1-es állapota. Erre az osztálykönyvtárra azért volt szükség, mert az Azure Function nem támogatta még az 5.0-ás verziót.
Y-PG30_CPV.DAL.Tests	Az adathozzáférési réteg unit teszt könyvtára.
Y-PG30_CPV.DB	SQL DDL parancsokat és lekérdezéseket tartalmaz.
Y-PG30_CPV.Functions	Az Azure Function-ök implementációját tartalmazza, a lényeges üzleti logika egy külön osztálykönyvtárban került lefejllesztésre.
Y-PG30_CPV.Logic	Az Azure Function-ökhöz tartozó számításokat végző és címtár szinkronizációs modulok logikáját tartalmazza.
Y-PG30_CPV.Logic.Tests	A számításokat végző és címtár szinkronizációs modulok unit teszt könyvtára.
Y-PG30_CPV.TestHarness	Egyszerű konzolos alkalmazás gyors manuális teszteléshez.

1. táblázat: A kódrendszer projektjei



6. ábra: A kódrendszer projektjei közötti függőségek (Forrás: saját szerkesztés)

8.3. A CPV kalkulációk alapelvei

A CPV leglényegesebb része a különböző kalkulációs folyamatok, melyeknek eredményei kerülnek vizualizálásra Power BI segítségével. Három féle kalkulációs típust különböztetünk meg, azaz BIAS, CPV és CPVi. A BIAS kalkuláció a legelemibb értékek, amik meghatározásra kerülnek. Ezeket az értékeket használják fel a CPV és CPVi kalkulációk a későbbiekben. A BIAS értékek kiszámítása mindig kérdőív szinten történik meg, amelynek célja, hogy statisztikailag korrigálja a válaszadó jelenlegi kitöltés értékeit a korábbi kitöltései alapján. A CPV kalkulációk során meghatározzuk a jelenlegi és az elvárt teljesítményt, és ezeket az értékeket mindig kiszámoljuk különböző aggregációs szinteken is, mint például válaszadó, ügyfél és versenytárs. Tehát ugyanazt a kalkulációt végezzük minden esetben, viszont változik a felhasznált adathalmaz mérete. A CPVi kalkuláció nem sokban különbözik hagyományos CPV társától; ebben az esetben is a különböző aggregációs szintek között fedezhető fel eltérés, például az aggregációk típusai régió, ország és terméktípus szintekből tevődnek össze.

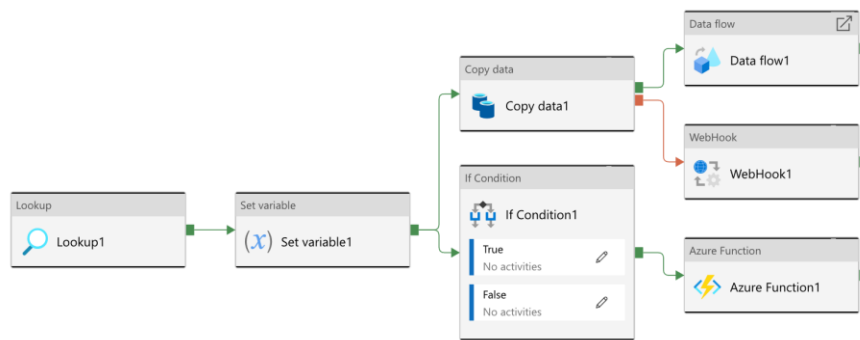
A számítások logikájának lényeges része, vagyis a CPV és CPVi kalkuláció az adattárház modulban, míg a BIAS kalkuláció a számításokat végző modulban került implementálásra.

8.4. Adattárház modul implementálása

Az adattárház modult Azure Data Factory-ben készítettem el, amely a klasszikus ETL folyamatok megvalósításán felül a különböző komponensek orkesztrációjáért és felügyeltéért felel.

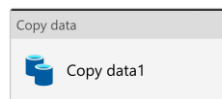
Az implementáció során az alábbi ADF komponenseket használtam:

- **Pipeline:** Az adatcsatornák különböző szolgáltatások natív csatlakozóinak segítségével adatok másolására és a munkafolyamatok szervezésére szolgálnak (lásd: 7. ábra). Sokszor hivatkozunk a pipeline-okra vezérlés irányítóként is, mivel végrehajtási sorrendet tudunk általuk definiálni, logikai egységbe zárják a különböző ADF objektumokat, triggereket pedig kizárólag rájuk tudunk létrehozni.



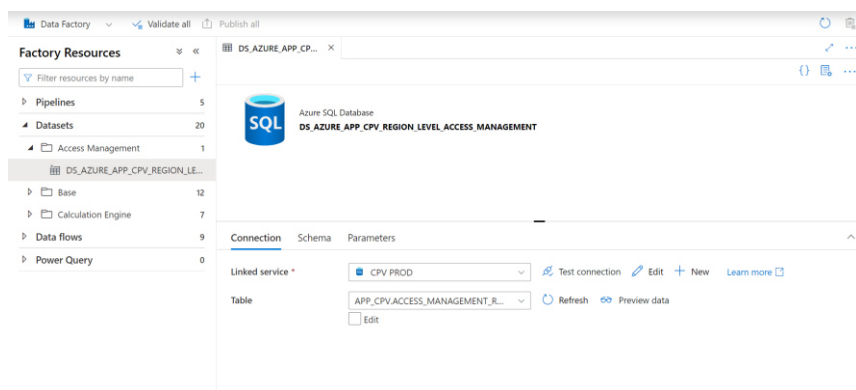
7. ábra: Példa pipeline-ra (Forrás: saját szerkesztés)

- **Activity:** A feladat blokkok az adatcsatornák részét képezik és valamilyen dedikált feladatot látnak el, melyek egy vagy több szolgáltatás csatlakozót támogatnak. Ilyen például a Copy azaz másolás blokk, amely egy adatforrásból (source) másol adatokat egy cél adattárolóba (sink) (lásd: 8. ábra). Forrás és cél egyaránt lehet adatbázis tábla, FTP szerverten fellelhető állomány, stb.



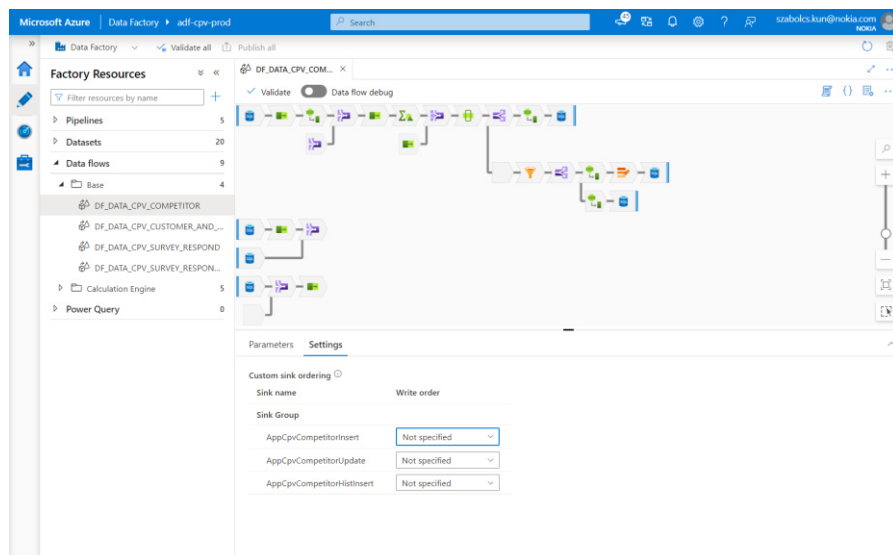
8. ábra: Példa másolás activity-re (Forrás: saját szerkesztés)

- **Datasets:** Az adatkészletek különböző adattárolókban tárolt adatok azonosítására szolgálnak, mint például adatbázis táblák és nézetek, állományok, API végpontok, hierarchikus dokumentumok, stb. Egy dataset leírja az általa hivatkozott entitás tulajdonságait és adott esetben akár a sémáját is (lásd: 9. ábra).



9. ábra: Példa dataset-re: egy Azure SQL Database dataset (Forrás: saját szerkesztés)

- **Data flows:** Az adatfolyamok segítségével vizuálisan tudjuk megtervezni egy probléma adattranszformációját anélkül, hogy ehhez bármilyen kódot is kellene írunk (lásd: 10. ábra). Az így létrejövő adatfolyamatokat a pipeline-okon belül a data flow activity segítségével tudjuk végrehajtani. Ezek a data flow-k automatikusan skálázódó Apache Spark klasztereken futnak le.



10. ábra: Példa data flow-ra (Forrás: saját szerkesztés)

- **Linked services:** A kapcsolt szolgáltatások lehetővé teszik, hogy különböző szolgáltatásokkal létesíthessünk kapcsolatot. Ezek olyan natív szolgáltatás csatlakozók, amelyek segítségével olyan, az adatforrásokon értelmezett műveleteket tudunk végrehajtani, mint például olvasás, másolás, törlés és mozgatás. Egy linked service magában foglalja az adattárolók eléréséhez szükséges konfigurációt, valamint a csatlakozási kulcsot és a hitelesítés típusát (lásd: 11. ábra).

Edit linked service
 Azure SQL Database [Learn more](#)

Name *
 CVP_PROD

Description

Connect via integration runtime *
 AutoResolveIntegrationRuntime

Connection string [Azure Key Vault](#)

Account selection method
☐ From Azure subscription ☒ Enter manually

Fully qualified domain name *
 xyz.database.windows.net

Database name *
 xyz-db

Authentication type *
 System Assigned Managed Identity

Managed identity name: **adf-cvp-prod**
 Managed identity object ID: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
 Grant Data Factory service managed identity access to your Azure SQL Database. [Learn more](#)

Always encrypted ☐

Additional connection properties
 + New

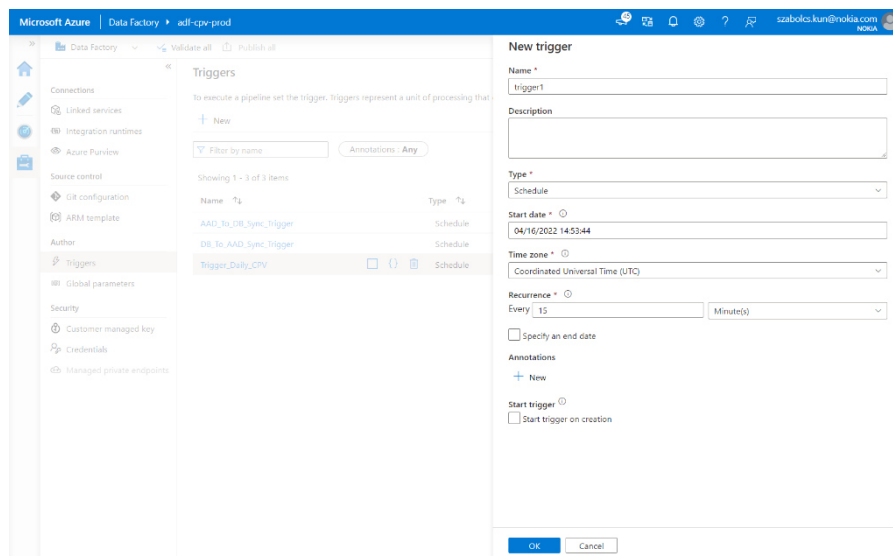
Annotations
 + New

> Parameters
 > Advanced

Save Cancel Test connection

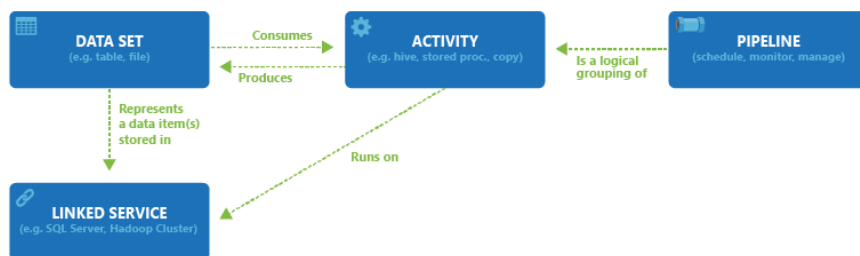
11. ábra: Linked service szerkesztésére és létrehozására szolgáló varázsló (Forrás: saját szerkesztés)

- **Integration runtime (IR):** Az integrációs futtatókörnyezet különböző pipeline-oknak biztosít számítási infrastruktúrát, ezáltal adatintegrációs feladatok ellátását téve lehetővé data flow-k, activity-k és SQL Server Integration Services (SSIS) csomagok számára. Három fajta IR létezik: Azure, Self-Hosted és Azure-SSIS. Az Azure IR egy teljesen menedzselt, elosztott, kiszolgáló nélküli számítási erőforrás. Mindig annyi erőforrást biztosít, amennyire szükség van az adott feladat ellátásához. Self-Hosted IR segítségével saját földi vagy szeparált felhős környezetünket tudjuk összekötni ADF-el, ehhez pedig saját szervereinket tudjuk felhasználni. Fontos megjegyezni, hogy csak adatmozgatásra és activity-k végrehajtásra tudjuk használni, adattranszformációra nem. Az Azure-SSIS IR akkor válhat hasznunkra mikor meglévő SSIS környezetünket szeretnénk lift-and-shift stratégiával a felhőbe mozgatni. Továbbiakban az implementáció során csak Azure IR-t használunk, ami magától létrejön minden egyes ADF példány létrehozásakor és legtöbb esetben „AutoResolveIntegrationRuntime”-nak nevezik.
- **Triggers:** A programindítók segítségével tudjuk automatizálni különböző pipeline-jaink indítását (lásd: 12. ábra).



12. ábra: Trigger létrehozására szolgáló varázsló (Forrás: saját szerkesztés)

A pipeline-ok vannak a legmagasabb szinten az objektumok közötti hierarchiában. Egy pipeline-t meghívhat egy vagy több trigger vagy akár egy másik különálló pipeline is, míg egy data flow-t csak is pipeline hívhat meg (lásd: 13. ábra).



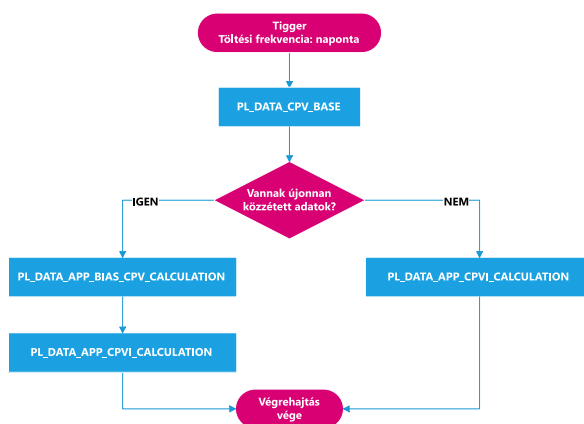
13. ábra: Kapcsolatok ADF objektumok között [17]

8.5. Adatcsatornák implementálása

Adatcsatornákat azaz pipeline-okat használtam fel ahhoz, hogy különböző alkotóelemeket, mint például data flow-kat, function-öket és egyéb egyszerű adatmozgatásokat könnyen értelmezhető logikai egységekbe zárjak. Igyekeztem arra törekedni, hogy teljes mértékben kihasználjam a pipeline-ok nyújtotta lehetőségeket és csak annyit hozzak létre amennyivel elkerülhetőek a kód duplikációk és jól strukturált kódbázis érhető el.

8.5.1. A PL_DATA_CPV_BASE adatcsatorna megvalósítása

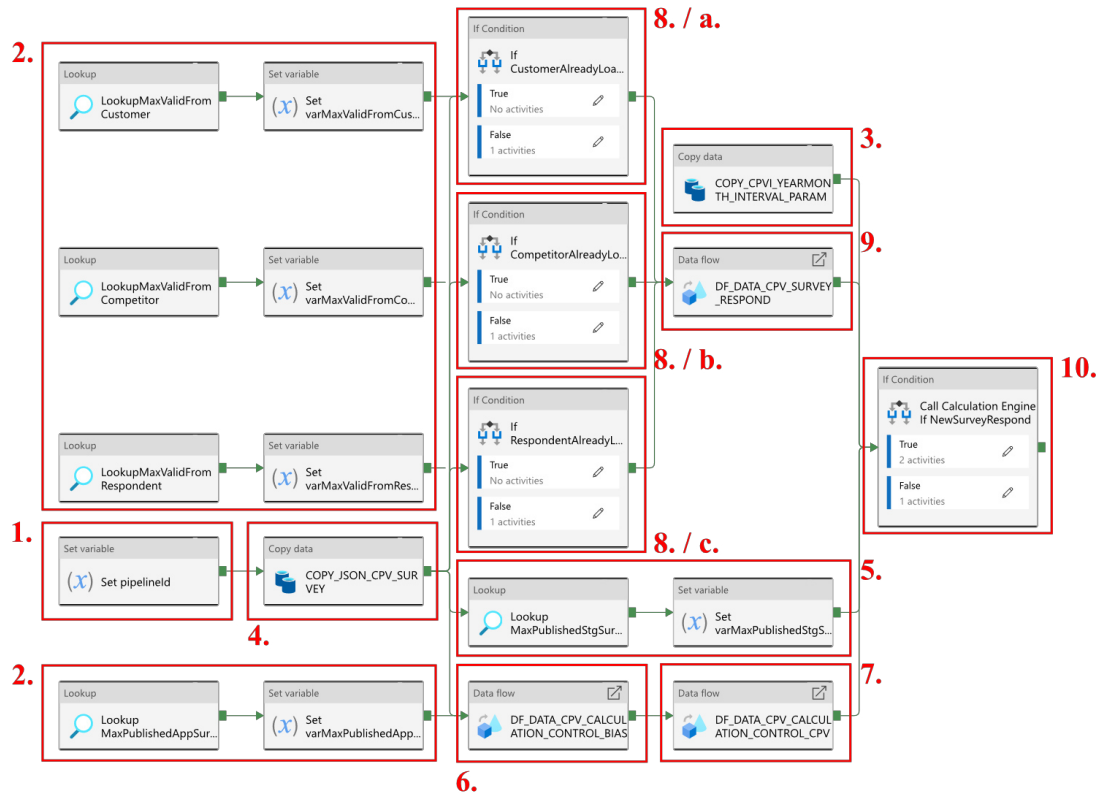
A PL_DATA_CPV_BASE nevezetű pipeline célja a köztes (staging) és az application séma tábláinak töltése, továbbá a PL_DATA_APP_BIAS_CPV_CALCULATION és a PL_DATA_APP_CPVI_CALCULATION pipeline-ok orkesztrációja (lásd: 14. ábra). Az adatcsatorna szintén meghatározza, hogy a különböző data flow-knak teljes vagy delta töltést kell-e végezniük.



14. ábra: A pipeline-ok közötti függőségek (Forrás: saját szerkesztés)

A pipeline végrehajtása naponta egyszer, éjfélkor megy végbe UTC-5 időzóna szerint, amiért egy trigger felelős. Először a PL_DATA_CPV_BASE adatcsatornában a köztes

tábla töltése kerül végrehajtásra, és ha ez sikeres volt, akkor ellenőrzöm, hogy van-e újonnan közzétett adat a staging táblában. Ha igen, akkor a teljes számítási modult kell lefuttatni a BIAS, CPV és CPVi számításokkal. Ha nem, akkor csak a CPVi-számításokat kell futtatni. Az alábbi (15.) ábra szemlélteti a pipeline felépítését.



15. ábra: A PL_DATA_CPV_BASE pipeline felépítése (Forrás: saját szerkesztés)

Az adatcsatorna az alábbi módon működik:

1. Beállítom a pipelineID változó értékét az adott futás GUID azonosítójára a @pipeline().RunId függvény segítségével. Ezt az értéket a LOAD_ID mezőbe töltöm be.
2. Lekérdezem a maximum VALID_FROM és PUBLISHED_DATE értékeket az alkalmazásréteg tábláiból:
 - LookupMaxValidFromCustomer: Lekérdezem a CPV_CUSTOM táblából a VALID_FROM mező maximum értékét és letárolom a varMaxValidFromCustomer változóban. Ha a tábla üres, akkor a változó értékét 1000-01-01 állítom.
 - LookupMaxValidFromCompetitor: Lekérdezem a CPV_COMPETITOR táblából a VALID_FROM mező maximum értékét és letárolom a varMaxValidFromCompetitor változóban. Ha a tábla üres, akkor a változó értékét 1000-01-01 állítom.

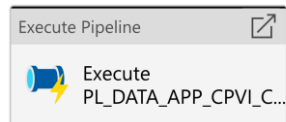
- LookupMaxValidFromRespondent: Lekérdezem a CPV_SURVEY_RESPONDENT_NAME táblából a VALID_FROM mező maximum értékét és letárolom a varMaxValidFromRespondent változóban. Ha a tábla üres, akkor a változó értékét 1000-01-01 állítom.
 - LookupMaxPublishedAppSurveyRespond: Lekérdezem a CPV_SURVEY_RESPOND táblából a PUBLISHED_DATE mező maximum értékét, ahol a „BIAS_” előtaggal rendelkező mezők értéke NULL és letárolom a varMaxPublishedAppSurveyRespond változóban. Ha a tábla üres, akkor a változó értékét 1000-01-01 állítom. Ez a CPV_CALCULATION_CONTROL betöltés során a delta-adatkészlet létrehozására szolgál.
3. Másolás activity: Az APP_CPV.CPVI_YEAR_MONTH_INTERVAL_PARAM táblázatot betöltése.
 4. Másolás activity: A forrás API végpontjáról lekéri az adatokat és betölti a STG_CPV.CPV_SURVEY_DAILY táblába. A másolás előtt letörölöm a tábla tartalmát.
 5. LookupMaxPublishedStgSurveyRespond: Lekérdezem a staging táblából a PUBLISHED_DATE mező maximum értékét és letárolom a varMaxPublishedStgSurvey változóban.
 6. A DF_DATA_CPV_CALCULATION_CONTROL_BIAS data flow meghívása. Ez az adatfolyam az APP_CPV.CPV_CALCULATION_CONTROL technikai táblába tölt CPV számítási szabályokkal kapcsolatos rekordokat.
 7. A DF_DATA_CPV_CALCULATION_CONTROL_CPV data flow meghívása. Ez az adatfolyam az APP_CPV.CPV_CALCULATION_CONTROL technikai táblába tölt CPV számítási szabályokkal kapcsolatos rekordokat.
 8. Ellenőrzöm, hogy bármilyen adat betöltésre került-e a dimenzió táblákba.
 - a. Ha az aktuális dátum nem nagyobb, mint a varMaxValidFromCustomer értéke, akkor meghívásra kerül a DF_CPV_CUSTOMER_AND_REGION_LEVEL_ACCESS_MANAGEMENT data flow. Ez az adatfolyam a CPV_CUSTOMER táblát tölti adatokkal.
 - b. Ha az aktuális dátum nem nagyobb, mint a varMaxValidFromCompetitor értéke, akkor meghívásra kerül a DF_DATA_CPV_COMPETITOR data flow. Ez az adatfolyam a CPV_COMPETITOR táblát tölti adatokkal.
 - c. Ha az aktuális dátum nem nagyobb, mint a varMaxValidFromRespondent értéke, akkor meghívásra kerül a DF_DATA_CPV_SURVEY_RESPONDENT_NAME data flow. Ez az adatfolyam a CPV_SURVEY_RESPONDENT_NAME táblát tölti adatokkal.
 9. Meghívom a DF_DATA_CPV_SURVEY_RESPOND data flow-t. Ez az adatfolyam a CPV_SURVEY_RESPOND táblát tölti adatokkal.
 10. Ellenőrzöm, hogy vannak-e új közzétett adatok a köztes táblában.

- Ha a varMaxPublishedStgSurvey változó értéke nagyobb, mint a varMaxPublishedAppSurveyRespond változó értéke, akkor meghívom a PL_DATA_APP_BIAS_CPV_CALCULATION és a PL_DATA_APP_CPVI_CALCULATION pipeline-okat. Lásd alábbi (16.) ábra.



16. ábra: A PL_DATA_APP_BIAS_CPV_CALCULATION és PL_DATA_APP_CPVI_CALCULATION pipeline-ok hívása (Forrás: saját szerkesztés)

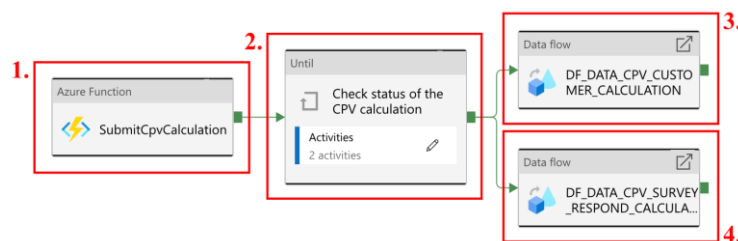
- Ha a varMaxPublishedStgSurvey változó értéke nem nagyobb, mint a varMaxPublishedAppSurveyRespond változó értéke, akkor meghívom a PL_DATA_APP_CPVI_CALCULATION pipeline-t. Lásd alábbi (17.) ábra.



17. ábra: A PL_DATA_APP_CPVI_CALCULATION pipeline hívása (Forrás: saját szerkesztés)

8.5.2. A PL_DATA_APP_BIAS_CPV_CALCULATION adatcsatorna megvalósítása

A pipeline feladata a BIAS és a CPV kalkulációk elvégzése. Az alábbi (18.) ábra szemlélteti a pipeline felépítését. A pipeline közvetlenül soha nem kerül meghívásra semmilyen trigger által. Egyedül a PL_DATA_CPV_BASE adatcsatorna hívhatja meg, mint alárendelt végrehajtó adatcsatornát.

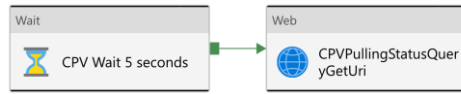


18. ábra: A PL_DATA_APP_BIAS_CPV_CALCULATION pipeline felépítése (Forrás: saját szerkesztés)

Az adatcsatorna az alábbi módon működik:

1. Meghívom a Submit-CpvCalculation Azure Function-t ezáltal elindítva a feladatot.

2. Ellenőrzöm az feladat futásának állapotát, amíg a function példány „Success” válasszal nem tér vissza. Lásd alábbi (19.) ábra.

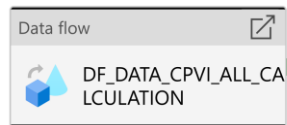


19. ábra: A function státuszának ellenőrzése

3. Meghívom a DF_DATA_CPV_CUSTOMER_CALCULATION data flow-t.
4. Meghívom a DF_DATA_CPV_SURVEY_RESPOND_CALCULATION data flow-t.

8.5.3. A PL_DATA_APP_CPVI_CALCULATION adatcsatorna megvalósítása

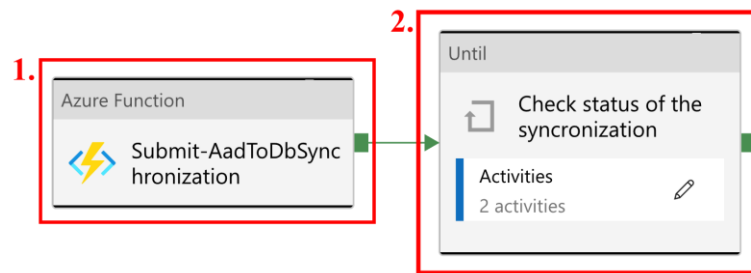
A pipeline feladata a CPVi kalkulációk elvégzése a DF_DATA_CPVI_ALL_CALCULATION data flow futtatásának elérésével. Az alábbi (20.) ábra szemlélteti a pipeline felépítését. A pipeline közvetlenül soha nem kerül meghívásra semmilyen trigger által. Egyedül a PL_DATA_CPV_BASE adatcsatorna hívhatja meg, mint alárendelt végrehajtó adatcsatornát.



20. ábra: A DF_DATA_CPVI_ALL_CALCULATION meghívása (Forrás: saját szerkesztés)

8.5.4. A PL_DATA_CPV_AAD_TO_DB_SYNC pipeline megvalósítása

A pipeline feladata a címtárból az adatbázisba történő szinkronizációért felelős function meghívása és a futási eredmény lekövetése. Az alábbi (21.) ábra szemlélteti a pipeline felépítését. Minden héten vasárnap este 11 órakor UTC-5 időzónában kerül trigger-elésre egy schedule trigger, azaz egy időzített programindító segítségével. Azért döntöttem az előző időpont mellett, mert ez fedi le a legtöbb régiót, amikor valamennyi Nokia alkalmazott éppen nem dolgozik, így az adattöltés ideális esetben nem befolyásolhatja munkavégzésüket. A heti egy töltés pedig azért elegendő, mert a szükséges címtáradat nem változik olyan frekvencián, hogy igény legyen a gyakoribb töltésre.



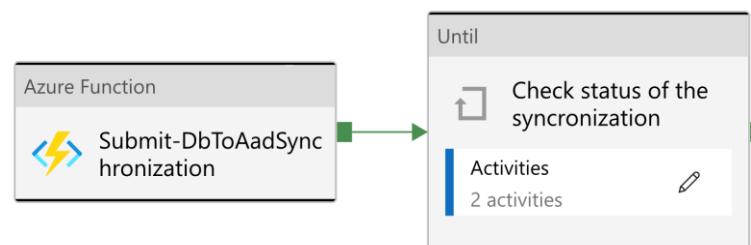
21. ábra: A PL_DATA_CPV_AAD_TO_DB_SYNC pipeline felépítése (Forrás: saját szerkesztés)

Az adatcsatorna az alábbi módon működik:

1. Meghívom a Submit-AadToDbSynchronization Azure Function-t ezáltal elindítva a feladatot.
2. Ellenőrzöm az feladat futásának állapotát, amíg a function példány „Success” válasszal nem tér vissza.

8.5.5. A PL_DATA_CPV_DB_TO_AAD_SYNC adatcsatorna megvalósítása

Az alábbi pipeline hasonlóképpen működik, mint az előzőleg már bemutatott PL_DATA_CPV_AAD_TO_DB_SYNC adatcsatorna, annyi eltéréssel, hogy az adatbázisból a címtárba történő szinkronizációért felelős function-t hívja meg és ennek a futási eredményét követi le. Az alábbi (22.) ábra szemlélteti a pipeline felépítését.



22. ábra: A PL_DATA_CPV_DB_TO_AAD_SYNC pipeline felépítése (Forrás: saját szerkesztés)

8.6. Adatfolyamok implementálása

Az adatfolyamok segítségével valósítottam meg az adatok transzformációját. Az általam létrehozott data flow-k két kategóriába sorolhatóak, azaz vannak a „base” és a „calculation engine” flow-k.

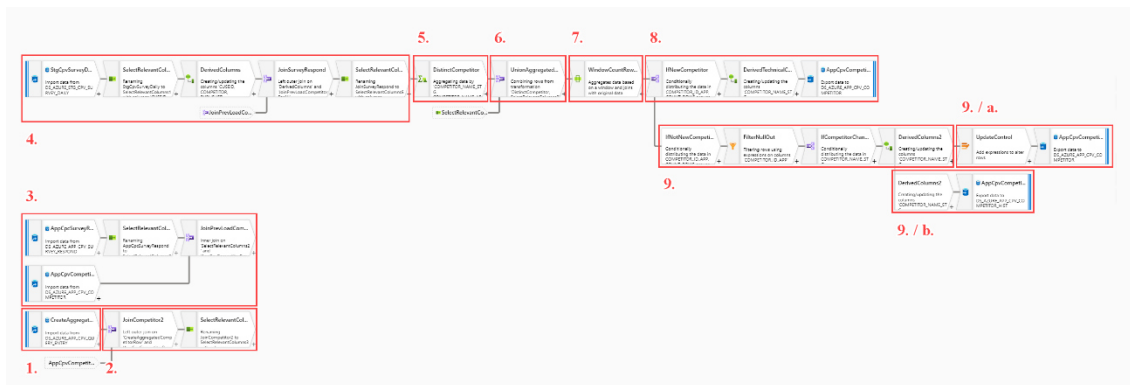
A base adatfolyamok fő célja egy vagy több tábla töltése és slowly changing dimension (SCD) megvalósítása a verziókövetés és adatintegritás megőrzésének reményében. Ha egy rekord valamelyik attribútuma megváltozik, akkor SCD Type 4-et alkalmazok ennek

lekezelésére, azaz az adott tábla historikus megfelelőjébe kerül átvezetésre a rekord jelenlegi verziója és a fő táblába íródik be az új verzió. Fontos kiemelnem, hogy sem hagyományos sem logikai törlést nem értelmeztem az implementáció során, mivel nem fordulhat elő olyan eset, hogy az API végpontról nyert adatból kivezetésre kerüljön bármilyen rekord.

A „calculation engine” flow-kban implementáltam az üzleti logika nagy részét, azaz a különböző kalkulációkat, mint például CPV és CPVi. Az üzleti logika maradék részét pedig a számításokat végző modulban található Azure Function segítségével valósítottam meg. Ezt részletesen a következő fejezetben tárgyalom.

8.6.1. A DF_DATA_CPV_COMPETITOR adatfolyam megvalósítása

Ennek a data flow-nak a célja APP_CPV.CPV_COMPETITOR tábla töltése, az esetleges változtatások átvezetése a CPV_COMPETITOR_HIST táblába és minden versenytársra elkészíteni az aggregált rekordokat a CPV_COMPETITOR táblában. Lásd alábbi (23.) ábra. A forrás API adathalmaza nem rendelkezik olyan kulcs mezővel, amely egyedileg azonosítaná az adott kérdőív rekord versenytárs adatait. Létrehoztam a táblában egy ORIGINAL_COMPETITOR_NAME nevezetű mezőt, amelyet a versenytárs nevével töltök fel és propagálom kulcsnak. A változásokat azért tudjuk azonosítani, mert mindig a teljes adathalmazt kérem le a forrás API-ról, így össze tudom hasonlítani az újabb válaszokat azzal, ami már az adatbázisban van. A táblák töltése során beszúrást és frissítést alkalmazok.



23. ábra: A DF_DATA_CPV_COMPETITOR data flow felépítése (Forrás: saját szerkesztés)

Az adatfolyam az alábbi módon működik:

1. Létrehozom az „Aggregated Competitors” rekordot.
2. Join segítségével összekapcsolom az „Aggregated Competitors” rekordot a CPV_COMPETITOR táblával, és kiválasztom a releváns mezőket. Erre azért van szükségem, mert ha már betöltöttem egyszer ezt a rekordot a táblába, akkor szükségem van a már meglévő mezőire.

3. Összekapcsolom az APP_CPV.SURVEY_RESPOND és az APP_CPV.CPV_COMPETITOR táblákat egymással, és kiválasztom a megfelelő mezőket. E lépés kimenetét arra fogom használni, hogy összehasonlítsam az alkalmazási rétegben lévő értékeket az új staging rekordokkal. Továbbá az APP_CPV.SURVEY_RESPOND táblára azért van szükségem, mert a versenytárs adatoknak nincs egyedi kulcsa a forrás API-adathalmazában.
4. Left join-nal összekötöm az alkalmazás réteg tábláit, ehhez pedig a CUSEID és PUBLISHED_DATE mező párost használom fel kulcsképzéshez. Végül pedig szűkítem az adathalmazt a szükséges versenytársak adataival kapcsolatos mezőkre.
5. Kiszűröm az ismétlődő sorokat a versenytársak adathalmazából.
6. Union segítségével egyesítem az „Aggregated Competitors” rekordot az előző lépésben előállított adathalmazzal.
7. Csoportosítom a köztes tábla rekordjait a COMPETITOR_NAME_STG mező mentén és megszámlolom, hogy hány rekorddal rendelkeznek a különböző csoportok.

Szétválasztom az eredményhalmazt:

8. Egy rekord akkor minősül új rekordnak, hogy ha az alkalmazás rétegbeli CPV_COMPETITOR táblában nem lelhető fel, ekkor beszúrom a táblába és beállítom az SCD mezőinek értékeit. Pontosabban a VALID_FROM értékét a jelenlegi időpontra, míg a VALID_TO értékét „2/31/9999” dátumra állítom. A rekord egyediségének eldöntéséhez megnézem, hogy a COMPETITOR_ID_APP mező NULL értékű-e és a 7. lépésből származó sorszáma egyenlő-e 1-el.
9. Ha egy versenytárs rekord nem minősül új rekordnak az alkalmazásrétegben, és a COMPETITOR_NAME_STG mező értéke nem egyenlő a COMPETITOR_NAME_APP mező értékével, akkor a versenytárs adatai változtak.
 - a. Frissítem a CPV_COMPETITOR táblában a VALID_FROM, VALID_TO és COMPETITOR_NAME mezőket.
 - b. Beszúrom a régi rekordot a CPV_COMPETITOR_HIST táblába.

8.6.2. ADF_CPV_CUSTOMER_AND_REGION_LEVEL_ACCESS_MANAGEMENT adatfolyam megvalósítása

A DF_CPV_CUSTOMER_AND_REGION_LEVEL_ACCESS_MANAGEMENT data flow-nak célja az APP_CPV.CPV_CUSTOMER tábla töltése ügyfél adatokkal, az esetleges változtatások átvezetése a CPV_CUSTOMER_HIST táblába és a row-level security (RLS) megvalósításához szükséges ACCESS_MANAGEMENT_REGION_LEVEL_RLS tábla töltése. Lásd a 3. számú mellékletet. Mint az előző data flow-nál, itt sem rendelkezik a részadathalmazom forrás

kulcsnak propagálható kulccsal. Habár a táblám rendelkezik SAP_ID nevezetű mezővel, viszont nem minden ügyfélnek van SAP ID-ja, ezért ezt nem használhatom kulcsként. Így a változások lekövetéséhez az ügyfél nevét használom fel forráskulcsként, mivel mindig teljes adathalmazzal dolgozom, továbbá egy válaszrekord állandóan rendelkezik konzisztens azonosítókkal.

Az adatfolyam az alábbi módon működik:

1. Összekapcsolom az AppCpvSurveyRespond és az AppCpvCustomerPrevLoad adatforrásokat.
2. Left join segítségével összekapcsolom az StgCpvSurveyDaily adatforrást az 1. lépés eredményével a CUSEID és PUBLISHED_DATE mezőkből képezett összetett kulcs segítségével és csak a szükséges mezőket tartom meg.
3. Kiszűröm az ismétlődéseket.
4. Csoportosítom a köztes tábla rekordjait a CUSTOMER_NAME_STG mező mentén és megszámlolom, hogy hány rekorddal rendelkezik a különböző csoport.

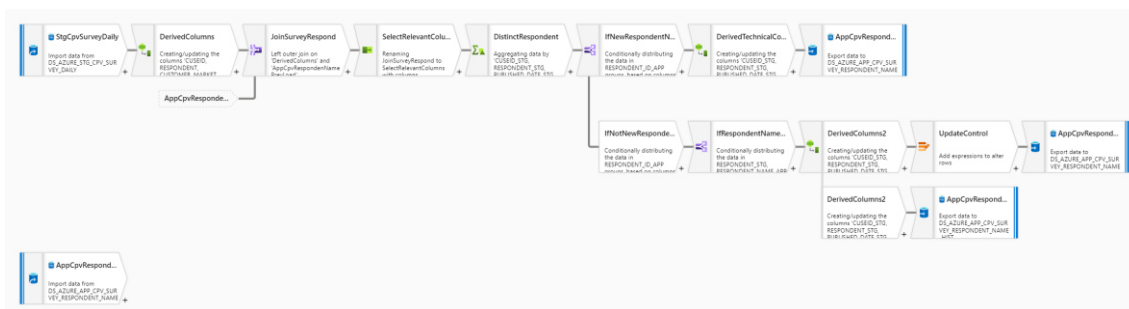
Szétválasztom az eredményhalmazt:

5. A rekord különbségének vizsgálata a versenytárs flow-val megegyezően történik.
6. Ha egy ügyfél rekord nem minősül új rekordnak az alkalmazásrétegben, akkor megvizsgálom, hogy módosult-e. Akkor minősül módosítottnak, hogy ha az alábbi mezők valamelyikének az értéke változott: CUSTOMER_STG, MARKET_STG, MU_GCBT_STG, CBT_STG, INT_ACCOUNT_STG, COUNTRY_STG és SAP_ID_STG.
 - a. Frissítem a CPV_CUSTOMER táblában a VALID_FROM és VALID_TO mezőket.
 - b. Beszúrom a régi rekordot a CPV_COMPETITOR_HIST táblába.
 - c. Összekötöm az ACCESS_MANAGEMENT_REGION_LEGION_LEVEL_RLS táblát a régi értékek alapján minden hozzáférési szinthez (MU GCBT, Market, Int. Account, Country, CBT). Összesítem a megváltozott access level értékeket, majd kiszűröm a duplikációkat és frissítem a céltáblát.

8.6.3. A DF_DATA_CPV_SURVEY_RESPONDENT_NAME adatfolyam megvalósítása

Ez a data flow az APP_CPV.CPV_SURVEY_RESPONDENT_NAME tábla töltéséért és a változások APP_CPV.CPV_SURVEY_RESPONDENT_NAME_HIST táblába történő lekövetéséért felel. Lásd alábbi (24.) ábra.

A data flow működési elve megegyezik a DF_DATA_CPV_COMPETITOR data flow-éval. Eltérés a forrás és cél adathalmazok között van, továbbá nem egészítjük ki a céladathalmazt plusz egy konstans sorral.



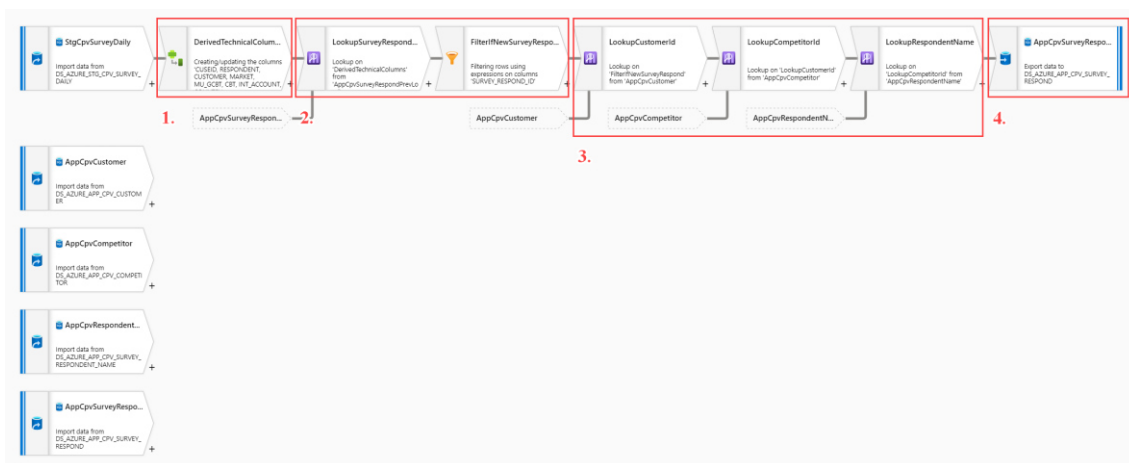
24. ábra: A DF_DATA_CPV_SURVEY_RESPONDENT_NAME data flow felépítése (Forrás: saját szerkesztés)

8.6.4. A DF_DATA_CPV_SURVEY_RESPOND adatfolyam megvalósítása

Ezen data flow felelős a válasz adatok kinyerésért a köztes táblából és a APP_CPV.CPV_SURVEY_RESPOND táblába töltéséért. Lásd alábbi (25.) ábra.

Az adatfolyam az alábbi módon működik:

1. Létrehozom a különböző technikai mezőket, mint például LOAD_ID, SRC_ID_1, SRC_ID_2, VALID_FROM és VALID_TO, továbbá elvégzem az adattípus konverziókat.
2. Előállítom a delta adathalmazt CPV_SURVEY_RESPOND tábla segítségével, úgy, hogy SRC_ID_1 és SRC_ID_2 mezőket összehasonlítom a két adathalmaz között és megtartom a kettő különbségét.
3. Kikeresem a dimenzió táblákból (CPV_CUSTOMER, CPV_COMPETITOR, CPV_SURVEY_RESPONDENT_NAME) az adott rekordhoz tartozó idegenkulcsokat és beállítom a referenciáikat.
4. Beszúrom az APP_CPV.CPV_SURVEY_RESPOND táblába a delta rekordokat.



25. ábra: A DF_DATA_CPV_SURVEY_RESPOND data flow felépítése (Forrás: saját szerkesztés)

8.6.5. A DF_DATA_CPV_CALCULATION_CONTROL_BIAS adatfolyam megvalósítása

A DF_DATA_CPV_CALCULATION_CONTROL_BIAS adatfolyam felel a részleges ügyfél adatok és publikációs dátumok áttöltéséért az APP_CPV.CPV_CALCULATION_CONTROL táblába, ami meghatározza, hogy majd mely rekordok eshetnek át BIAS kalkuláción. Lásd alábbi (26.) ábra.

Az adatfolyam az alábbi módon működik:

1. Kiszűröm a varMaxPublishedAppSurveyRespond változó értékénél kisebb vagy azzal egyenlő publikált dátumú rekordokat.
2. Eltávolítom a duplikált sorokat a CUSTOMER és PUBLISHED_DATE mező párosok mentén.
3. Kiegészítem az adathalmazt az alábbi kalkulált mezőkkel:
 - CALCULATION_INVOLVED: értéke konstans, azaz „BIAS”.
 - CALCULATION_LEVEL_NAME: értéke konstans, azaz „Customer”.
 - CALCULATION_LEVEL_VALUE: értéke konstans, azaz „CUSTOMER”.
 - CALCULATED_ENTITY: értéke konstans, azaz „n.a.”.
 - COUNTING_RULE_FLAG: értéke konstans, azaz „1”.
 - COUNTING_RULE_REASON: értéke konstans, azaz „Constant 1”.
 - CALCULATION_DATE, CALCULATION_YEAR, CALCULATION_MONTH, CALCULATION_YEAR_MONTH mezők a PUBLISHED_DATE értékéből generálódnak.
4. Beillesztem a rekordokat az APP_CPV.CPV_CALCULATION_CONTROL táblába.



26. ábra: A DF_DATA_CPV_CALCULATION_CONTROL_BIAS data flow felépítése (Forrás: saját szerkesztés)

8.6.6. A DF_DATA_CPV_CALCULATION_CONTROL_CPV adatfolyam megvalósítása

Az alábbi adatfolyam felel a részleges ügyfél adatok és publikációs dátumok áttöltéséért az APP_CPV.CPV_CALCULATION_CONTROL táblába, ami meghatározza, hogy majd mely rekordok eshetnek át CPV kalkuláción. Továbbá előállítja az APP_CPV.MYCPV_EFFECTIVE_HISTORY_RULE_PARAM táblát, amit a Power BI használ fel a MyCPV kalkulációk során. Lásd a 4. számú mellékletet.

Az adatfolyam az alábbi módon működik:

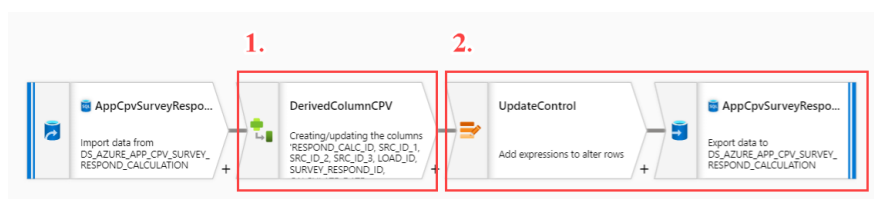
1. Kiszűröm azokat a rekordokat az AppCpvCalculationControlBIAS adatforrásból, amelyek a BIAS számítással kapcsolatosak. Később pedig létrehozok egy különálló adathalmazt a BIAS számítással kapcsolatos ügyfelekről, amely ismétlésektől mentes.
2. Beállítom a technikai mezőket, és egyesítem az StgCpvSurveyDaily-t az 1. lépés adathalmazával. A megvalósításhoz inner join-t alkalmazok, így a staging adathalmazt a BIAS rekordok fogják szűrni.
3. Kiszámolom EF_COUNTING_RULE_FLAG-ek értékét a rekordokhoz, ami megadja, hogy egy kérdőív kitöltés eléri-e azt az elégséges pontszámot, amivel felhasználható a CPV kalkuláció során.
 - a. Megszámolom a Nokia és a versenytársak által elért pontszámokat és szűkítem az adathalmazt a szükséges mezőkre.
 - b. Eltávolítom az esetleges ismétlődéseket.
 - c. Kiszámolom az EF_COUNTING_RULE_FLAG mező értékét az alábbi formula segítségével:
$$\sqrt{(c_{NR} + c_{CR})^2 - (c_{NR} - c_{CR})^2} > 32$$
ahol c_{NR} az adott NOKIA_RATE_COUNT, míg c_{CR} a COMPETITOR_RATE_COUNT mező értéke. Hogyha az eredmény értéke nagyobb, mint 32, akkor a mező értéke 1 lesz, ellenkező esetben 0. Hasonlóképp határozom meg az EF_COUNTING_RULE_REASON mező értékét annyi különbséggel, hogy a formula baloldalának eredményét hozzáfűzöm az „EF =” karakterláncához.
 - d. Beszúrom az új rekordokat a APP_CPV.MYCPV_EFFECTIVE_HISTORY_RULE_PARAM táblába.
4. Kiszámolom az ügyfél szintű számolással kapcsolatos szabály flag-eket.
 - a. Összekötöm a BIAS számításra jelölt rekordokat az 1. lépés adathalmazával a CUSTOMER és PUBLISHED mező párosok mentén. Továbbá megszámlolom az összes pontszámot egyenként a CUSTOMER és PUBLISHED csoportonként.
 - b. Összekötöm a 3. / c. lépés eredményhalmazával.
 - c. Kiszámolom a Nokia CPV Experience, Nokia CPV Expectation, Competitor CPV Experience és Competitor CPV Expectation számolás szabály flag-eket.
5. Kiszámolom a válasz szintű számolással kapcsolatos szabály flag-eket: a Nokia CPV Experience, Nokia CPV Expectation, Competitor CPV Experience és Competitor CPV Expectation.
6. Egyesítem a válasz és az ügyfélszintű számítási szabályok kimeneteit (4. és 5. lépés).
7. Betöltöm az eredmény halmazt az APP_CPV.CPV_CALCULATION_CONTROL táblát.

8.6.7. A DF_DATA_CPV_SURVEY_RESPOND_CALCULATION adatfolyam megvalósítása

Ennek a data flow-nak a feladata a válasz szintű CPV kalkulációk elvégzése. Lásd alábbi (27.) ábra.

Az adatfolyam az alábbi módon működik:

1. Kiszámolom a Nokia és versenytárs szintű CPV értékeket. A NOKIA_CPV értéket úgy számolom ki, hogy ha a NOKIA_CPV_EXPECTANCE null értékű, akkor NOKIA_CPV_EXPECTATION kapja értékül, míg ha a NOKIA_CPV_EXPECTATION null, akkor a NOKIA_CPV_EXPECTANCE értékével lesz egyenlő. Ha pedig se a NOKIA_CPV_EXPECTANCE és a NOKIA_CPV_EXPECTATION sem null, akkor összeadom a két számot majd elosztom kettővel és ezt adom eredményül. Ugyanezzel a mechanizmussal számolom ki a COMPETITOR_CPV értéket is annyi különbséggel, hogy a COMPETITOR_CPV_EXPECTANCE és a COMPETITOR_CPV_EXPECTATION mezőket használom fel a számolás során.
2. Frissítem a APP_CPV.CPV_SURVEY_RESPOND_CALCULATION tábla értékeit.



27. ábra: A DF_DATA_CPV_SURVEY_RESPOND_CALCULATION data flow felépítése
(Forrás: saját szerkesztés)

8.6.8. A DF_DATA_CPV_CUSTOMER_CALCULATION adatfolyam megvalósítása

Ezen adatfolyam felelős az ügyfél szintű adatok kalkulációjáért. Lásd alábbi (29.) ábra.

Az adatfolyam az alábbi módon működik:

1. Megszámolom, hogy hány olyan rekord van, ahol az összes Nokiával és versenytársakkal kapcsolatos pontszám mezők kitöltésre kerültek. Ehhez felveszek két számított mezőt, azaz a NOKIA_FULL_RESPOND_FLAG és COMPETITOR_FULL_RESPOND_FLAG mezőket. A NOKIA_FULL_RESPOND_FLAG mező értéke 0 ha a VALUE_PERCEPTION_NOKIA, OVERALL_SATISFACTION_NOKIA, STRATEGIC_FIT_WITH_NOKIA és RECOMMENDATION_NOKIA mezők értéke mind null, míg ellenkező esetben 1. Ezt a mechanizmust követve határozom

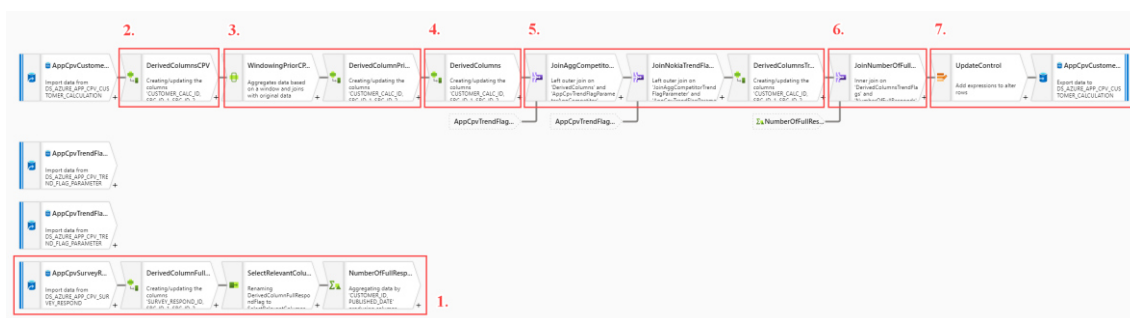
- meg a COMPETITOR_FULL_RESPOND_FLAG értékét is, annyi különbséggel, hogy a vizsgálat során a VALUE_PERCEPTION_COMPETITOR, OVERALL_SATISFACTION_COMPETITOR, STRATEGIC_FIT_WITH_COMPETITOR, RECOMMENDATION_COMPETITOR mezők értékeit nézem.
2. Kiszámolom, a NOKIA_CPV és AGGR_COMPETITOR_CPV értékeket ugyanazon az elven, mint az előző data flow-ban az 1-es lépésnél.
 3. Kiszámolom a Nokia és versenytárs prior CPV értékeket. Ehhez először particionálom a táblát CUSTOMER_ID mentén, azon belül pedig rendezem a rekordokat csökkenő sorrendben CALCULATE_DATE szerint. Létrehozok 3 részeredmény mezőt, amelyek értékét a lead függvény segítségével állítom be. A lead függvénnyel a fizikailag soron következő rekord értékeit tudom felhasználni. Ezek az mezők a NOKIA_PRIOR_CPV_ALL, ami a következő NOKIA_CPV értéket kapja, a PRIOR_PUBLISHED, ami a soron következő CALCULATE_DATE értékével lesz egyenlő és az AGGR_COMPETITOR_PRIOR_CPV_ALL, ami a következő AGGR_COMPETITOR_CPV értékét kapja meg. Végül pedig a részeredmény mezőket felhasználva határozom meg a NOKIA_PRIOR_CPV és AGGR_COMPETITOR_PRIOR_CPV eredményeket. Mindkét esetben megnézem, hogy ha a CALCULATE_DATE és PRIOR_PUBLISHED dátumok között 24 vagy annál kevesebb hónap telt el és ha igen akkor az mezőnek megfelelő CPV-ALL értéket kapja meg, ellenkező esetben null értékű lesz.
 4. Kiszámolok különböző ügyfél specifikus CPV mutatókat, melyet a 28. ábra szemléltet.

Derived column's settings	Optimize	Inspect	Data preview
Output stream name *	DerivedColumns	Learn more	
Incoming stream *	DerivedColumnPriorCPVColumnsFilter		
	+ Add	Clone	Delete Open expression builder
Column	Expression		
☐ NOKIA_CPV_PPTS	NOKIA_CPV - NOKIA_PRIOR_CPV	12f	+ 🗑
☐ NOKIA_OUTLOOK_FLAG	case(NOKIA_CPV_EXPECTATION-NOKIA_CPV_EXPE...	123	+ 🗑
☐ AGGR_COMPETITOR_CPV_PPTS	AGGR_COMPETITOR_CPV - AGGR_COMPETITOR_P...	12	+ 🗑
☐ AGGR_COMPETITOR_OUTLOOK_FLAG	case(AGGR_COMPETITOR_CPV_EXPECTATION-AGG...	123	+ 🗑
☐ DELTA_CPV	NOKIA_CPV - AGGR_COMPETITOR_CPV	12	+ 🗑
☐ PRIOR_DELTA_CPV	NOKIA_PRIOR_CPV - AGGR_COMPETITOR_PRIOR...	12f	+ 🗑
☐ DIFFERENTIATION	(NOKIA_CPV - AGGR_COMPETITOR_CPV)/(NOKIA...	12	+ 🗑
☐ PRIOR_DIFFERENTIATION	(NOKIA_PRIOR_CPV - AGGR_COMPETITOR_PRIOR...	12f	+ 🗑

28. ábra: A CPV mutatók formulái (Forrás: saját szerkesztés)

5. Összekapcsolom az adathalmazt kétszer az APP_CPV.CPV_TREND_FLAG_PARAMETER táblával: egyszer a Nokia szintű adatokkal és még egyszer az aggregált versenytársak adataival.
6. Összekapcsolom az 1. lépés eredményhalmazával.

7. Frissítem a céltáblát tartalmát az új eredményekkel.



29. ábra: A DF_DATA_CPV_CUSTOMER_CALCULATION data flow felépítése (Forrás: saját szerkesztés)

8.7. A számításokat végző és a címtár szinkronizációs modulok implementálása

Egyes üzleti igények implementálásához nem bizonyultak elegendőnek az Azure Data Factory által nyújtott funkcionalitások, ezért Azure Function segítségével oldottam meg a BIAS számítás és a címtár szinkronizáció implementálását. A BIAS kalkuláció esetében azért volt szükség különösen function-re, mert a számítás módja kiemelten szenzitív információnak minősül. Így az ügyfél előre implementálta a különböző kalkulációs metódusokat egy osztálykönyvtár segítségével, A fejlesztés során pedig a fordított osztálykönyvtár DLL állományaihoz kaptam hozzáférést, ami így feketedobozként működik az én megoldásomban. Ezt a .NET Core alapú DLL állományt pedig function implementáció segítségével tudtam a lehető leghatékonyabban megvalósítani.

Habár az ADF alkalmas API-okkal kommunikálni, egyes esetekben nem a legkényelmesebb választás a vizuális fejlesztő interfészéből fakadó limitációk végett. Mellesleg nehézkesen lehet megvalósítani a párhuzamos feldolgozást lap strukturált API-ok estében. Ezek alapján első sorban kényelmi és másodlagosan technológiai döntés volt az Azure Function használata a címtár szinkronizáció megvalósításához.

Mindkét modult egyetlen .NET Core 3.1 Azure Function projektben implementáltam és Entity Framework Core 3.1 könyvtárat használtam az Azure SQL Database-el való kommunikációhoz. Továbbá a különböző function végpontokat egy Azure Function példányról hosztoltam.

8.7.1. A BIAS kalkuláció megvalósítása

A BIAS számításához egy HTTP végpontú durable function-t implementáltam. A durable és hagyományos function-ök között az egyik lényeges különbség, hogy durable function-ök hosszabb ideig futhatnak hagyományos társaikhoz képest. Ezt az eltérő architektúráik teszik lehetővé. Egy durable function esetében mindig rendelkezünk egy indító function-nel, ami elindít egy folyamatszervező function-t, másnéven orkesztrátort. Ezután az

orkesztrátor felelős az egy vagy több activity function-ök ütemezésért, követésükért és státusz információk szolgáltatásáért.

Az implementált function-ök listája (lásd: 30. ábra):

- Submit-CpvCalculation: HTTP végpontú indító function.
- Orchestrator-CpvCalculation: vezérlésért felelős function.
- Activity-CpvCalculation: logikát végrehajtó function.



30. ábra: A BIAS kalkulációért felelős function-ök adatfolyam diagramja (Forrás: saját szerkesztés)

Ebben az esetben csak egy activity function-t implementáltam, mivel a durable function-ök hosszabb idejű futtatásra építek. Ez az activity 3 fő lépésből áll: táblák lekérdezése, kalkuláció elvégzése és adatok visszairása az adatbázisba. A logikában a BIAS_AND_ALL_CPV_CALCULATION_CONTROL_V view-t és CPV_SURVEY_RESPOND táblákat kötöm össze és kérdezem le. Ezt követően csoportosítom az adathalmazt ügyfelek mentén és elvégzem a kalkulációkat azokra a rekordokra a csoportokon belül, amelyek a vezérlő tábla megjelölt BIAS számításra. Végül frissítem az adatbázist az eredményekkel.

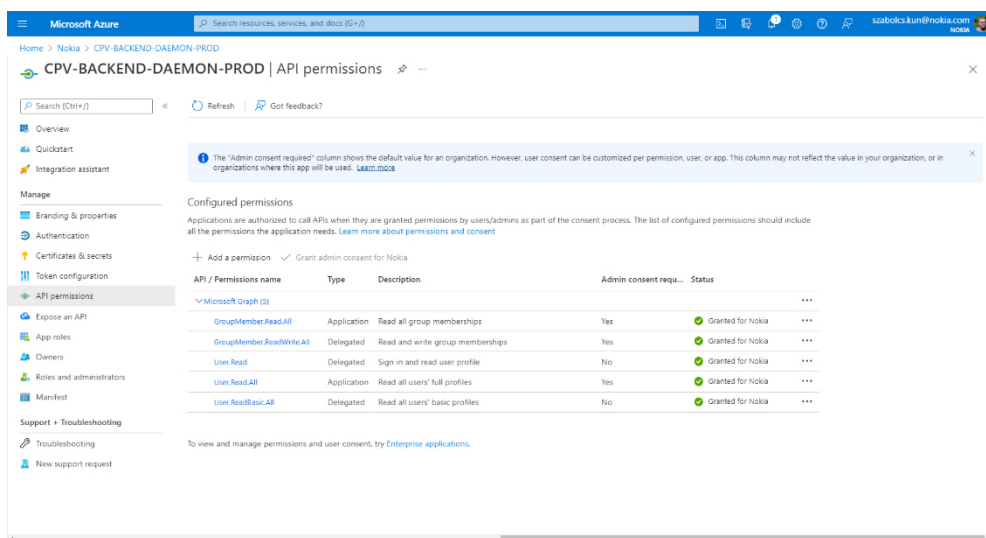
8.7.2. Címtár szinkronizáció

A hozzáférés menedzsment modulhoz szükséges adatokat a címtár szinkronizációs modul biztosítja, továbbá érvényesíti az adatbázisban történt változtatásokat a címtárban. Az Azure Active Directory címtárból a Microsoft Graph API-án keresztül nyerem ki az adatokat és vezetem át a módosításokat.

Kétirányú szinkronizációt valósítok meg, amely során egy function megadott AAD csoportok segítségével kinyerhet egy listát az összes Nokia alkalmazott nevééről, e-mail címéről és egyedi azonosítójáról. A Graph-on keresztül kinyert adatokat töltöm be az adatbázisba. Ezt a listát tudja felhasználni a hozzáférés menedzsment ahhoz, hogy egyes alkalmazottakat specifikus biztonsági csoportokhoz rendelhessenek hozzá a felhasználók. Ebben az esetben történik a visszafelé szinkronizáció, amikor egy function felveszi ezeket a hozzárendeléseket és Graph-on keresztül érvényesíti azokat az AAD címtárban. Minden esetben az AAD a „single source of truth” vagyis az kizárólagos egyedi információforrás. Ez annyit jelent, hogy bármilyen csoport hozzárendelés történhet az adatbázisban, de ha a címtárban ez alatt történt valami változás az adott entitással, mint például fiók megszűnése vagy csoportváltás, akkor az alkalmazásban történt változtatások felülíródnak az entitás jelenlegi AAD-s állapotával. Tehát két function-t implementáltam,

egyik a címtárból az adatbázisba, míg a másik az adatbázisból a címtárba történő szinkronizációért felelős. Mindkét esetben durable function-t használok, mivel szükségem van a hosszabb futásidőre és a párhuzamos feladatszerzés nyújtotta előnyökre.

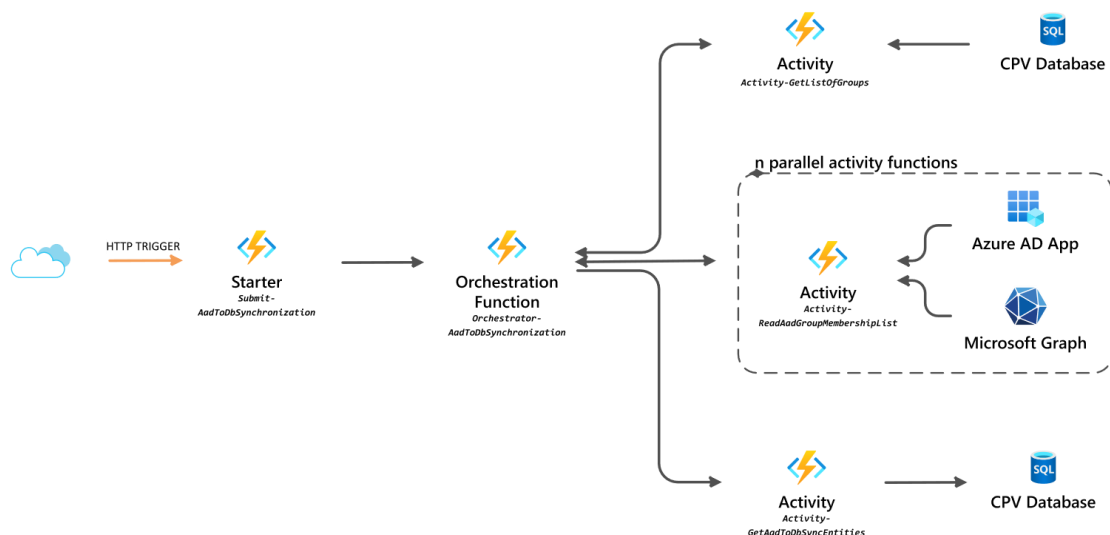
A Graph API-al való autentikációhoz létrehoztam egy Azure Active Directory Application-t (AAD APP). Az AAD APP-ok felhasználótól független identitások, amelyek különböző alkalmazások számára teszik lehetővé az autentikációt más AAD védett szolgáltatásokkal. Egy AAD APP-nak meg tudjuk adni, hogy milyen jogosultságokkal rendelkező token-t igényelhetnek tőle. Ezek a jogosultságok lehetnek beépítettek, amelyek valamilyen Microsoft szolgáltatásokhoz biztosítanak jogosultságokat és lehetnek valamilyen általunk fejlesztett alkalmazáshoz jogosultságok. Továbbá két verziójuk létezik: Application és Delegated. A Delegated jogosultságok csak felhasználói címtár fiókok számára oszthatóak ki. Ahhoz, hogy a Graph API csoportokkal kapcsolatos végpontjait használhassam, az AAD APP-on meg kellett igényelnem a GroupMember.Read.All és GroupMember.ReadWrite.All jogosultságokat a domain adminisztrátorától (lásd: 31. ábra).



31. ábra: Az AAD APP jogosultságai (Forrás: saját szerkesztés)

8.7.3. Címtárból az adatbázisba történő szinkronizáció megvalósítása

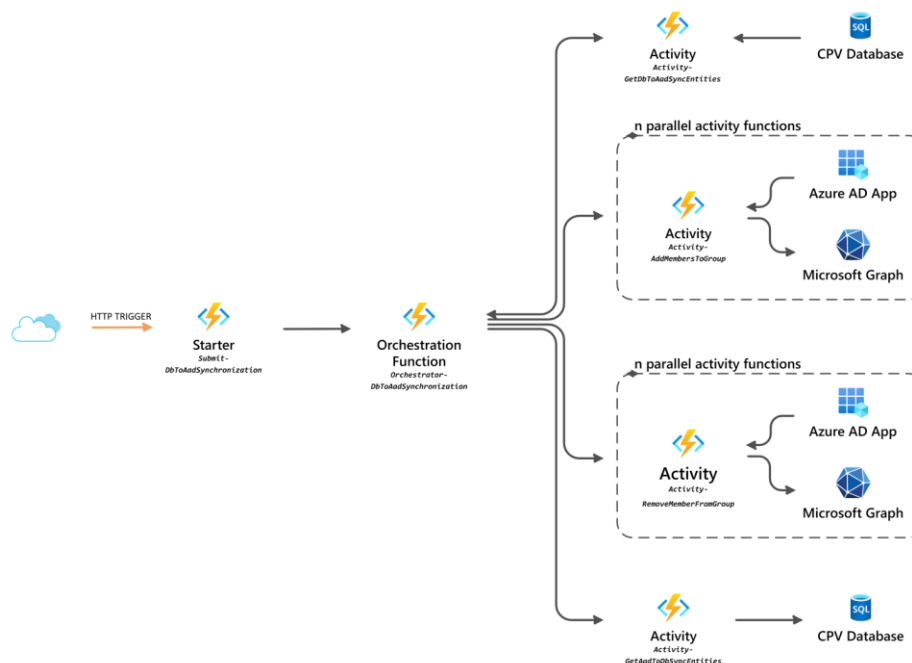
A címtár az adatbázisba történő szinkronizáció megvalósításához egy HTTP végpontú durable function-t implementáltam (lásd: 32. ábra).



32. ábra: A címtárból az adatbázisba történő szinkronizációért felelős function-ök adatfolyam diagramja (Forrás: saját szerkesztés)

1. A Submit-AadToDbSynchronization egy HTTP végpontú indító függvény, amely a végrehajtás indításáért felelős.
2. Az Orchestrator-AadToDbSynchronization nevezetű függvény felelős a feladat végző függvények vezérléséért és az eredmény átadásért. Az orkesztrátor a függvénylánc-modellt követve indítja el és valósítja meg a különböző activity függvények párhuzamos végrehajtását.
3. Elsőre az Activity-GetListOfGroups activity függvény hívódik meg, ami egy egyszerű lekérdezés segítségével lekéri az AAD csoportok listáját az adatbázisból és visszatér vele.
4. Ezután a vezérlő függvény végig iterál a csoportok listáján és párhuzamosan meghív minden csoportra egy-egy Activity-ReadAadGroupMembershipList feladat végző függvényt. Az Activity-ReadAadGroupMembershipList felel azért, hogy a paraméterül kapott csoport tagságlistáját kinyerje a Microsoft Graph-on keresztül. Először lekér egy tokenet az AAD APP-tól, hogy autentikálni tudja magát, majd pedig ezzel a tokennel és az azonosítójával hívja meg a Graph API-t. Végül pedig visszatér a Graph eredményével, ami a csoport tagjainak a listája.
5. A vezérlő egyesíti a párhuzamosan futó Activity-ReadAadGroupMembershipList függvényeknek eredményeit és meghívja az Activity-GetAadToDbSyncEntities függvényt, aminek paraméter listáján keresztül átadja az egyesített eredményt. A függvény ezután pedig aktualizálja az adatbázis rekordjait, ezzel pedig véget ér a vezérlő futása is.

Az ellentétes irányú szinkronizáció működése hasonló, a csoportokhoz történő hozzáadás és azokból eltávolítás szintén párhuzamosítva történik, az irány különbözik csak, ahogy az a 33. ábrán is látható.



33. ábra: Az adatbázisból a címtárba történő szinkronizációért felelős function-ök adatfolyam diagramja (Forrás: saját szerkesztés)

8.8. Hozzáférés menedzsment modul implementálása

A hozzáférés menedzsment modul két részből áll, egy backend API-ból és egy frontend alkalmazásból. Az API feladata, hogy a különböző csoport hozzárendeléseket és módosításokat teljesítse az adatbázisban és adatokat szolgáltatson megjelenítésre a frontend számára. Továbbá lehetővé teszi a különböző felhasználók megkötéseinek rögzítését a Power BI-ban beépített dinamikus sorszintű hozzáférésmenedzsmenthez (row-level security). Az adat lekérdező végpontok mindegyike OData végpont, míg a létrehozó, módosító és törlő végpontok hagyományos ASP.NET API végpontok.

A projekt ezen részén a backend API fejlesztése volt az én feladatom, így erről értekezem részletesebben. A backend ASP.NET Core 3.1 keretrendszerre épül és szabványos OData végpontokat implementál, amihez az OData keretrendszert használtam fel. Az OData egy nyílt protokoll, amely szabványos módon teszi lehetővé egyszerűen lekérdezhető és interoperábilis REST API-ok létrehozását. Azért esett a választásom az OData-ra, mert az üzleti igények gyakorta változása arra sarkallott, hogy valami olyan API keretrendszert használjak, amely magas rugalmasságot és személyreszabhatóságot biztosít a felhasználói interfész számára. Így legtöbb esetben elegendő egyszerű adatszolgáltató végpontokat létrehoznom és a frontend maga tudja az OData sztenderd lekérdező paraméterein keresztül szűkíteni az adathalmazt, vagy éppenséggel bővíteni azt.

Az adatbázissal való kommunikációhoz itt is – mint az Azure Function-nél – EF Core-t használtam, annyi különbséggel, hogy az 5.0-s verziót választottam. Erre azért volt szükség, mert egyes OData hibajavítások egyedül ezzel a verzióval váltak elérhetővé.

8.8.1. Végpontok

A projekt során több különböző végpontot implementáltam, viszont csak egy adatszolgáltató és egy adatmanipulációs végpontot mutatok be, mivel a legtöbb végpont működése többnyire megegyezik a felhasznált adathalmazt leszámítva. Az API az APP_CPV sémán belül az ACCESS_MANAGEMENT előtaggal rendelkező táblákkal dolgozik.

Minden adatszolgáltató végpont fix lapmérettel rendelkezik, ami megszabja, hogy egy HTTP válaszban hány rekord adható vissza egyidejűleg. Jelenleg a lapméret 100 rekordban lett meghatározva. Ha egy lekérés eredménye nem fér bele egy lapba, akkor az API automatikusan kiegészíti a válasz JSON-t egy „@odata.nextLink” nevezetű attribútummal, melynek értéke egy link, amely az azt követő lapra mutat. Ha szeretnénk a lekérdezés teljes eredményhalmazát megkapni, akkor egy ciklus segítségével végiglapozhatunk a rekordokon mindaddig, amíg el nem jutunk az utolsó laphoz, amely már nem rendelkezik egy ilyen következő lapra mutató linkkel. A lapozás egyik legnagyobb előnye, hogy erőforrás takarékos, hiszen mindig egy adott szeletével tér csak vissza az adathalmaznak. Továbbá lehetővé teszi különböző töltési stratégiák implementálását frontend oldalon, ami kifejezetten előnyös lehet mikor több ezer rekord megjelenítésre van szükség. Példa URL következő lap betöltésére: [https://app-cpv-devqa.azurewebsites.net/api/GetAllUsers?\\$skiptoken=id-161](https://app-cpv-devqa.azurewebsites.net/api/GetAllUsers?$skiptoken=id-161).

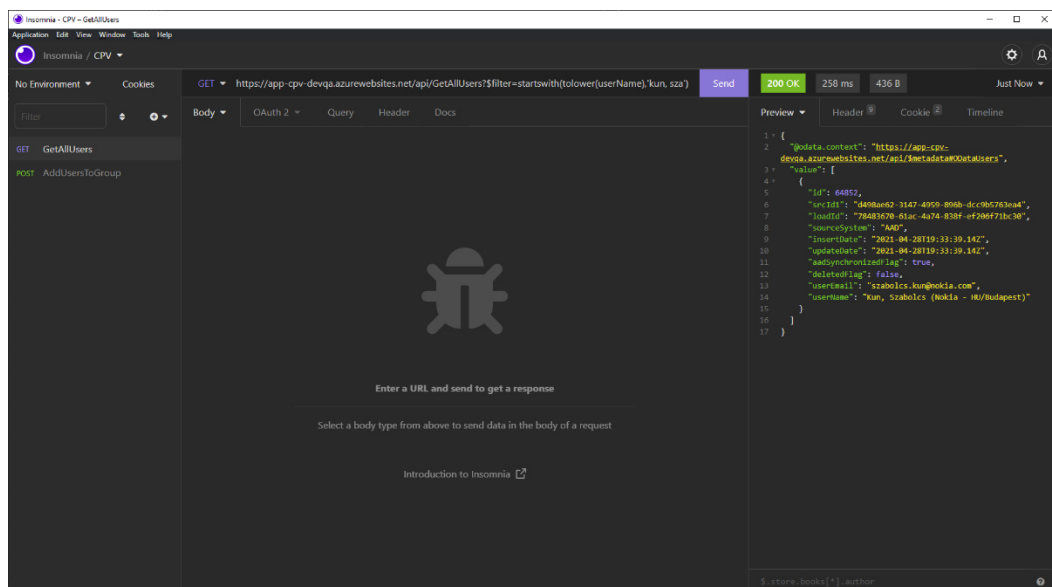
Egy HTTP válasz nem csak a visszaadandó értékkel és opcionális következő linkkel rendelkezik, hanem mindig visszatér egy „@odata.context” nevezetű attribútummal is. Ez az attribútum az adott adathalmazhoz tartozó séma metaadatára mutató linket tartalmazza, amelyet általában OData kompatibilis szoftverek részesítenek előnyben, mint például Excel vagy Power BI. A metaadat által előre meghatározhatóvá válik az API-tól várható adathalmaz sémája. Példa URL metaadat lehívására: [https://app-cpv-devqa.azurewebsites.net/api/\\$metadata#ODataUsers](https://app-cpv-devqa.azurewebsites.net/api/$metadata#ODataUsers).

8.8.2. A GetAllUsers adatszolgáltató végpont megvalósítása

Ennek a végpontnak a célja, hogy az ACCESS_MANAGEMENT_USER tábla tartalmát biztosítsa a felhasználói interfész számára. A tábla teljes tartalmát visszaadja a végpont, amihez a frontend-nek a /api/GetAllUsers relatív URL-t kell meghívnia. Ha szeretnénk szűkíteni az adathalmazt, akkor ezt megtehetjük az OData paraméterek segítségével. Ilyen paraméterek például a teljesség igénye nélkül:

- a \$select, mellyel pontosan megmondhatjuk, hogy mely mezőkre vagyunk kíváncsiak,
- a \$filter, ami által szűrhetjük az adathalmazunkat,
- a \$expand, amivel összeköthetjük a végpontunk mögött álló táblát más táblákkal,
- a \$orderBy, amely segítségével rendezhetjük az adatforrásunkat
- és a \$count, amellyel megszámlolhatjuk, hogy hány rekorddal rendelkezik a végpont.

Például ha szeretnénk lekérdezni azoknak a felhasználóknak a listáját, amelyeknek a neve „kun, sza”-val kezdődik, akkor azt megtehetjük az alábbi paraméter kombinációkkal: /api/GetAllUsers?\$filter=startswith(tolower(userName),'kun, sza') (lásd: 34. ábra). Ahogy láthatjuk, nem csak funkció paraméterek, hanem különböző függvények is a rendelkezésünkre állnak, mint például a „tolower”, ami szerver oldalon kisbetűs formára hozza a kívánt mezők értékeit.



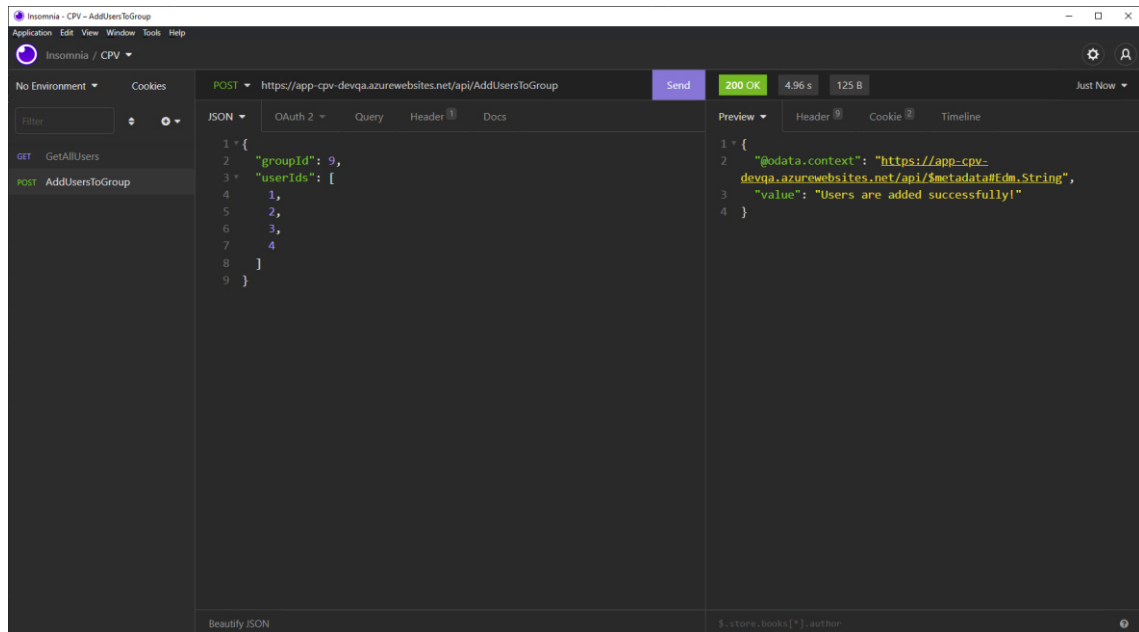
34. ábra: A GetAllUsers végpont hívása Insomnia segítségével (Forrás: saját szerkesztés)

8.8.3. Az AddUsersToGroup adatmanipulációs végpont megvalósítása

Az AddUsersToGroup végpont felelős azért, hogy egy csoporthoz egyidejűleg egy vagy több felhasználót rendelhessünk hozzá. A végpont hívásakor egy JSON objektumot vár értékül a body-jában. Ez a JSON egy groupId értékkel és egy userIds tömbbel rendelkezik. A groupId attribútum tárolja az azonosítóját annak a csoportnak, amihez szeretnénk felhasználókat hozzárendelni, míg az userIds tömb azoknak a felhasználónak az azonosítóját tartalmazza, akiket szeretnénk a csoporthoz rendelni. A hívást követően a backend ellenőrzi és feldolgozza a kérést, majd végül végrehajtja a ACCESS_MANAGEMENT_GROUP_X_USER táblán (lásd: 35. ábra). Ha sikerült a csoporthoz hozzárendelés, akkor egy 200-as válaszkóddal és egy visszaigazoló üzenettel

tér vissza. Ha a JSON sémája eltér a várttól vagy olyan csoporthoz szeretnénk felhasználót rendelni, amihez nem lehetséges, akkor a hibának megfelelő kóddal és hibaüzenettel tér vissza az API. Az alábbi hibakódok tapasztalhatóak a teljesség igénye nélkül:

- 204-es kód, ha üres a kérés body,
- 400-as kód, hogy ha a kérés body nem felel meg a strukturális elvárásoknak,
- és 401-es kód, hogy ha a felhasználó nem autentikált.



35. ábra: Az AddUsersToGroup végpont hívása Insomnia segítségével (Forrás: saját szerkesztés)

8.8.4. Autentikáció és autorizáció

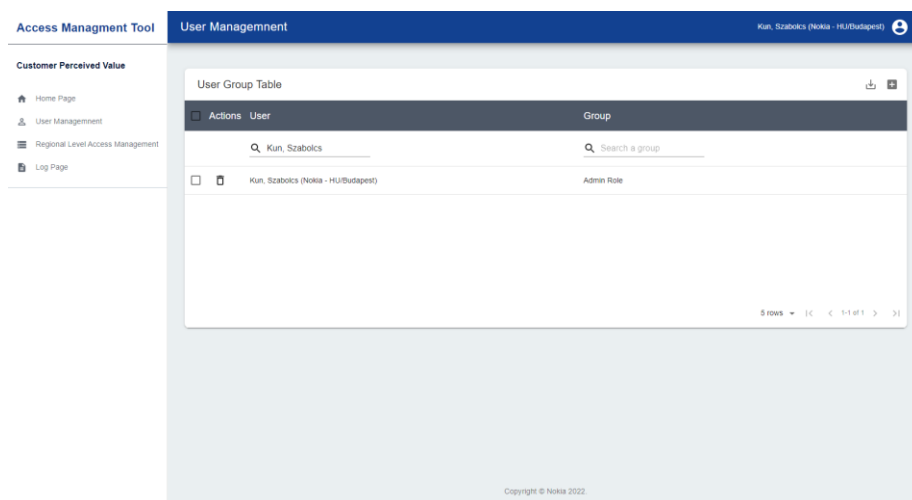
Mind a backend API, mind a frontend alkalmazás AAD autentikációval van ellátva, ezzel lehetővé téve a felhasználók azonosítását és jogosultságok kezelését. Minden végpont meghívásához rendelkezünk kell egy érvényes tokennel amit a hívások során mellékelünk kell a header-ben. Tokeneket könnyedén kérvényezhetünk az AAD API végpontjainak segítségével. Például a tesztelés során az interaktív autentikációhoz a <https://login.microsoftonline.com/nokia.com/oauth2/v2.0/authorize> API végpontot használtam, hogy tokenet kérhessek le a munkahelyi fiókkal. Kliens alkalmazások és egyéb szoftverek esetében az MSAL (Microsoft Authentication Library) nevezetű könyvtár lehet segítségemre, amellyel még egyszerűbben tudok token-eket lekérni. Csak meg kell hívni a megfelelő autentikációs folyamat metódusát a helyes konfigurációval és a könyvtár maga felépíti és végrehajtja az API hívásokat.

Az API-t csak azok a felhasználók hívhatják meg, akik tagjai az AAD_S_CPV_Admin nevezetű security group-nak. Ezt az AAD APP ellenőrzi a token kérés során; ha valaki

nem tagja a csoportnak, akkor nem kap token. Ezen az elven történik meg a felhasználók körének autorizációja.

8.8.5. Frontend

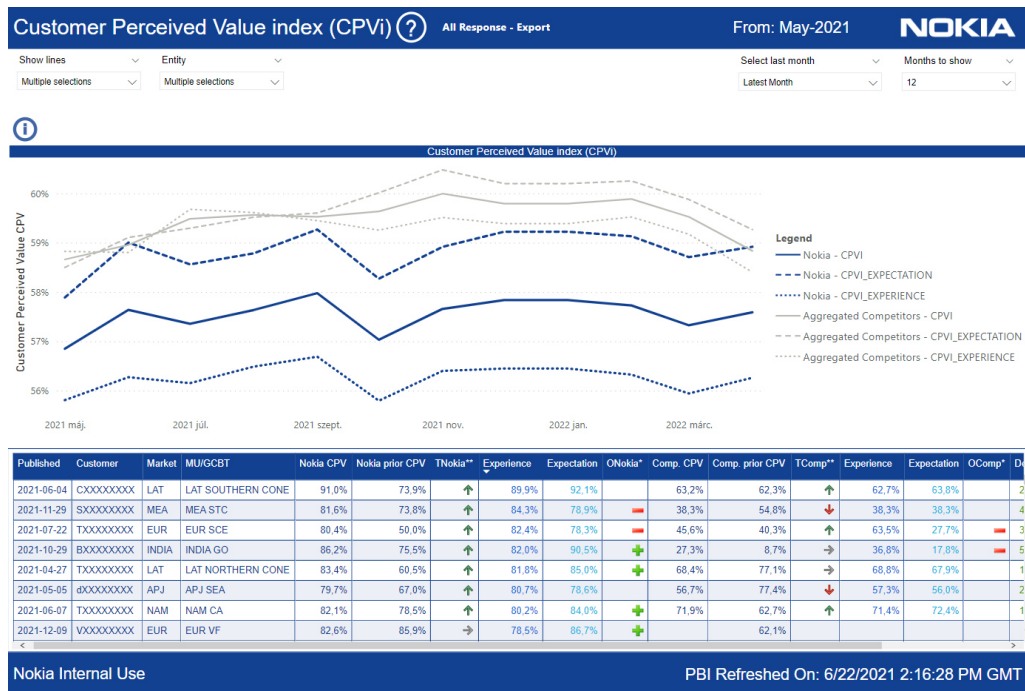
A frontend felelős a backend hívásához szükséges token lekéréséért és frissítéséért. Továbbá a backend végpontok hívásáért, adatok vizualizációjáért és kérések továbbításáért a backend felé. A frontend React-ban került lefejlesztésre és a statikus fordított állományai egy Azure Web App-ról hosztolódnak. Az alábbi (36.) ábrán látható például a felhasználók csoporthoz rendelésért felelős interfész.



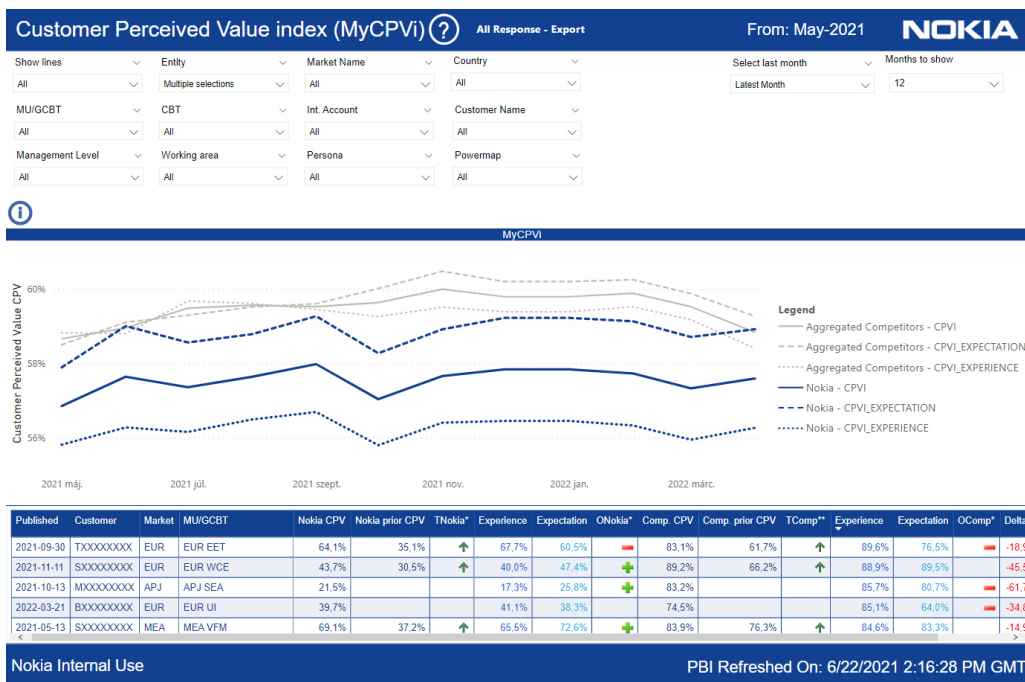
36. ábra: A frontend csoporttagságok szerkesztését kezelő felülete (Forrás: saját szerkesztés)

8.9. BI integráció

A Power BI-al könnyebb integráció érdekében különböző nézeteket implementáltunk az adatbázisban, hogy az egyes táblákat egyesítsük vagy pivotáljuk szükség szerint. Ezáltal az erőforrásigényes transzformációk az adatbázisba kerülnek végrehajtásra, ezzel is gyorsítva az adatok áttöltését a riportokba. Mivel Azure SQL Database-t használtunk, így a Power BI közvetlenül tud csatlakozni az adatbázishoz, míg földi adatbázis esetében szükség lett volna egy integrációs alkalmazásra, amely a földi környezetbe telepítésével teszi lehetővé az adatelérést. Az adatok frissítése minden nap 1 órával az adattárház töltése után, azaz hajnali 1 órakor történik UTC-5 időzóna szerint. A 37. ábrán látható a CPVi kalkulációk vizualizációja, míg a 38. ábrán a MyCPVi eredmények.



37. ábra: A CPV és CPVi eredmények vizualizációjáért felelős riport (Forrás: saját szerkesztés)

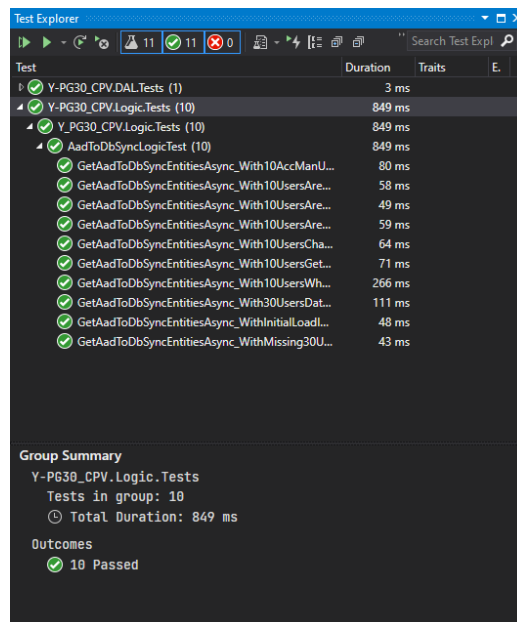


38. ábra: A MyCPVi eredmények vizualizációjáért felelős riport (Forrás: saját szerkesztés)

9. TESZTELÉS

A tesztelés elengedhetetlen rész egy fejlesztés alatt álló nagyvállalti alkalmazás életciklusában. A különböző modulokhoz különböző tesztelési stratégiákat alkalmaztam. Az adattárház eredményeinek validálásához regressziós tesztet vettem be, mivel a forrás API-ról nem csak a kérdőív kitöltések adatai, hanem a kalkulációs eredmények is elérhetőek. Így ezt fel tudtam arra használni, hogy összehasonlítsam a megoldásom által generált értékeket a forrásrendszer értékeivel.

A címtárszinkronizáció logikájának ellenőrzéséhez unit tesztet alkalmaztam, amivel biztosra tudtam menni, hogy jól implementáltam a különböző halmaz műveleteket (lásd: 39. ábra). Unit tesztek segítségével egy adott kódrész különböző dependenciáit helyettesítve tudom tesztelni, hogy egy adott bemenetre a várt kimenetet kapom-e eredményül. A backend API és a frontend tesztelése többnyire manuálisan történt, amihez bevonásra került a megrendelő oldalán felállított tesztelő csapat és közösen határoztunk meg különböző teszteseteket.



39. ábra: A címtárszinkronizáció logikáját ellenőrző unit tesztek eredményei (Forrás: saját szerkesztés)

10. ELÉRT EREDEMÉNYEK

A szakdolgozatom témája a Nokia egyik belső nagyszabású projektje volt, melynek megvalósításával sikerült modernizálni és hatékonyabbá tenni az eddig meglévő CPV platformot. Jelenleg pilot program keretein belül az alkalmazottak párhuzamosan használhatják a régi és az új megoldást, amely során visszajelzéseket gyűjtünk arról, hogy mennyire vannak megelégedve az új alkalmazásokkal, illetve min változtatnának szívesen. Két területen tapasztaltunk szignifikáns javulást a régi platformhoz képest: Visszajelzések alapján az egyik az előzőnél felhasználóbarátabb Power BI riportok, amelyet nagyobb odaadással vesznek igénybe a munkatársak. Továbbá a régi megoldásnak több, mint 4 órába telt az adatok áttöltése, illetve a különböző kalkulációk elvégzése, míg az új háttéralkalmazások átlagosan kevesebb, mint 30 perc alatt elvégzik ugyanezt a munkát, ami átszámolva 90%-os teljesítmény-megtakarításnak felel meg.

A projekt sikerességét és a megrendelők elégedettségét az is tükrözi, hogy megrendelésre került a platform második kiadása, amely során további modernizáció és a funkcionalitások bővítése kerül majd terítésre.

11. TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

A projekt éleciklusa során kollégáim és én gyakran szembesültünk azzal, hogy az igények hamar változtak, ezért sokszor kellett köztes megoldásokhoz folyamodnunk. Igyekeztünk arra törekedni, hogy egy adott pontig lefejlesztésre került állapotot csak minimálisan kelljen módosítani az új igények implementálásához. Ebből kifolyólag vannak olyan részek, ahol lehetne optimalizálni a megoldást. Ilyenek például az adatmodell és a backend API.

Az adatmodell során próbáltuk szem előtt tartani, hogy egy olyan modellt alkossunk meg, amely egyaránt képest kiszolgálni az adattárház és egyben a különböző alkalmazásokat is. Ezért egy rendkívül hibrid modell keletkezett a végére, melyet a teljes követelménylista tudatában újraterveznék. Elsősorban szétbontanám az alkalmazás sémát két részre, azaz egy kalkulációs sémára és egy adattárház sémára. A kalkulációs séma lényegében arra lenne optimalizálva, hogy a különböző kalkulációs folyamatokat támogassa meg a felépítésével, míg végül ezekből kerülne feltöltésre az adattárház séma. Ez tisztán dimenzionális adatmodellezés elvei mentén kerülne implementálására.

A backend API implementálása során az OData használata egy szükségszerű döntés volt, mivel kifejezetten képlékeny volt a megrendelők elképzelése a végleges felhasználói felületről és a szükséges funkciókról. Habár az OData kényelmes megoldást jelentett számunkra a szabad paraméterezhetőségével, egyik hátránya, hogy jelentősen erőforrásigényes. Emiatt pedig a backend API-nak egy erős számítási kapacitásra van szüksége, hogy a különböző memóriában futó OData motorokat üzemeltesse. Ennek orvoslásául kivezetném az OData-t és hagyományos adatlekérő API végpontokat implementálnék.

A projekt folytatása részben ezeknek a köztes megoldásoknak az orvoslására fókuszál majd, illetve további funkciók implementálására. Ilyen funkciók például az ajánlási motor implementálása és automatizált ügyfélkövetés, vagy a vállalatról alkotott vélemények követése az ügyfélkör közösségi médiás hálózataiban. Például, ha egy ügyfél arról posztol, hogy milyen jól sikerült egy projekt, ahol Nokia termékek kerültek felhasználásra, akkor ott nagy valószínűséggel van lehetőség további értékesítésre.

12. ÖSSZEGZÉS

A Nokia számára mindig is fontos volt, hogy innovatív, megbízható és iparágvezető hardveres és szoftveres telekommunikációs megoldásokat biztosítson. Ahhoz, hogy megőrizhesse piaci helyzetét stratégiájának részévé vált, hogy ügyfelek visszajelzéseire alapozzon, és ezek mentén szervezze folyamatait a termékfejlesztéstől a gyártáson át az értékesítésig. A fejlesztés során Microsoft termékek használatára törekedtem, mivel a Nokia számára stratégiai technológia platformnak minősülnek a Microsoft Azure és Microsoft 365 megoldások. Mivel a projekt célja egy már meglévő alkalmazás platform modernizációja és új alapokra helyezése, ezért arra törekedtem, hogy a lehető legjobban kihasználjam a rendelkezésemre álló technológiák nyújtotta lehetőségeket. Ezáltal egy olyan megoldást sikerült fejleszteni, amely beleillik a Nokia alkalmazás portfóliójába és kényelmesen használható a már meglévő alkalmazásokkal. Az új megoldással szignifikáns teljesítmény növekedést sikerült elérni a kalkulációs folyamatokban, és nem mellesleg a modern felhasználói felületekkel pedig sikerült megszerettetni a platform alkalmazásait a felhasználókkal. A sikertörténet részét képezi az is, hogy az partnereink megrendelték a projekt folytatását, melynek keretein belül továbbfejleszthetjük a platformot. A projektben különböző szakterületeken próbálhattam ki magamat, így értékes tapasztalatot gyűjthettem azzal kapcsolatban, hogy különböző szakágak hogyan tudnak egymással együtt dolgozni. Az ügyfelek Nokiáról alkotott véleményeit gyűjtő és analízáló ügyfélelégedettségi platform az új alapokra helyezéssel hatékonyá és széles körben használttá vált az értékesítéstől a termékfejlesztésen át a döntéshozatalig. Lehetőségek tárháza áll a projekt előtt, hiszen potenciálisan kiegészíthető további értékteremtő funkciókkal, mint például gépi tanúlással és automatizációs folyamatokkal.

13. SUMMARY

Nokia has always been committed to providing innovative, reliable, and industry-leading hardware and software telecommunications solutions. To maintain its market position, it has become part of its strategy to build on customer feedback and align its processes from product development to manufacturing and sales around them. In the development process, I have focused on using Microsoft products, as Microsoft Azure and Microsoft 365 solutions are considered a strategic technology platform at Nokia. As the project aims to modernize an existing application platform and put it on new foundations, I tried to make the best use of the available technologies. In this way, I was able to develop a solution that fits into Nokia's application portfolio and ensures smooth use with the existing applications landscape. With the new solution, we have achieved significant performance improvements in the calculation processes and managed to make the platform's applications more loveable to users through modern user interfaces. Another success story is the fact that our partners have ordered the continuation of the project, which gives us the opportunity to develop the platform further. This project has given me the opportunity to challenge myself in different areas of expertise, giving me valuable experience of how different professionals can work together. By developing the new customer satisfaction platform, which collects and analyzes customer experience data about Nokia, we created an effective and widely used system from sales to product development and decision-making. The project has a ton of potential as it can be enhanced with additional value-added functionalities such as machine learning and automation processes.

14. KÖSZÖNETNYILVÁNÍTÁS

Köszönettel tartozom Dr. Nagy Enikőnek a szakdolgozatom írása során végzett konzulensi munkájáért és Jambricsa Évának a Nokia IT Advanced Analytics Specialist-jének az áldozatos segítség nyújtásért adattárházak és adatmodellezés területeken. Továbbá szeretném megköszönni főnökeimnek Bianco Candidának (Nokia – Head of GFS Quality) és Rühlmann Katri-Sofiának (Nokia – Head of Global Functions and Services), hogy támogatták szakdolgozatom munkahelyi berkeken belül történő írását. Ezentúl szeretném megköszönni barátaimnak, hogy külső szemlélőként töretlenül megosztották velem észrevételeiket ezzel is segítve az szakmai és tárgyilagos szakdolgozatírást.

15. RÖVIDÍTÉSEK

- ETL: Extract, Transform, Load
- BI: Business Intelligence
- CPV: Customer Perceived Value
- KPI: key performance indicators
- CES: Customer Experience Survey
- SSO: Single Sign-On
- IaaS: Infrastructure as a Service
- PaaS: Platform as a Service
- SaaS: Software as a Service
- NIST: National Institute of Standards and Technology
- SDK: Software Development Kit
- CAPEX: Capital Expenditure
- OPEX: Operational Expenditure
- VPN: Virtual Private Network
- AD: Active Directory
- AAD: Azure Active Directory
- IT: Information Technology
- ERP: Enterprise Resource Planning
- APR: Application Portfolio Rationalization
- IIS: Internet Information Services
- REST API: Representational State Transfer Application Programming Interface
- HTTPS: Hypertext Transfer Protocol Secure
- TLS: Transport Layer Security
- ARM: Azure Resource Manager
- ADF: Azure Data Factory
- DevOps: Development and Operations
- SLA: Service Level Agreement
- API: Application Programming Interface
- SYN: Azure Synapse Analytics
- ELT: Extract, Load, Transform
- SQL: Structured Query Language
- CI/CD: Continuous Integration / Continuous Delivery
- SSIS: SQL Server Integration Services
- MPP: Massively Parallel Processing
- T-SQL: Transact-Structured Query Language
- AKS: Azure Kubernetes Service
- ACI: Azure Container Instances

- ASF: Azure Service Fabric
- HTTP: Hypertext Transfer Protocol
- FTP: File Transfer Protocol
- UDP: User Datagram Protocol
- DNS: Domain Name System
- CORS: Cross-Origin Resource Sharing
- EF: Entity Framework
- MSAL: Microsoft Authentication Library
- JWT: JSON Web Token
- JS: JavaScript
- MSI: Managed Service Identity
- DAL: Data Access Layer
- URL: Uniform Resource Locators
- IP: Internet Protocol
- IDE: Integrated Development Environment
- DDL: Data Definition Language
- DLL: Dynamic Link Library
- IR: Integration Runtime
- UTC: Coordinated Universal Time
- GUID: Globally Unique Identifier
- ID: Identifier
- SCD: Slowly Changing Dimension
- RLS: Row-level Security
- JSON: JavaScript Object Notation
- DB: Database

16. IRODALOMJEGYZÉK

- [1] J. Chudzicka, „Insourcing as a New Trend in Global Business,” *Foundations of Management*, 5. kötet, 2. szám, pp. 7-24, 2013.
- [2] P. M. Mell és P. M. Mell, „The NIST definition of cloud computing,” 2011.
- [3] D. Mangiuc, „Accountants and the cloud – Involving the professionals,” *Journal of Accounting and Management Information Systems*, 16. kötet, 1. szám, pp. 179-198, 2017.
- [4] S. Goyal, „Public vs Private vs Hybrid vs Community - Cloud Computing: A Critical Review,” *International Journal of Computer Network and Information Security*, 6. kötet, 3. szám, pp. 20-29, 2014.
- [5] K. Kaliyaperumal, „Right Positioning of Application Portfolio Rationalization and R-Lane in the Cloud Era,” DZone, 06 01 2020. [Online]. Available: <https://dzone.com/articles/right-positioning-of-application-portfolio-rationa>. [Hozzáférés dátuma: 2022. 05. 05.].
- [6] S. Orban, „6 Strategies for Migrating Applications to the Cloud | AWS Cloud Enterprise Strategy Blog,” AWS, 01 11 2016. [Online]. Available: <https://aws.amazon.com/blogs/enterprise-strategy/6-strategies-for-migrating-applications-to-the-cloud/>. [Hozzáférés dátuma: 2022. 05. 05.].
- [7] Cloudflare, „What is vendor lock-in? | Vendor lock-in and cloud computing,” [Online]. Available: <https://www.cloudflare.com/learning/cloud/what-is-vendor-lock-in/>. [Hozzáférés dátuma: 2022. 05. 05.].
- [8] Microsoft Corporation, „SLA for Azure SQL Database,” [Online]. Available: https://azure.microsoft.com/en-us/support/legal/sla/azure-sql-database/v1_6/. [Hozzáférés dátuma: 2022. 05. 05.].
- [9] Microsoft Corporation, „SLA for App Service,” [Online]. Available: https://azure.microsoft.com/en-us/support/legal/sla/app-service/v1_4/. [Hozzáférés dátuma: 2022. 05. 05.].
- [10] Microsoft Corporation, „Service Level Agreements (SLA) for Online Services,” [Online]. Available: <https://www.microsoft.com/licensing/docs/view/Service-Level-Agreements-SLA-for-Online-Services?isToggleToList=True&lang=16&year=2021>. [Hozzáférés dátuma: 2022. 05. 05.].

- [11] Microsoft Corporation, „Introduction to Azure Data Factory - Azure Data Factory,” [Online]. Available: <https://docs.microsoft.com/en-us/azure/data-factory/introduction>. [Hozzáférés dátuma: 2022. 05. 05.].
- [12] Microsoft Corporation, „What is Azure Synapse Analytics? - Azure Synapse Analytics,” [Online]. Available: <https://docs.microsoft.com/en-us/azure/synapse-analytics/overview-what-is>. [Hozzáférés dátuma: 2022. 05. 05.].
- [13] Microsoft Corporation, „Overview - Azure App Service,” [Online]. Available: <https://docs.microsoft.com/en-us/azure/app-service/overview>. [Hozzáférés dátuma: 2022. 05. 05.].
- [14] Microsoft Corporation, „Azure Functions Overview,” [Online]. Available: <https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview>. [Hozzáférés dátuma: 2022. 05. 05.].
- [15] Microsoft Corporation, „Durable Functions Overview - Azure,” [Online]. Available: <https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview>. [Hozzáférés dátuma: 2022. 05. 05.].
- [16] Microsoft Corporation, „What is the Azure SQL Database service? - Azure SQL Database,” [Online]. Available: <https://docs.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview?view=azuresql>. [Hozzáférés dátuma: 2022. 05. 05.].
- [17] Microsoft Corporation, „Datasets in Azure Data Factory and Azure Synapse Analytics,” [Online]. Available: <https://docs.microsoft.com/en-us/azure/data-factory/concepts-datasets-linked-services>. [Hozzáférés dátuma: 2022. 05. 05.].
- [18] B. Hámori, V. Beáta, L. Tóth, A. Derecskei és S. Prónay, *Érzelmek és indulatok a gazdaságban - A gazdasági szereplők viselkedésének sajátosságai a döntésekben és folyamatokban*, Szegedi Tudományegyetem Gazdaságtudományi Kar.
- [19] Microsoft Corporation, „Managed identities for Azure resources,” [Online]. Available: <https://docs.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/overview>. [Hozzáférés dátuma: 2022. 05. 05.].
- [20] Microsoft Corporation, „OAuth 2.0 client credentials flow on the Microsoft identity platform,” [Online]. Available: <https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-client-creds-grant-flow>. [Hozzáférés dátuma: 2022. 05. 05.].

17. MELLÉKLETEK

- **1. számú melléklet:** Az infrastruktúra architektúrájának diagramja (Forrás: saját szerkesztés)
- **2. számú melléklet:** Az adatbázis séma diagramja (Forrás: saját szerkesztés)
- **3. számú melléklet:** A DF_CPV_CUSTOMER_AND_REGION_LEVEL_ACCESS_MANAGEMENT data flow felépítése (Forrás: saját szerkesztés)
- **4. számú melléklet:** A DF_DATA_CPV_CALCULATION_CONTROL_CPV data flow felépítése (Forrás: saját szerkesztés)

18. ÁBRAJEGYZÉK

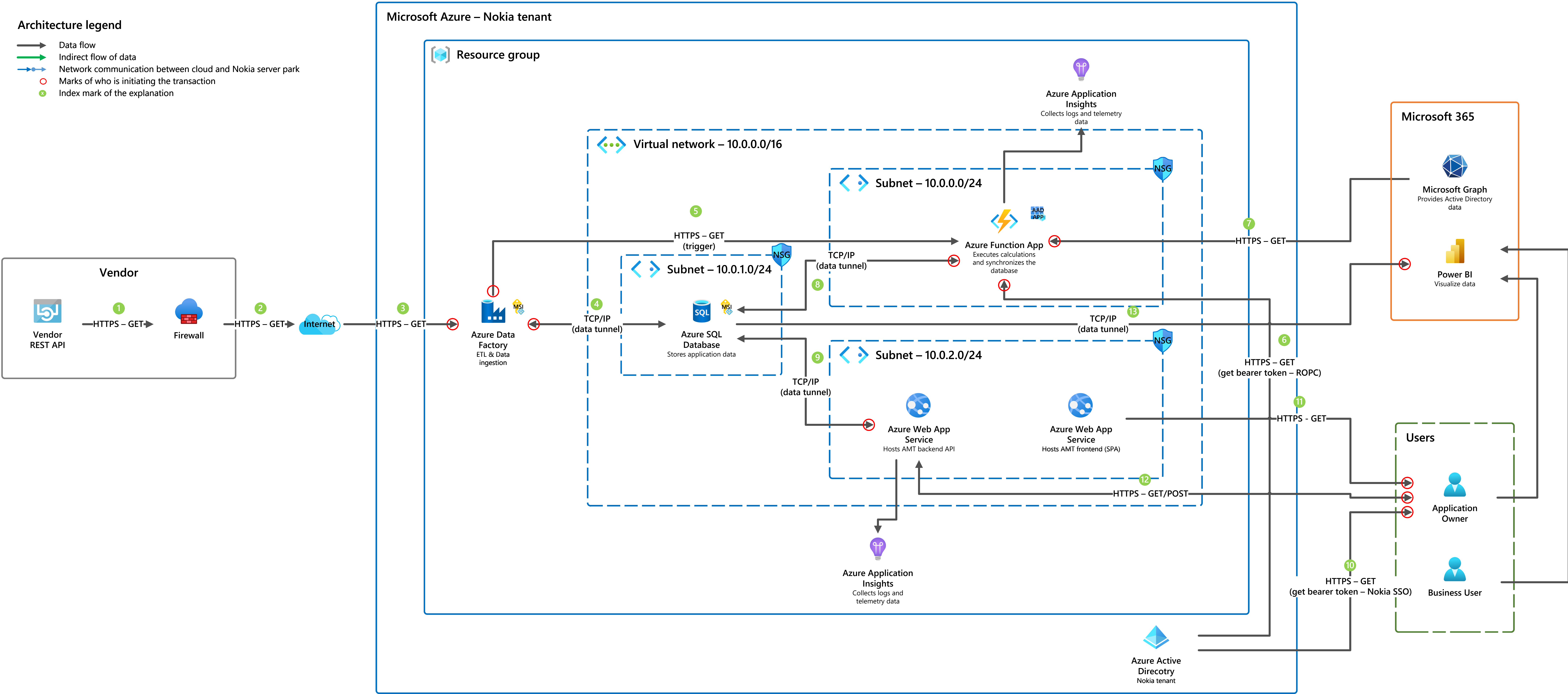
1. ábra: A Nokia értékeinek filozófiadaraja (Forrás: saját szerkesztés).....	14
2. ábra: A szolgáltatás modellek összehasonlítása felelősség szempontjából (Forrás: saját szerkesztés)	19
3. ábra: Az APR stratégia felhőre tovább dolgozott változatának folyamatábrája (Forrás: [5] alapján saját szerkesztés).....	23
4. ábra: Alkalmazások életciklusa az Azure DevOps-ban (Forrás: saját szerkesztés)	29
5. ábra: Példa Azure Function létrehozására a felhasználói felületen keresztül a szolgáltatás varázsló segítségével (Forrás: saját szerkesztés).....	41
6. ábra: A kódbázis projektjei közötti függőségek (Forrás: saját szerkesztés).....	42
7. ábra: Példa pipeline-ra (Forrás: saját szerkesztés)	44
8. ábra: Példa másolás activity-re (Forrás: saját szerkesztés)	44
9. ábra: Példa dataset-re: egy Azure SQL Database dataset (Forrás: saját szerkesztés) .	44
10. ábra: Példa data flow-ra (Forrás: saját szerkesztés)	45
11. ábra: Linked service szerkesztésére és létrehozására szolgáló varázsló (Forrás: saját szerkesztés)	45
12. ábra: Trigger létrehozására szolgáló varázsló (Forrás: saját szerkesztés).....	46
13. ábra: Kapcsolatok ADF objektumok között [17]	47
14. ábra: A pipeline-ok közötti függőségek (Forrás: saját szerkesztés)	47
15. ábra: A PL_DATA_CPV_BASE pipeline felépítése (Forrás: saját szerkesztés)	48
16. ábra: A PL_DATA_APP_BIAS_CPV_CALCULATION és PL_DATA_APP_CPVI_CALCULATION pipeline-ok hívása (Forrás: saját szerkesztés)	50
17. ábra: A PL_DATA_APP_CPVI_CALCULATION pipeline hívása (Forrás: saját szerkesztés)	50
18. ábra: A PL_DATA_APP_BIAS_CPV_CALCULATION pipeline felépítése (Forrás: saját szerkesztés).....	50
19. ábra: A function státuszának ellenőrzése	51
20. ábra: A DF_DATA_CPVI_ALL_CALCULATION meghívása (Forrás: saját szerkesztés)	51
21. ábra: A PL_DATA_CPV_AAD_TO_DB_SYNC pipeline felépítése (Forrás: saját szerkesztés)	52

22. ábra: A PL_DATA_CPV_DB_TO_AAD_SYNC pipeline felépítése (Forrás: saját szerkesztés)	52
23. ábra: A DF_DATA_CPV_COMPETITOR data flow felépítése (Forrás: saját szerkesztés)	53
24. ábra: A DF_DATA_CPV_SURVEY_RESPONDENT_NAME data flow felépítése (Forrás: saját szerkesztés)	56
25. ábra: A DF_DATA_CPV_SURVEY_RESPOND data flow felépítése (Forrás: saját szerkesztés)	56
26. ábra: A DF_DATA_CPV_CALCULATION_CONTROL_BIAS data flow felépítése (Forrás: saját szerkesztés)	57
27. ábra: A DF_DATA_CPV_SURVEY_RESPOND_CALCULATION data flow felépítése (Forrás: saját szerkesztés)	59
28. ábra: A CPV mutatók formulái (Forrás: saját szerkesztés)	60
29. ábra: A DF_DATA_CPV_CUSTOMER_CALCULATION data flow felépítése (Forrás: saját szerkesztés)	61
30. ábra: A BIAS kalkulációért felelős function-ök adatfolyam diagramja (Forrás: saját szerkesztés)	62
31. ábra: Az AAD APP jogosultságai (Forrás: saját szerkesztés)	63
32. ábra: A címtárból az adatbázisba történő szinkronizációért felelős function-ök adatfolyam diagramja (Forrás: saját szerkesztés)	64
33. ábra: Az adatbázisból a címtárba történő szinkronizációért felelős function-ök adatfolyam diagramja (Forrás: saját szerkesztés)	65
34. ábra: A GetAllUsers végpont hívása Insomnia segítségével (Forrás: saját szerkesztés)	67
35. ábra: Az AddUsersToGroup végpont hívása Insomnia segítségével (Forrás: saját szerkesztés)	68
36. ábra: A frontend csoporttagságok szerkesztését kezelő felülete (Forrás: saját szerkesztés)	69
37. ábra: A CPV és CPVi eredmények vizualizációjáért felelős riport (Forrás: saját szerkesztés)	70
38. ábra: A MyCPVi eredmények vizualizációjáért felelős riport (Forrás: saját szerkesztés)	70
39. ábra: A címtárszinkronizáció logikáját ellenőrző unit tesztek eredményei (Forrás: saját szerkesztés)	71

1. számú melléklet: Az infrastruktúra architektúrájának diagramja
(Forrás: saját szerkesztés)

Adatvezérelt folyamatos ügyfél-elégedettségi platform fejlesztése és egységesítése az üzleti döntéshozatal támogatásához a Nokia Solutions and Networks Kft.-nél

Kun Szabolcs István



2. számú melléklet: Az adatbázis séma diagramja (Forrás: saját szerkesztés)

Adatvezérelt folyamatos ügyfél-elégedettségi platform fejlesztése és egységesítése az üzleti döntéshozatal támogatásához a Nokia Solutions and Networks Kft.-nél

Kun Szabolcs István

Staging

CPV_SURVEY_DAILY	
CUSEID	
RESPONDENT	
CUSTOMER	
MARKET	
MU_GCBT	
CBT	
INT_ACCOUNT	
COUNTRY	
TYPE_OF_INTERVIEW	
MGT_LEVEL	
PERSONA	
POWERMAP	
WORKING_AREA	
COMPETITOR	
VALUE_PERCEPTION_NOKIA	
OVERALL_SATISFACTION_NOKIA	
STRATEGIC_FIT_WITH_NOKIA	
RECOMMENDATION_NOKIA	
VALUE_PERCEPTION_COMPETITOR	
OVERALL_SATISFACTION_COMPETITOR	
STRATEGIC_FIT_WITH_COMPETITOR	
RECOMMENDATION_COMPETITOR	
PUBLISHED	
SAP_ID	
JOB_TITLE	
INTERVIEW_DATE	
NOKIA_CPV	
NOKIA_CPV_EXPECTATION	
NOKIA_CPV_EXPERIENCE	
COMPETITOR_CPV	
COMPETITOR_EXPECTATION	
COMPETITOR_EXPERIENCE	

Application Layer

CPV(i) tables

CPV_NOKIA_ALL_LEVEL_CALCULATION	
PK	CPV_NOKIA_ALL_CALC_ID
U1	SRC_ID_1
U1	SRC_ID_2
U1	SRC_ID_3
U1	SRC_ID_4
	LOAD_ID
	CALCULATE_DATE
	CALCULATE_YEAR
	CALCULATE_MONTH
	CALCULATE_YEARMONTH
	CALCULATION_LEVEL_NAME
	CALCULATION_LEVEL_VALUE
	NUMBER_OF_RESPONDS
	NOKIA_CPV
	NOKIA_CPV_EXPECTATION
	NOKIA_CPV_EXPERIENCE
	NOKIA_PRIOR_CPV
	NOKIA_CPV_PPTS
	NOKIA_TREND_FLAG
	NOKIA_OUTLOOK_FLAG
	NOKIA_NUMBER_OF_FULL_RESPONDS

CPV_COMPETITOR_ALL_LEVEL_CALCULATION	
PK	CPV_COMPETITOR_ALL_CALC_ID
U1	SRC_ID_1
U1	SRC_ID_2
U1	SRC_ID_3
U1	SRC_ID_4
	LOAD_ID
	COMPETITOR_ID
	CALCULATE_DATE
	CALCULATE_YEAR
	CALCULATE_MONTH
	CALCULATE_YEARMONTH
	CALCULATION_LEVEL_NAME
	CALCULATION_LEVEL_VALUE
	NUMBER_OF_RESPONDS
	COMPETITOR_CPV
	COMPETITOR_CPV_EXPECTATION
	COMPETITOR_CPV_EXPERIENCE
	COMPETITOR_PRIOR_CPV
	COMPETITOR_CPV_PPTS
	COMPETITOR_TREND_FLAG
	COMPETITOR_OUTLOOK_FLAG
	DELTA_CPV
	PRIOR_DELTA_CPV
	DIFFERENTIATION
	PRIOR_DIFFERENTIATION
	COMPETITOR_NUMBER_OF_FULL_RESPONDS

CPV_SURVEY_RESPOND_CALCULATION	
PK	RESPOND_CALC_ID
U1	SRC_ID_1
U1	SRC_ID_2
U1	SRC_ID_3
U1	LOAD_ID
	SURVEY_RESPOND_ID
	CALCULATE_DATE
	CALCULATE_YEAR
	CALCULATE_MONTH
	NOKIA_CPV
	NOKIA_CPV_EXPECTATION
	COMPETITOR_CPV
	COMPETITOR_CPV_EXPECTATION

CPV_CUSTOMER_CALCULATION	
PK	CUSTOMER_CALC_ID
U1	SRC_ID_1
U1	SRC_ID_2
U1	SRC_ID_3
U1	LOAD_ID
	CUSTOMER_ID
	CALCULATE_DATE
	CALCULATE_YEAR
	CALCULATE_MONTH
	NUMBER_OF_RESPONDS
	NOKIA_CPV
	NOKIA_CPV_EXPECTATION
	NOKIA_CPV_EXPERIENCE
	NOKIA_PRIOR_CPV
	NOKIA_CPV_PPTS
	NOKIA_TREND_FLAG
	NOKIA_OUTLOOK_FLAG
	AGGR_COMPETITOR_CPV
	AGGR_COMPETITOR_CPV_EXPECTATION
	AGGR_COMPETITOR_CPV_EXPERIENCE
	AGGR_COMPETITOR_PRIOR_CPV
	AGGR_COMPETITOR_CPV_PPTS
	AGGR_COMPETITOR_TREND_FLAG
	AGGR_COMPETITOR_OUTLOOK_FLAG
	DELTA_CPV
	PRIOR_DELTA_CPV
	DIFFERENTIATION
	PRIOR_DIFFERENTIATION
	NOKIA_NUMBER_OF_FULL_RESPONDS
	AGGR_COMPETITOR_NUMBER_OF_FULL_RESPONDS

BIAS_AND_ALL_CPV_CALCULATION_CONTROL_V	
	CALCULATION_DATE
	CALCULATION_YEAR
	CALCULATION_MONTH
	CALCULATION_YEARMONTH
	SURVEY_RESPOND_ID
	CUSTOMER_ID
	CUSTOMER_SRC_ID_1
	CUSTOMER_SRC_ID_2
	CUSTOMER_SRC_ID_3
	SURVEY_RESPOND_SRC_ID_1
	SURVEY_RESPOND_SRC_ID_2
	SURVEY_RESPOND_SRC_ID_3
	SURVEY_RESPOND_LOAD_ID
	BIAS_RULE_FLAG
	CUSTOMER_CPV_EXPECTATION_NOKIA_FLAG
	CUSTOMER_CPV_EXPECTATION_NOKIA_FLAG
	CUSTOMER_CPV_EXPECTATION_COMP_FLAG
	RESPONDENT_CPV_EXPECTATION_NOKIA_FLAG
	RESPONDENT_CPV_EXPECTATION_NOKIA_FLAG
	RESPONDENT_CPV_EXPECTATION_COMP_FLAG
	RESPONDENT_EXPECTATION_COMP_FLAG

CPV_CALCULATION_CONTROL	
	CALCULATION_DATE
	CALCULATION_YEAR
	CALCULATION_MONTH
	CALCULATION_YEARMONTH
	CALCULATION_INVOLVED
	CALCULATION_LEVEL_NAME
	CALCULATION_LEVEL_VALUE
	CALCULATED_ENTITY
	COUNTING_RULE_FLAG
	COUNTING_RULE_REASON
	CPV_INTERVAL_START_DATE
	CPV_INTERVAL_END_DATE
	COMPETITOR_ID

CPV_MONTH_INTERVALS_V	
	YEARMONTH
	INTERFRAME_IN_MONTH
	INTERVAL_START_DATE
	INTERVAL_END_DATE

CPV_YEARMONTH_INTERVAL_PARAM	
	YEARMONTH
	INTERFRAME_IN_MONTH
	INTERVAL_START_DATE
	INTERVAL_END_DATE

CPV_TREND_FLAG_PARAMETER	
	CPV_RATE_BOTTOM
	CPV_RATE_TOP
	UPPER_TRESHOLD_PPTS
	LOWER_TRESHOLD_PPTS

CPV_HIERARCHY_LEVELS_V	
	LEVEL_NAME
	LEVEL_VALUE

MYCPV_EFFECTIVE_HISTORY_RULE_PARAM	
	PUBLISHED_DATE
	CUSTOMER_NAME
	EF_COUNTING_RULE_FLAG
	EF_COUNTING_RULE_REASON

ACCESS_MANAGEMENT_CUSTOMER_HIERARCHY_V	
	MARKET_NAME
	MU_GCBT
	COUNTRY
	CUSTOMER_NAME

Base tables

CPV_COMPETITOR_HIST	
	COMPETITOR_ID
	SRC_ID_1
	LOAD_ID
	VALID_FROM
	VALID_TO
	ORIGINAL_COMPETITOR_NAME
	COMPETITOR_NAME

CPV_COMPETITOR	
PK	COMPETITOR_ID
U1	SRC_ID_1
	LOAD_ID
	VALID_FROM
	VALID_TO
	ORIGINAL_COMPETITOR_NAME
	COMPETITOR_NAME

CPV_SURVEY_RESPONDENT_NAME_HIST	
	RESPONDENT_ID
	SRC_ID_1
	SRC_ID_2
	VALID_FROM
	VALID_TO
	RESPONDENT_NAME

CPV_SURVEY_RESPONDENT_NAME	
PK	RESPONDENT_ID
U1	SRC_ID_1
U1	SRC_ID_2
	VALID_FROM
	VALID_TO
	LOAD_ID
	RESPONDENT_NAME

CPV_SURVEY_RESPOND	
PK	SURVEY_RESPOND_ID
U1	SRC_ID_1
U1	SRC_ID_2
	LOAD_ID
	VALID_FROM
	VALID_TO
	RESPONDENT_ID
	CUSTOMER_ID
	COMPETITOR_ID
	CUSEID
	PUBLISHED_DATE
	INTERVIEW_DATE
	TYPE_OF_INTERVIEW
	MGT_LEVEL
	PERSONA
	POWERMAP
	WORKING_AREA
	JOB_TITLE
	VALUE_PERCEPTION_NOKIA
	OVERALL_SATISFACTION_NOKIA
	STRATEGIC_FIT_WITH_NOKIA
	BIAS_VALUE_PERCEPTION_NOKIA
	BIAS_OVERALL_SATISFACTION_NOKIA
	BIAS_STRATEGIC_FIT_NOKIA
	BIAS_VALUE_PERCEPTION_COMPETITOR
	BIAS_RECOMMENDATION_NOKIA
	OVERALL_SATISFACTION_COMPETITOR
	STRATEGIC_FIT_WITH_COMPETITOR
	RECOMMENDATION_COMPETITOR
	BIAS_VALUE_PERCEPTION_COMPETITOR
	BIAS_OVERALL_SATISFACTION_COMPETITOR
	BIAS_STRATEGIC_FIT_COMPETITOR
	BIAS_RECOMMENDATION_COMPETITOR

CPV_CUSTOMER	
PK	CUSTOMER_ID
U1	SRC_ID_1
	LOAD_ID
	VALID_FROM
	VALID_TO
	ORIGINAL_CUSTOMER_NAME
	CUSTOMER_NAME
	MARKET_NAME
	MU_GCBT
	CBT
	INT_ACCOUNT
	COUNTRY
	SAP_ID

CPV_CUSTOMER_HIST	
	CUSTOMER_ID
	SRC_ID_1
	LOAD_ID
	VALID_FROM
	VALID_TO
	ORIGINAL_CUSTOMER_NAME
	CUSTOMER_NAME
	MARKET_NAME
	MU_GCBT
	CBT
	INT_ACCOUNT
	COUNTRY
	SAP_ID

CPV_CUSTOMER_X_CUSTOMER	
PK	CUSTXCUST_ID
U1	SRC_ID_1
U1	SRC_ID_2
	LOAD_ID
	CHILD_CUSTOMER_ID
	PARENT_CUSTOMER_ID
	RELATIONSHIP_TYPE

Access Management

ACCESS_MANAGEMENT_GROUP	
PK	GROUP_ID
U1	SRC_ID_1
	LOAD_ID
	INSERT_DATE
	UPDATE_DATE
	AAD_SYNCHRONIZED_FLAG
	DELETED_FLAG
	GROUP_NAME
	GROUP_DESCRIPTION
	SOURCE_SYSTEM

ACCESS_MANAGEMENT_GROUP_X_USER	
PK	GROUPXUSER_ID
U1	SRC_ID_1
U1	SRC_ID_2
	LOAD_ID
	SOURCE_SYSTEM
	INSERT_DATE
	UPDATE_DATE
	DELETED_FLAG
	AAD_SYNCHRONIZED_FLAG
	GROUP_ID
	USER_ID
	RELATIONSHIP_TYPE

ACCESS_MANAGEMENT_USER	
PK	USER_ID
U1	SRC_ID_1
	LOAD_ID
	SOURCE_SYSTEM
	INSERT_DATE
	UPDATE_DATE
	DELETED_FLAG
	AAD_SYNCHRONIZED_FLAG
	USER_EMAIL
	USER_NAME

ACCESS_MANAGEMENT_REGION_LEVEL_RLS	
PK	REGION_LEVEL_RLS_ID
U1	SRC_ID_1
U1	SRC_ID_2
U1	SRC_ID_3
	USER_ID
	ACCESS_LEVEL
	ACCESS_VALUE
	ACCESS_GRANTED
	ACCESS_REVOKED

ACCESS_MANAGEMENT_AUDIT_TRAIL	
PK	LOG_ID
U1	TABLE_NAME
	USER_AAD_ID
	USER_NAME
	AFFECTED_USER_AAD_ID
	AFFECTED_USER_NAME
	EVENT_TYPE
	EVENT_DETAIL
	EVENT_DATETIME

Reporting Layer

CPV_CUSTOMER_CALCULATION_V	
	CUSTOMER_CALC_ID
	CUSTOMER_ID
	PUBLISHED_DATE
	PRIOR_PUBLISHED_DATE
	PUBLISHED_MONTH_LAST_DAY
	PUBLISHED_YEARMONTH
	NUMBER_OF_RESPONDS
	NOKIA_CPV
	NOKIA_PRIOR_CPV
	NOKIA_CPV_EXPECTATION
	NOKIA_PRIOR_CPV_EXPECTATION
	NOKIA_TREND_FLAG
	NOKIA_OUTLOOK_FLAG
	NOKIA_NUMBER_OF_FULL_RESPONDS
	COMPETITOR_CPV
	COMPETITOR_PRIOR_CPV
	COMPETITOR_CPV_EXPECTATION
	COMPETITOR_PRIOR_CPV_EXPECTATION
	COMPETITOR_TREND_FLAG
	COMPETITOR_OUTLOOK_FLAG
	COMPETITOR_NUMBER_OF_FULL_RESPONDS
	DELTA_CPV
	PRIOR_DELTA_CPV
	DIFFERENTIATION
	PRIOR_DIFFERENTIATION

CPV_CUSTOMER_V	
	CUSTOMER_ID
	CUSTOMER_NAME
	MARKET_NAME
	MU_GCBT
	CBT
	INT_ACCOUNT
	COUNTRY
	SAP_ID

CPV_COMPETITOR_V	
	COMPETITOR_ID
	COMPETITOR_NAME
	ORIGINAL_COMPETITOR_NAME

CPV_CUSTOMER_DIRECT_V	
	USER_EMAIL
	CUSTOMER_ID

CPV_RESPONDENT_NAME_W_RLS_GROUP_V	
	SURVEY_RESPOND_ID
	RESPONDENT_NAME
	RLS_GROUP

MYCPV_SURVEY_RESPOND_V	
	YEARMONTH
	INTERVAL_START_DATE
	INTERVAL_END_DATE
	CUSTOMER_ID
	COMPETITOR_NAME
	WORKING_AREA
	PERSONA
	POWERMAP
	MGT_LEVEL
	SURVEY_RESPOND_ID
	PUBLISHED_DATE
	BIAS_VALUE_PERCEPTION_NOKIA
	BIAS_OVERALL_SATISFACTION_NOKIA
	BIAS_STRATEGIC_FIT_NOKIA
	BIAS_VALUE_PERCEPTION_COMPETITOR
	BIAS_OVERALL_SATISFACTION_COMPETITOR
	BIAS_STRATEGIC_FIT_COMPETITOR
	BIAS_RECOMMENDATION_COMPETITOR

CPV_ALL_LEVEL_CALCULATION_V	
	CALCULATION_DATE
	CALCULATION_YEAR
	CALCULATION_MONTH
	CALCULATION_YEARMONTH
	CALCULATION_LEVEL_NAME
	CALCULATION_LEVEL_VALUE
	NUMBER_OF_RESPONDS
	COMPETITOR_NUMBER_OF_FULL_RESPONDS
	COMPETITOR_CPV
	COMPETITOR_NUMBER_OF_RESPONDS
	COMPETITOR_CPV_EXPECTATION
	COMPETITOR_PRIOR_CPV
	COMPETITOR_TREND_FLAG
	COMPETITOR_OUTLOOK_FLAG
	NOKIA_NUMBER_OF_RESPONDS
	NOKIA_CPV
	NOKIA_CPV_EXPECTATION
	NOKIA_CPV_EXPERIENCE
	NOKIA_PRIOR_CPV
	NOKIA_TREND_FLAG
	NOKIA_OUTLOOK_FLAG
	NOKIA_NUMBER_OF_FULL_RESPONDS
	DELTA_CPV
	PRIOR_DELTA_CPV
	DIFFERENTIATION
	PRIOR_DIFFERENTIATION

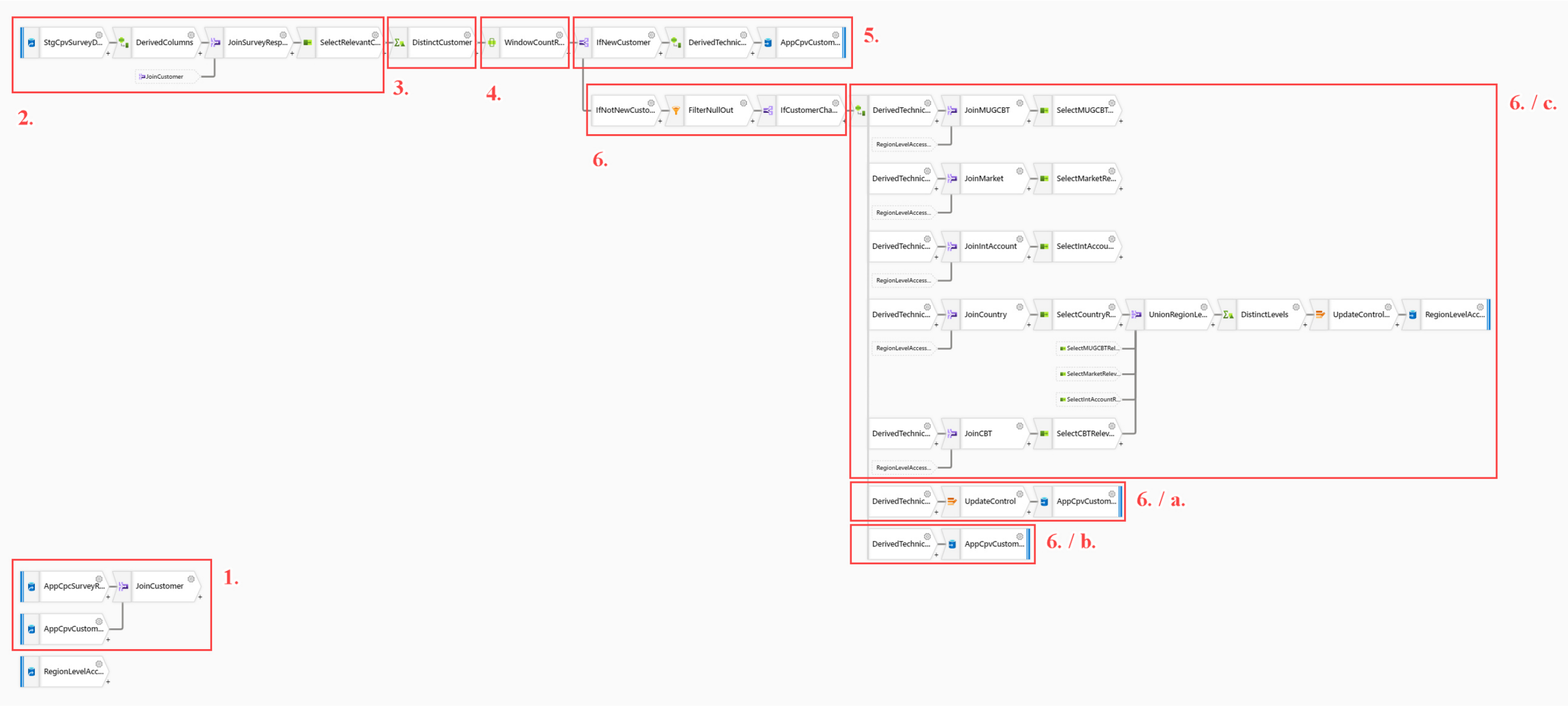
MYCPV_EFFECTIVE_HISTORY_RULE_PARAM_V	
	PUBLISHED_DATE
	CUSTOMER_NAME
	EF_COUNTING_RULE_FLAG
	EF_COUNTING_RULE_REASON

MYCPV_SURVEY_RESPOND_ALL_LEVEL_V	
	YEARMONTH
	INTERVAL_START_DATE
	INTERVAL_END_DATE
	MEASURE_TYPE
	SURVEY_RESPOND_ID
	PUBLISHED_DATE
	CUSTOMER_ID
	CALCULATION_ENTITY_ID
	CALCULATION_ENTITY_NAME
	MGT_LEVEL
	WORKING_AREA
	PERSONA
	BIAS_VALUE_PERCEPTION
	BIAS_OVERALL_SATISFACTION
	BIAS_STRATEGIC_FIT
	BIAS_RECOMMENDATION

3. számú melléklet: A DF_CPV_CUSTOMER_AND_REGION
_LEVEL_ACCESS_MANAGEMENT data flow felépítése (Forrás: saját
szerkesztés)

Adatvezérelt folyamatos ügyfél-elégedettségi platform fejlesztése és egységesítése az üzleti
döntéshozatal támogatásához a Nokia Solutions and Networks Kft.-nél

Kun Szabolcs István



4. számú melléklet: A DF_DATA_CPV_CALCULATION

_CONTROL_CPV data flow felépítése (Forrás: saját szerkesztés)

Adatvezérelt folyamatos ügyfél-elégedettségi platform fejlesztése és egységesítése az üzleti döntéshozatal támogatásához a Nokia Solutions and Networks Kft.-nél

Kun Szabolcs István

