IoT integration in Microsoft Dynamics ERP

Peter Porteleki

Multisoft Kft., Bartók Béla street 113., Budapest 1115, Hungary, peter.porteleki@multisoft.hu

Abstract

This article presents a software development project, that is based on integrating between Microsoft Dynamics ERP system, and other systems that are separate. Understanding customer needs, and the workflow of those separate systems, and creating an universal integrated product was essential to solving the specified requirements. This product provides device-independent interoperability between almost any hardware or software, making users life easier, business workflows more effective, eliminating third parties. The article provides a concrete example of this module and it's functions.

Keywords: ERP; IOT; Integration; Microsoft Dynamics; NAV; Navision; Business Central;

IoT integráció a Microsoft Dynamics ERP rendszerben

Porteleki Péter

Multisoft Kft., Bartók Béla út 113., Budapest 1115, Magyarország, peter.porteleki@multisoft.hu

Absztrakt

A cikkben egy vállalatirányítási rendszer a Microsoft Dynamics ERP és a különböző, általában szigetrendszerként működő hardverek és szoftverek közötti integráció kerül bemutatásra. A feladat megoldásához elengedhetetlen volt az ügyfelek igényeinek felmérése, és az egyes vállalatok által használt elszigetelt rendszerek működésének megismerése, valamint egy univerzális termék létrehozása, amely könnyen integrálható a különböző beállítottságú vállalatokhoz. Ezen termék eszközfüggetlen integrációt és átjárhatóságot biztosít a különböző rendszerek között, segítségével automatizmusok hozhatók létre, megkönnyítve ezzel a rendszert használók munkáját, ezáltal pénz, de főleg idő spórolható meg a harmadik fél kiiktatásával. A cikk egy konkrét példát felhasználva mutatja be a modult és annak funkcióit.

Kulcsszavak: ERP; IOT; Integráció; Microsoft; Dynamics; NAV; Navision; Business Central;

1. Bevezető

Néhány ügyfelünk részéről a közelmúltban felmerült az igény különböző egymástól elszigetelt rendszerek integrálhatóságára a Microsoft Dynamics NAV, illetve Business Central rendszerbe (Microsoft, 2020). Az ügyfeleink köre nagyon sokrétű, ezáltal az integrálásra váró rendszerek zöme teljesen eltér egymástól, mégis felmerült egy ötlet, hogy hogyan lehetne egységesíteni ezeket egy közös platform alá. A fejlesztés legfőbb célja ezen platform kialakítása volt, amelyet így minimális módosítással elérhetővé tegyünk a partnereinknek. Ezen modul egyaránt használható egy fős vállalkozások, valamint nagyvállalati környezetben egyaránt, függetlenül a különböző rendszerek földrajzi pozíciójuktól, az egyetlen dolog amire szükség van az az internetelérés. A múltban ügyfélspecifikusan már történtek hasonló irányban kutatások, és

fejlesztések, de ezek nem voltak egyszerűen átültethetők egy másik vállalathoz, így mindenképpen egy univerzális megoldást kellett találnunk, amelyhez könnyen megírható az ügyfélspecifikus kiegészítés. A cikkben ezen ügyfélspecifikus kiegészítés egy kártyás beléptető rendszer formájában kerül bemutatásra. (1. ábra)

1. ábra Kártyás beléptetőrendszer

1.1. A Protokoll kiválasztása

A tervezés során sok ötlet felmerült, hogy milyen protokoll az, amely elég univerzális ahhoz, hogy bármilyen egymástól független rendszerek között megvalósítható legyen a kommunikáció. Végül a saját okosotthon rendszerem tervezése alatt megszerzett tudás alapján az MQTT[1] (Eclipse Mosquitto, 2020) mellett döntöttem. Ezen protokoll nagyon rugalmas, valamint kis erőforrást igényel, nyílt forráskódú, publish-subscribe alapokon működő hálózati protokoll, amely üzeneteket közvetít az eszközök között, és alkalmas olyan környezetbe, ahol a hálózati sávszélesség erősen limitált. Az MQTT protokoll erősen hasonlít a Dynamics ERP rendszerében használható EventPublisher és EventSubscriber metódusokhoz, ahol bizonyos

[1] Message Queuing Telemetry Transport – Üzenetsorbaállító Telemetriai Transzport

eseményeket hozhatunk létre kód szinten, majd ezen eseményekre feliratkozva írhatunk funkciókat. Ez a hasonlóság nagyban megkönnyíti az MQTT integrációt.

1.2. MQTT Kliens a Dynamics-ben

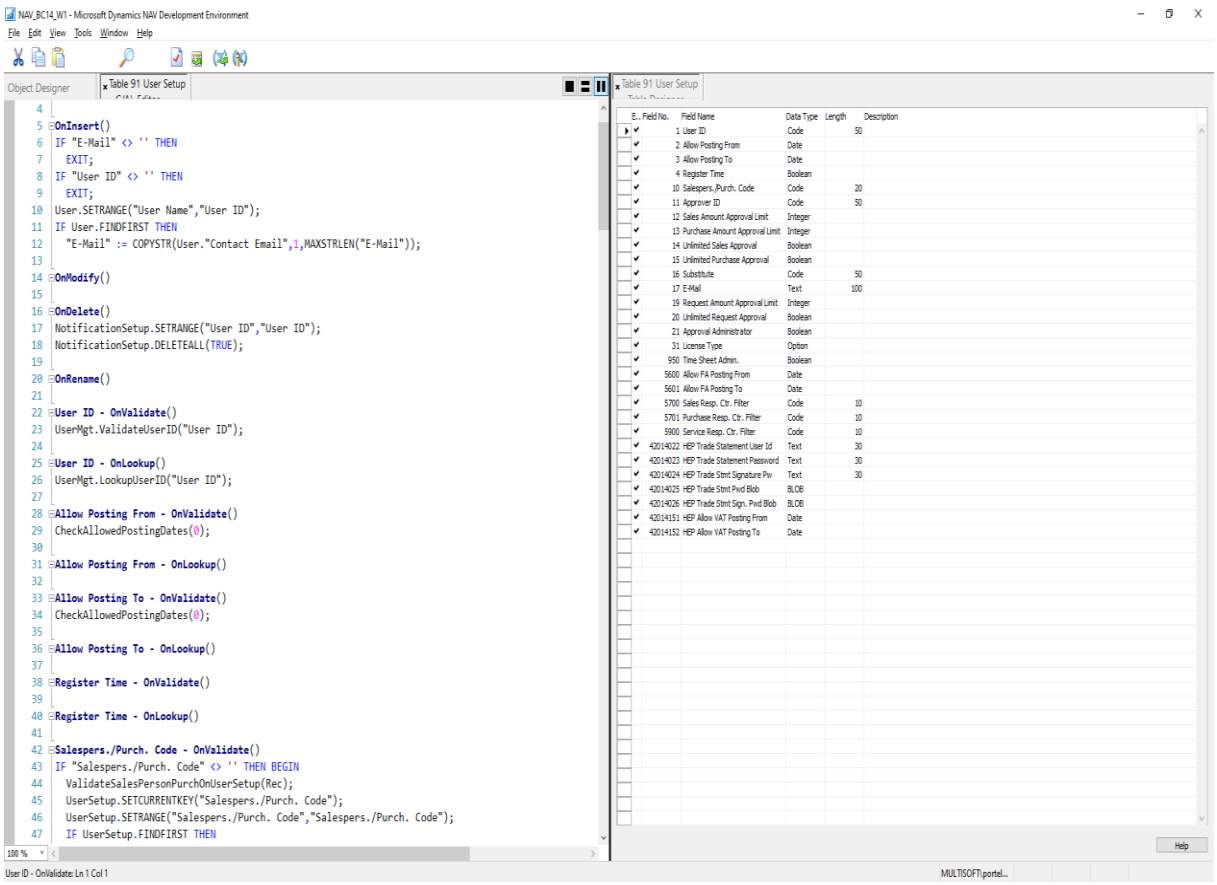
Hogy használni tudjuk az MQTT által nyújtott szolgáltatásokat szükségünk lesz egy kliens szoftverre, amin keresztül a kommunikáció megvalósítható a két rendszer között. A Dynamics NAV, illetve Business Central On-Premise változataiban használható külső .NET alapú könyvtár így a legkézenfekvőbb megoldás az volt, hogy keresni vagy írni kell egy ilyen kell egy ilyen könyvtárat. Ezt meg is találtam az M2MQTT (Paolo Patierno, 2015) személyében, amely akár módosítás nélkül használható a kitalált célra. Az MQTT brókerhez csatlakozva és a megfelelő topikokra feliratkozva, az ERP rendszer képes üzeneteket fogadni és küldeni a bróker irányába.

1.3. MQTT üzenet lista

Az ERP rendszerben, hogy a kliens által lekért és küldött üzeneteket átláthatóan kezelni tudjuk egy táblára van szükségünk, valamint egy Codeunit-ra, amely megadott időközönként lekérdezi az MQTT brókertől a friss üzeneteket és tölti az említett táblát. A Dynamics-ban az üzleti logika nem különül el élesen a különböző objektum típusoktól, így egy tábla, amely SQL szinten valóban egy SQL táblának felel meg, az ERP oldalon programkódot is tartalmazhat, akár a teljes táblára, akár az egyes mezőkre vonatkozóan. (2. ábra) Ezt a lehetőséget kihasználva két Triggert fogunk használni a táblánkban, erre a tervezés részben bővebben kitérünk.

1.4. Standard objektum kiegészítések

A Dynamics fejlesztők, néhány speciális objektum, kivételével hozzáférnek a teljes vállalatirányítási rendszer üzleti logikájának kódjához. Erre szükség is van mert sok esetben az ügyfél igényeinek megfelelően módosításokat kell eszközölni, de jelen esetben csak kiegészítjük a gyári objektumokat, valamint új objektumokat hozunk létre. A beléptető rendszer használatához, néhány új táblára is szükségünk van, amelyben definiáljuk a Kártyákat, a Kártyákhoz hozzárendelt Ajtókat, Egy Ajtók és Beléptetőkapuk tábla, valamint ki kell egészítenünk a Felhasználó táblát, hogy hozzárendelhessük az egyes felhasználókhoz a megfelelő kártyát/kártyákat.



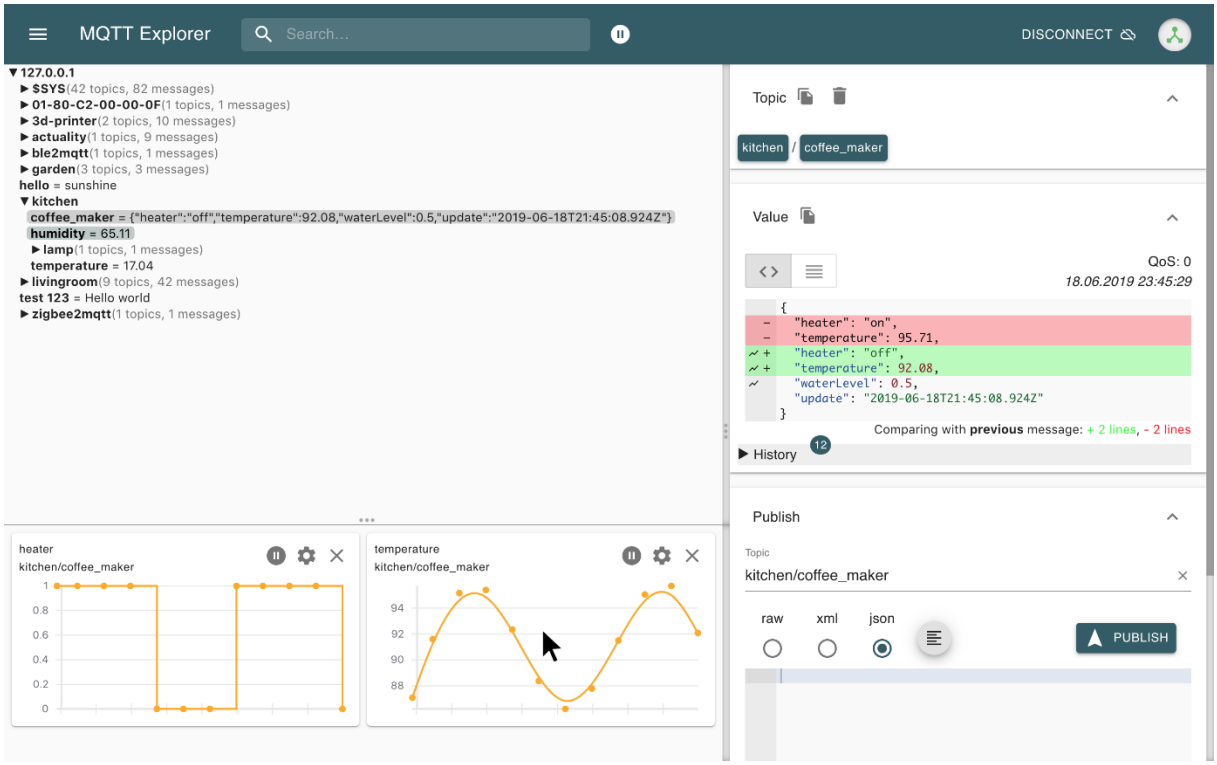
2. ábra User Setup tábla felépítése és üzleti logikája

2. Tervezés és Implementáció

A Dynamics ERP EventPublisher/Subscriber, valamint az MQTT Publish/Subscribe közti logikai hasonlóságok miatt a tervezés nem ütközött technikai bonyodalmakba. Alapvetően két részre lehet elkülöníteni a tervezési és implementációs fázist.

2.1. MQTT oldal

Ezen az oldalon, hacsak nincs speciális igényünk nincs szükség speciális tervezésre. A bróker szoftvert egy tetszőleges környezetbe feltelepítjük, vagy használhatunk a felhőben létrehozott ingyenes vagy fizetős szolgáltatást az igényeinknek megfelelően. Érdemes beszerezni emellé az MQTT Explorer (3. ábra) (Thomas Nordquist, 2020) vagy más, hasonló alkalmazást, amellyel közvetlenül kapcsolódhatunk a brókerhez, lekérdezhetjük az összes üzenetet, nyomon követhetjük az egyes topikokban a változásokat, feliratkozhatunk topikokra, valamint üzeneteket tehetünk közzé az egyes topikokban, megkönnyítve ezzel a tesztelést, valamint a rendszer monitorozását.



3. ábra MQTT Explorer

A hardvereket, jelen esetben a kártyás beléptető rendszert, valamint az ajtónyitó mechanikát csatlakoztatjuk az MQTT brókerhez a megfelelő topikba.

Például: /dooraccess/readers/reader1, valamint /dooraccess/doors/door1

Az első topikba beérkező üzenetet feldolgozzuk, majd amennyiben szükséges a megfelelő üzenetet a második topikba, amely az ajtó vezérlése elküldjük. Maguk az eszközök is tehetnek közzé üzenetet a saját topikjukba, így megoldható a nyitás után az automatikus zárás, ez lehetővé teszi egy adott ajtó aktuális állapotának lekérdezését is.

Az ajtónyitó hardver a megfelelő topikban egyetlen paraméterrel rendelkezik, amely az „Open” névre hallgat és két állapotot vehet fel, igaz vagy hamis annak megfelelően, hogy az ajtó nyitható vagy zárva van, nyitás után a beállított időtartamnak megfelelően automatikusan visszaállítja a zárt állapotot elkerülve az illetéktelenek belépését.

2.2. Dynamics oldal

Az alaprendszer használatához szükségünk van az MQTT Kliens könyvtárat példányosító Codeunitra, valamint ebben a könyvtárban iratkozunk fel az egyes topikokra, ezen felül szükség

van egy táblára a lekérdezett üzenetek tárolására, amellyel gyakorlatilag SQL oldalon reprodukáljuk az MQTT bróker aktuális, és múltbéli állapotait. Az MQTT kliens codeunitot szűk időközönként meghívjuk, hogy valós időben követhető legyen az üzenetek folyama. Szükség van még a Quality of Service[2] (QoS) definiálására. Ez a protokoll szempontjából szükséges, hogy a brókerbe beküldött üzenetünk a feliratkozók részére milyen fontossággal bírnak.

QoS-ből az MQTT három szintet támogat:

1. Legfeljebb egyszer (0)
2. Legalább egyszer (1)
3. Pontosan egyszer (2)

Ezen szintek használata a felhasználás jellegéből adódóik, és valóban azt fogalmazza meg hogy mennyire fontos az egyes topikokba beküldött üzenet nyugtázásának fontossága.

Például: ha hőmérséklet monitorozásra használjuk, ahol 30 másodpercenként beküldjük a megfelelő szenzorhoz tartozó topikba az aktuális hőmérsékletet, de erre az eseményre feliratkozott szoftverhez (amely például egy hőmérsékletet mutató kijelzőben működik) nem jut el, nem jelzünk hibát, és próbáljuk meg mindenképpen eljuttatni az üzenetet, hiszen 30 másodperc múlva jön a következő, friss értéket tartalmazó üzenet, így tanácsos a „0” szint használata. Azonban, ha olyan kritikus rendszerről beszélünk, ahol feltétlenül célba kell juttatni az üzenetet célszerű magasabb szintet használni, így a rendszer, noha rendelkezik frissebb bejegyzéssel az előző üzeneteket is elküldi a végeszközök számára.

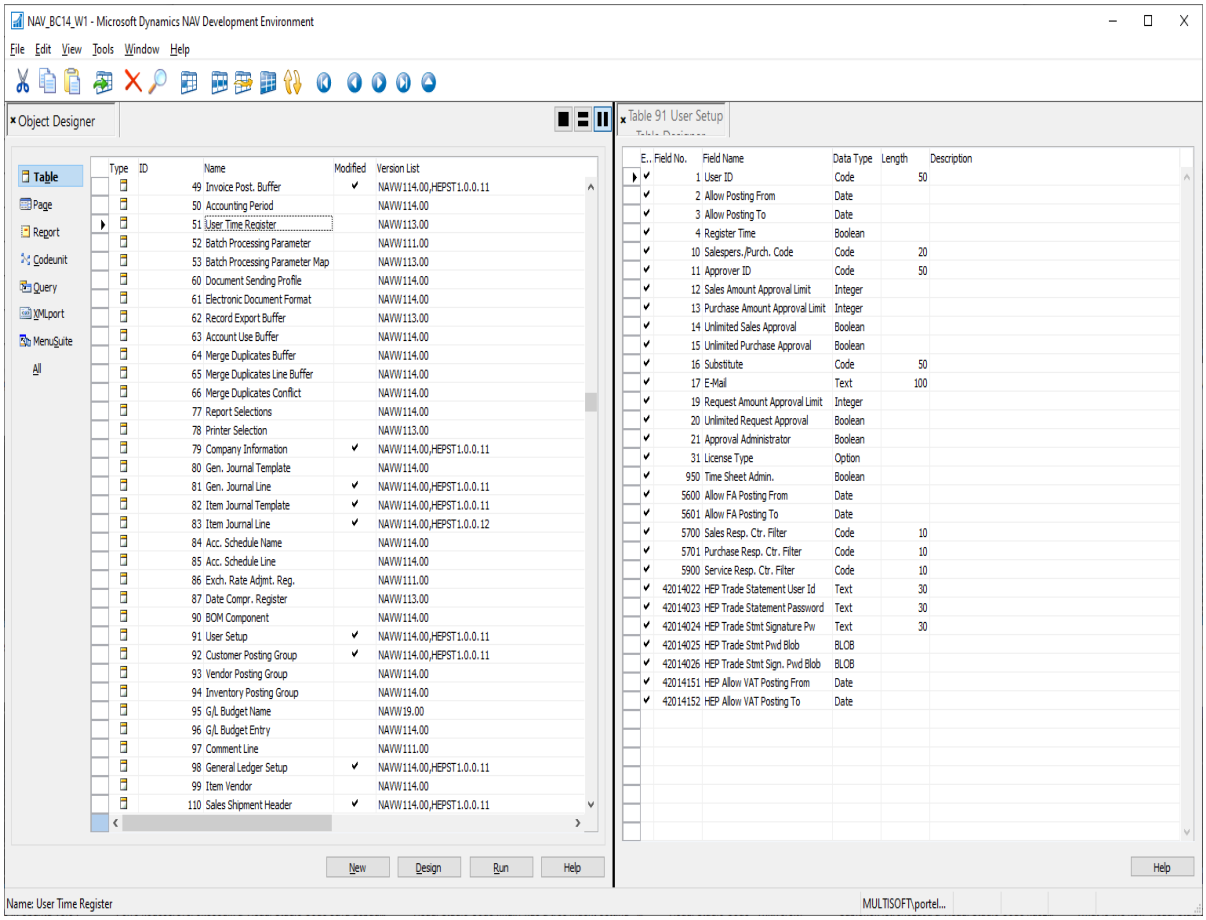
Jelen esetben a 0 szint használata megfelel az igényeinknek, mert egy kártyás beléptető rendszer esetén ha az üzenet valamilyen okból nem jut el a Dynamics oldalra, oda ahol a beléptetés authentikáció lezajlik, a felhasználó még egyszer meg fogja próbálni a belépést, valamint nem szükséges a régebbi üzeneteket eljuttatni az ERP felé, hiszen ez egy hálózati kimaradás esetén akár biztonsági kockázatot is jelenthet, az ajtók az ERP rendszer legközelebbi bekapcsolásakor kinyitja az ajtókat, noha az üzenet továbbítása és fogadása között akár napok telhetnek el, ezért is fontos jelen esetben hogy csak a legutolsó állapot kerüljön bejegyzésre.

[2] Jelen esetben a szolgáltatáson közzétett üzenetek prioritását jelöli

2.2.1. Táblák

Ahogy fentebb is említés esett róla, szükségünk van új táblák létrehozására a rendszerben. Az alaprendszer működéséhez egyetlen tábla is elegendő, amelyben az üzeneteket tároljuk. Ehhez a táblához két EventPublisher funkció hozzáadása is szükséges, amelyek a táblában végzett műveletnek megfelelően egy eseményt hoznak létre amire más objektumokból fel lehet iratkozni. A két Publisher az OnAfterModify, amely egy rekord módosítása után kerül meghívásra, illetve az OnAfterInsert, amely egy új rekord bekerülése után kerül meghívásra.

A tábla felépítése igen egyszerű, egy elsődleges kulcs, valamint egy üzenet fielddel rendelkezik, ennyi már elegendő az alapvető működéshez. (4. ábra)



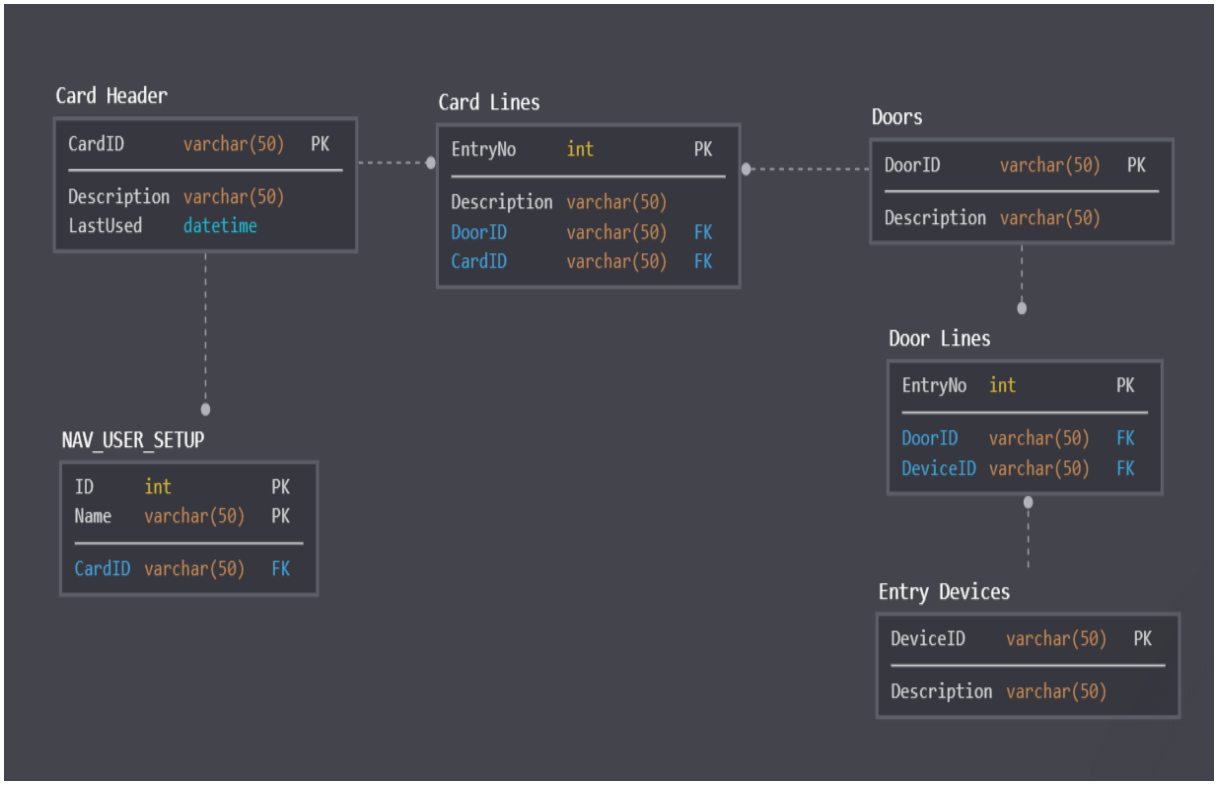
4. ábra Business Central C/AL Development Environment

A beléptetőrendszerhez szükségünk van még további táblákra, amelyek a következők:

1. Card Header: Az egyes kártyák alapvető adatait tartalmazza.
2. Card Lines: Az egyes kártyák által nyitható beléptető eszközöket tartalmazza

3. Doors: Az ajtókat tartalmazza
4. Door Lines: Az ajtókhoz tartozó beléptető eszközöket tartalmazza
5. Entry Devices: A beléptető eszközöket tartalmazza

Az adatbázis kapcsolatokat (5. ábra) általában papíron szoktam megtervezni és megrajzolni, később, amennyiben a projekt megkívánja a dokumentációba automatikus eszközökkel generálunk vizuális ábrákat.



5. ábra Adatbázis kapcsolatok

2.2.2. Pagek

A fent említett táblákhoz néhány oldal létrehozása is szükséges, hogy a felhasználó egyszerűen végezhessen módosításokat, egy-egy page az alapvető CRUD műveleteken kívül tartalmazhat más funkciókat is, de elsősorban az adatbázis kezelésére használatos, jelen esetben sincs szükség többre. Egy-egy oldal automatikusan kezeli az egyes táblák közti kapcsolatokat, leegyszerűsítve az adatok kezelését.

2.2.3. Code Unitok

Az MQTT kliens codeuniton kívül szükségünk van még a kívánt funkciónak megfelelő kód megírására.

Az ajtónyitáshoz szükséges codeunit működése nagy vonalakban:

A codeunitban feliratkozunk a táblában létrehozott OnAfterInsert eseményre (jelen esetben csak ezt használjuk mert minden üzenetet egyesével tárolunk, függetlenül attól, hogy melyik topikba érkezett). Így minden MQTT Üzenetek táblába történő íráskor lefut ebben a feliratkozóban megírt kód.

A codeunitban a táblán előzetesen elvégzünk egy szűrést, hogy kizárólag a /dooraccess/readers/ alá tartozó üzeneteket lássuk így bármely eszközön történő belépőkártya használatot látni fogunk.

Amennyiben a topikon belül az eszköz szerepel az adatbázisban pl. reader1 összehasonlítást végzünk, hogy a kártyák táblában található kártya aktivált-e, valamint van-e jogosultsága az adott ajtót kinyitni, amennyiben nem, úgy a felhasználót nem engedjük be, szükség esetén rögzíthetjük egy plusz táblába, hogy melyik ajtót mikor ki nyitotta ki/vagy épp nem nyitotta ki a kártyájával.

Amennyiben a kártya szerepel az adott ajtót nyitó kártyák között, az MQTT könyvtáron keresztül kiküldünk egy üzenetet az ajtónak megfelelő (/dooraccess/doors/ajtóneve) topikba, amelyben beállítjuk az állapotát nyitottra, így a felhasználó be tud lépni az ajtó által védett helyiségbe.

3. Tesztelés

A tesztelés az egyik legfontosabb eleme a szoftverfejlesztésnek, a legfőbb célja a hibák csökkentése, illetve megtalálása. A tesztelés tervezésekor úgy kell megválasztani a tesztelés módját, hogy a lehető legtöbb hibát kiszűrhessük, illetve kijavíthassuk. A kód írása közben a fejlesztő folyamatosan tesztel, eközben nálunk rendszerint egy független fejlesztő teszt codeunitokat ír, amelyek az automatizált tesztelést segítik. Ezen tesztek rendszerint pozitív, illetve negatív teszteket is tartalmaznak, jelen esetben az MQTT protokollon hardverek nélkül is követhetőek az üzenetek, szimulálhatóak a hardverek működései.

Hiba esetén nem kell végignéznünk a teljes forráskódot, mert az ERP rendszer, valamint a Visual Studio (Microsoft, 2020) is rendelkezik Debuggerrel[3], amely megállítja a program futását a töréspontokon, illetve a hibát kiváltó eseményről is információkkal szolgál.

Egy hiba felismerése és megtalálása esetén újbóli tesztekre van szükség, hogy meggyőződjünk a helyes működésről.

Mi általában csak a test codeunitokat szoktuk használni tesztelés céljából, ezeket úgy írjuk meg hogy a lehető legnagyobb részét lefedje a kódnak, valamint ügyfeleink általában rendelkeznek teszt rendszerrel, ahol az elkészült fejlesztéseket az ügyfeleink kipróbálhatják, az esetleg előforduló hibákat jelenthetik mielőtt az éles rendszerbe bekerülne.

4. Összegzés

A projekt alapvetően a saját hobbim kiterjesztése a munkámhoz céllal jött létre(ezt át kell fogalmazni kicsit furán hangzik: A projekt alapvetően azzal a céllal jött létre, hogy a saját hobbimat kiterjeszthessem a munkámra, viszont néhány ügyfél érdeklődése miatt folyamatos fejlesztés alatt áll, hogy később egy univerzálisan elérhető termék jöhessen létre. A legnehezebb ezen univerzális funkciók megtalálása, amely bárki számára egységes előnyöket nyújt, valamint a könnyű bővíthetőség megalkotása volt.

Természetesen előfordulhat, hogy az alaprendszert részben át kell tervezni egy egyéni igény miatt, de ez a rendszerbe integrálható hardverek, illetve szoftverek sokszínűsége miatt néha elkerülhetetlen, a platform által nyújtott lehetőségek tárháza gyakorlatilag végtelen.

Az egyes cégeknél dolgozóknak nagy segítség, hogy egyetlen rendszer használatát szükséges elsajátítaniuk a munkavégzésükhöz, és a munkakörüknek megfelelően azokhoz a modulokhoz férnek hozzá, amikhez jogosultságuk van. A külső rendszerek mély integrációjával egységes az egységes kezelőfelület mellett rugalmasan fejleszthető rendszert is kapunk, ami az ügyfél igényeinek megfelelően átalakítható, és személyre szabható.

Felhasznált szoftverek

Microsoft (2020). Microsoft Dynamics Business Central 14.0 [Online] https://dynamics.microsoft.com/en-us/business-central/overview/ (2020.05.10)

[3] Forráskód elemző és hibakereső szoftver

Microsoft (2020). Microsoft Visual Studio Code [Online] https://code.visualstudio.com/ (2020.05.10)

Eclipse Mosquitto (2020). MQTT Broker 1.6.9 [Online] https://mosquitto.org/ (2020.05.10)

Thomas Nordquist (2020). MQTT Explorer 0.4.0-beta [Online] http://mqtt-explorer.com/ (2020.05.10)

Paolo Patierno (2015). M2MQTT Client 4.3.0 [Online] https://www.nuget.org/packages/M2Mqtt/ (2020.05.10)

Rövid szakmai életrajz

Porteleki Péter szoftverfejlesztő munkakörben dolgozik a Multisoft Számítástechnikai Kft.-nél. Felsőfokú tanulmányait a Dunaújvárosi Egyetem, mérnökinformatikus BSc képzésén, szoftverfejlesztés specializáción szerezte meg. Érdeklődési területe a programozás, gitározás.