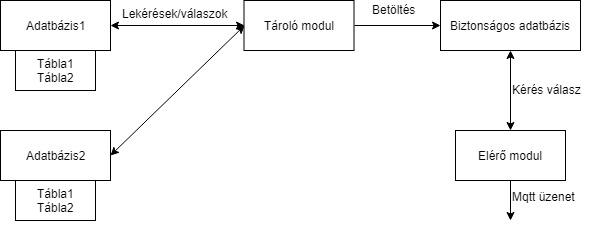
Gateway alkalmazás tároló és elérő modul rendszerterv

A teljes gateway alkalmazás azt a feladatot látja el, amivel különböző típusú adatforrások csoportok azonos módszerrel tárolásra kerüljenek, majd szükséges validálás után egy központi adatbázisba tárolásra kerüljenek. Az alkalmazás bármilyen típusú architektúrán képes működni, ami megteremti a szükséges környezetet, amin a moduljai futtathatók, implementálva van rajta a kommunikációs protokollhoz való környezet, illetve egy adatbázis. A választott hardware egy Raspberry Pi, mivel a felsorolt környezetet támogatja, mind a JVM,mind pedig a mySQL könnyen implementálható, illetve MQTT brókerként is üzemeltethető. Továbbá mérete miatt könnyen hordozható.



A gateway tetszőleges számú modulból állhat, hiszen alap megközelítés szerint minden modul független, a modulok csak a vezérléssel kommunikálnak, ezért minden modulnak szüksége van a vezérlés számára üzeneteket küldeni, illetve a vezérléstől üzeneteket fogadni. Ez a kommunikáció jelenti mind a vezérlés által küldött konfigurációs paraméterek fogadását, illetve MQTT üzenetek fogadását, illetve küldését. Alap megközelítésben a gateway egy biztonságos adatbázisból indul. Ezt követi az elérő modul, ami a biztonságos adatbázisból való olvasásért, illetve az adatok megfelelő formátumú továbbításáért felel. Ezt követi a validáló modul, ami a megkapott paraméterek alapján eldönti, hogy a mérések helyesek,hibásak,duplikáltak voltak-e. Majd az utolsó küldő modul elküldi a központi adatbázis számára a validált recordokat.

Biztonságos adatbázis: A biztonságos adatbázis fő célja, hogy az adatok egy központosított adatbázisban kerüljenek tárolásra feldolgozás előtt. Jelen megközelítésben a biztonságos adatbázis egy mySQL alapú adatbázis táblája. Ez a tábla lehet lokálisan a gatewayen, vagy pedig egy központi adatbázisban.

Ha ez a biztonságos adatbázis nem létezik, tehát az adatok tárolása nincs központilag megoldva, akkor egy tároló modullal kiegészül az egész struktúra. Ezt a modult nem a vezérlés indítja, illetve paraméterezi fel, hanem a vezérléssel együtt indul. A biztonságos adatbázis létrehozásáért nem a tároló modul felel, a modul úgymond csak egy csőként funkcionál.

Mivel a vezérlés központilag üzemel, így előre meghatározott formában történik a kommunikáció, az üzenet formája kötött a hatékony futás végett. E modul feladata, hogy modulok által létrehozott fájlok menedzselése, áthelyezése, törlése,létrehozása,üzenetek fogadása, modulok indítása,újraindítása, továbbá a modul feladata a fájlok tárolásához szükséges mappaszerkezetek létrehozása.A teljes gateway struktúra felett egy felügyeleti modul fogja ellenőrizni a programok működését, helyes működését, továbbá felelős lesz riasztások küldéséért.

Egy központi fájl felel a teljes gateway futásához szükséges paraméterek tárolásáért. Ebben a fájlban központilag módosíthatók a modulok futásához szükséges változók. Ezzel a fájllal csak a vezérlés érintkezik, az itt megadott paraméterek alapján indítja el a szükséges modulokat.



Mivel a vezérlésnek néhány információt tudnia kell a modulokról, ezért ezek egységesen a saját paraméterek előtt szerepelnek. E paraméterek között megjelenik az adott modul indítási parancsa, a vezérlés ez alapján indítja el a modulokat a szükséges paraméterek kiegészítésével. Mivel minden modul a vezérléssel kommunikál, így az adott modul azon topikja elől szerepel, amire az adott modul üzeneteket küld MQTT segítségével, továbbá a vezérlés számára itt lehet definiálni, hogy miután a modul elvégezte a feladatát, a már módosított fájlt új helye hol legyen a mappa rendszerben. A modulok a saját mappájukban lévő fájlokat módosítják, ezen fájlok mozgatásáért a vezérlés felelős. A modul saját paramétereit a parameters alatt adhatók meg, ezeket a vezérlés induláskor kiosztja a modulok számára, elválasztás alapján.

Mivel a moduloknak bármely architektúrán működni kell, így kézenfekvő, hogy valamiféle projektbuildert használjunk. A modulokban a buildelést az Apache Maven framework-el oldjuk meg. A Maven egy olyan szoftver amelyet szoftverprojektek menedzselésére és a build folyamat automatizálására lehet használni. A Maven bevezeti a **P**rogram **O**bject**m**odel-t, ami egy létrehozandó projektet buildelését és ezek függőségeit írja le. Ezen tulajdonságokat egy XML fájlban lehet definiálni,amelynek van egy dependencies, illetve build része.A függőségek részben bármely könyvtár (amit a Maven támogat) megadható megfelelő formátumban, e könyvtárak a program futásakor a Maven importálja. A build részben lehet definiálni, hogy a projekt milyen compilert használjon, milyen környezeten fusson. Mivel a modulok külső könyvtárakból használnak előre definiált metódusokat, különböző típusú objektumokat, ezért a modulok a következő függőségeket használják.

Kotlin: JetBrains által fejlesztett programozási nyelv, lefordítható JavaVirtualMachine-re, illetve JavaScriptre. A Kotlin együttműködik a Java-ban írt kóddal és épít a java programkönyvtár részeire.

Maven dependency: <dependency>

<groupId>org.jetbrains.kotlin</groupId>

<artifactId>kotlin-stdlib-jdk8</artifactId>

<version>1.2.70</version>

</dependency>

Gson(Google Json): A Gson egy nyílt forráskódú Java könyvtár, amivel Java objektumok könnyen szerializálhatóak json formátumra, illetve json formátumból könnyen alakíthatók Java objektumokká. Mivel a Kotlin képes az összes java objektummal dolgozik jelen esetben is használható.

Meghatározott json formátum:{“content”:”{“attribútum név”:”mérés”}”}

Maven dependency: <dependency>

<groupId>com.google.code.gson</groupId>

<artifactId>gson</artifactId>

<version>2.8.5</version>

</dependency>

JDBC(Java DataBase Connectivity):A Java Database Connectivity, röviden JDBC egy API a Java programozási nyelvhez, amely az adatbázis hozzáférést támogatja. A JDBC definiálja az adatbázisok lekérdezéséhez és módosításához szükséges osztályokat és metódusokat.

Maven dependency: <dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.12</version>

</dependency>

Eclipse Paho: Az Eclipse Paho könyvtár implementálja az MQTT kommunikáló protokoll használatához szükséges metódusokat, brókerhez való csatlakozást, üzenet küldést,fogadást, topikra való feliratkozást.

Maven dependency: <dependency>

<groupId>org.eclipse.paho</groupId>

<artifactId>

org.eclipse.paho.client.mqttv3

</artifactId>

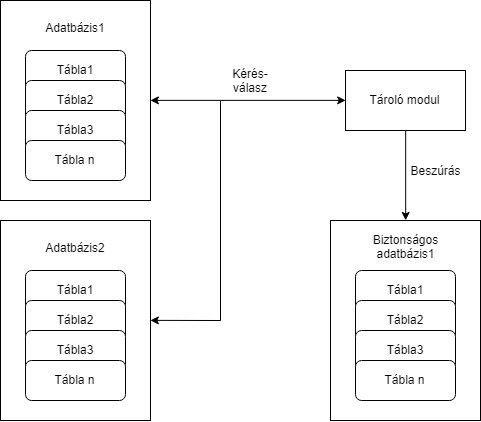
<version>1.2.0</version>

</dependency>

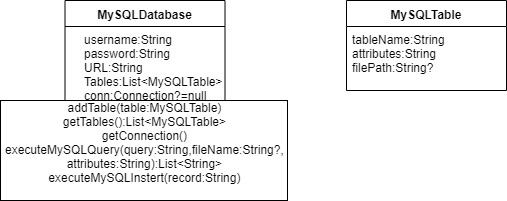
Mindkét modulban az adatok nagy részének tárolása listákkal, illetve map-ekkel történik, ehhez a Kotlin megvalósított Collection könyvtárat használjuk, azonban ha ez nem elérhető, keresni kell egy hasonló típusú könyvtárat melyben a listák, illetve hasonló típusú adatszerkezetek megvannak valósítva.

Tároló modul

A modul fő feladata az azonos típusú adatforrás csoportok egy közös típusú adatbázisba való betöltése, ezzel a betöltéssel a mérések azonos néven lennének eltárolva, így úgymond egy megfeleltető funkciót is ellát.



Mivel a modul adatbázisok között üzemel, így egy olyan osztály létrehozása kézenfekvő, amelyben az adatbázis tulajdonságok, illetve metódusok definiálásához. Ez az osztály a felhasználó által definiált, azonban nagy hangsúlyt fektet a JDBC könyvtárban definiált metódusokra, illetve változókra. Mivel SQL alapú adatbázisokra lesz felkészülve a modul, így ezen osztály mellett még definiálni kell egy osztályt, ami az adatbázisokban lévő táblák tulajdonságainak tárolására szolgál. Az adatbázisok a jelenlegi gateway verzióban MySQL mind az adatforrásoknál, mint pedig a biztonságos adatbázisnál.



Az UML diagram által definiált MySQLDatabase osztályban megvalósítjuk a szükséges metódusokat, mellyel az adatbázis csatlakozás, rekord lekérés, illetve rekord beillesztés megvalósíthatók.

Adatbázis csatlakozás: A metódus megvalósítja az objektum létrehozásakor megadott adatok alapján az adatbázis csatlakozást. A metóduson belül az adatbázis url-en kívül, még szükségesek különböző hozzáférési, illetve a kapcsolat tulajdonságára vonatkozó adatok. Ezen adatok közül az alapok a felhasználónév, illetve a jelszó. Ezek az adatok map-ben kerülnek tárolásra, ezt a map-et a JDBC-ben definiált Connection típusú változó getConnection() metódusához hozzáadható. A metódus meghívásakor történik meg az adatbázis kapcsolat kiépítése.

Lekérdezés futtatás: A metódus létező adatbázis kapcsolaton futtatható. Az átadott MySQL utasítást, futtatja az adatbázisban, majd az visszakapott kurzort egy eredmény listává alakítja, ezzel a az eredmény listával tér vissza. A kurzoron való végigiteráláshoz a JDBC-ben használt metódussal iterálunk végig.A kurzoron való iterálás közben az értékek kinyerése az attribútum lista alapján történik. Az eredmény listában egy rekord jelent egy elemet. A visszakapott értékeket egy “,” karakterrel választja el, tehát a rekord egy kezdeti átalakításon megy keresztül, egy rekordból, egy String lesz. Ez a String már arra a formára lesz hozva, amit egy MySQL utasításban már csak bele kell illeszteni. Miután a listához adás megtörtént, a táblához létrejött fájlban a lekért rekord AutoIncrement-es ID oszlopának tárolásra kerül. A modul ezzel a megoldással oldja meg egy leállás utáni visszaállást.

Beszúrás futtatás:A metódus a megkapott SQL insert utasítást futtatja a adott adatbázis objektumon. Ehhez a JDBC-ben definiált update műveletet használja.

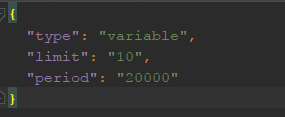
Mivel egy adatbázison belül több táblából történhet az adatelérés, így egy saját mySQLTable osztály létrehozása szükséges, amelyben a tábla tulajdonságait tárolom. Ezen tulajdonságok mellett tárolom a táblához tartozó fájlnak az elérési útját, amely fájl célja, hogy az utolsó lekért ID tárolásra kerüljön.Ez a változó lehet null, hiszen azon tábláknak nincs szüksége erre a fájlra amiből nem történik meg lekérés.

Program futása:

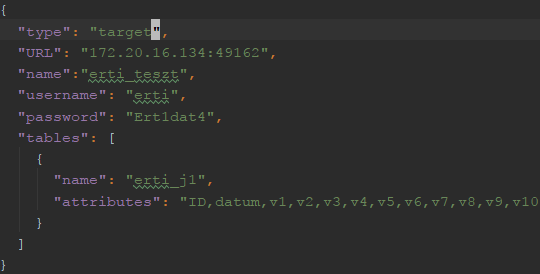
* változó deklaráció
* konfigurációs paraméterek értelmezése
* adatbázis csatlakozás
* lekérés és betöltés

Kezdeti listák, változók deklarálása: A külső adatforrás csoportokból lekért rekordok számának, illetve a lekérések gyakoriságának tárolása változókban, ezek az értékek a konfigurációs fájl értelmezése után felülíródnak. Két üres lista létrehozása , az egyik listában a külső adatforrás csoportok fognak tárolásra kerülni, a másik listában pedig a biztonságos adatbázisok kerülnek tárolásra. Mivel az adatbázis típusú objektumok, belső változók alapján megkülönböztethetők, így elégendő a listák megkülönböztetése.

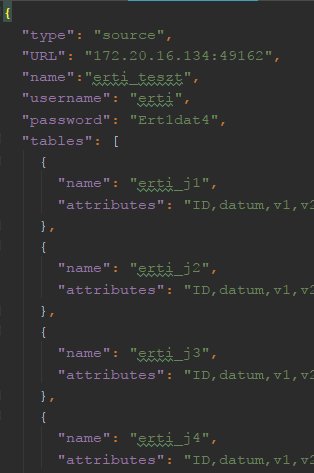
Konfigurációs paraméterek értelmezése: A modul indításakor definiált értékek felülírása, a listák feltöltése. A konfigurációs fájlból kapott json típus mezője alapján történik a megkülönböztetés. E mező alapján kerülnek hozzáadásra a megfelelő listához, vagy íródnak felül a megfelelő változók.



Ebben a paraméterben a futáshoz szükséges változók kerülnek átadásra.



A target típusú paraméterben a cél adatbázishoz való csatlakozás,illetve beszúráshoz szükséges adatok jelennek meg.



Az adatforrás csoportok jelenleg mySQL táblák, ezért paraméterei hasonlítanak a cél adatbázis paramétereihez.

Mivel az adatbázisokkal való kapcsolat kiépítéséhez azonos paraméterek szükségesek, így a json formátuma gyakorlatilag megegyezik.A modulnak tetszőleges számú adatbázis átadható paraméterként.Ha az adatbázishoz kéréseket küld a program, akkor létrejön egy fájl az adott tábla nevével, amiben az utolsó lekért és sikeresen letöltött rekord id-je kerül tárolásra, alapértelmezetten a fájl létrehozáskor egy 0 kerül bele, ha ez a fájl létezik, akkor innen folytatódnak tovább a lekérések.

Az args tömbben megkapott paraméterek feldolgozásához valamilyen könyvtárat célszerű alkalmazni, hiszen enélkül a paraméterek értelmezése nehézkes lenne, a kapott paramétert String-ként kellene kezelni, pedig előre megformázott formátumban van már.

A Gson könyvtár a megkapott paraméterekből JsonObjektumokat készít, ami a könyvtár egy előre megírt osztályának példányosítása. Ennek az objektumnak az előnyei, hogy előre megírt metódusokkal lekérhetők paraméterei, akár tömbként is, amit az adatbázis paraméterek átadására szolgáló json használ is.

Adatbázis csatlakozás:A listákon való végigiterálás közben mindegyik adatbázishoz a megfelelő metódussal kiépül a kapcsolat. Ezek megvalósítására a mySQLDatabase osztályban definiált getConnection metódust használja a program. Mivel az adatbázisok a kapcsolódáshoz szükséges információt változóikban tárolják, így elegendő az őket tároló listákon végig iterálni.

timer=Timer(period)

timer.scheduleAtFixedRate{

for(sourceDatabase in sourceDBs){

for(sTable in sourceDatabase.getTables()){

results: EmptyList<String>=sourceDatabase.executeMySQLQuery(param)

for(record in results){

for(targetDatabase in targetDBs){

for(tTable in targetDatabase.getTables(){

targetDatabase.executeMySQLInsert(param)

}

}

}

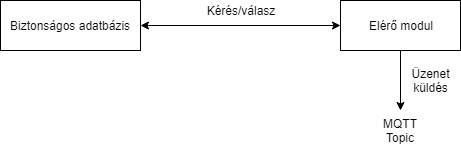
}

}

}

Lekérés, adatok átalakítása, kiegészítése, betöltése: Mivel a paraméterekben átadott összes adatforrásból az adatok betöltése, mind hisztorikusan, mind pedig valós időben szükséges, így az adatok lekérése,átalakítása,betöltése foreach ciklusok egymásba ágyazásával történik. A program azt a logikát követi, hogy az összes adatforrásból az adatok bekerülésre kerüljenek a biztonságos adatbázisokba, a bekerülés sorrendje tetszőleges, hiszen a biztonságos tárolás lényege nem az adatok feldolgozása, csak tárolása, mielőtt a gateway alkalmazáson áthaladnak. A biztonságos adatbázisban egy külön attribútum jelöli az adatok eredetét, ebbe az attribútumban a tábla neve szerepel, melyből a lekérés történt. A program a rekordokat egyesével szúrja be az adatbázisba, a saját osztályban definiált metódussal. Mivel a programnak ez a része egy külön szálon fut, amit egy Scheduler időzít, így ez megoldást nyújt mint a valós idejű, mint pedig a hisztorikus működésre, hiszen a program adott időközönként kéréseket intéz az adatbázishoz, így ha ezen kérések közötti időtartam kisebb, mint a beérkezendő adatok közötti idő, akkor az adatbázisba az összes mért adat be fog kerülni

## Elérő modul

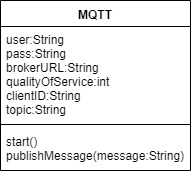


Ez a modul felel az adatok adatbázisból való kiolvasásáért és a kezdeti alakításáért. Az adatelérő modul indításakor a vezérlő modulnak létre kell hoznia egy konfig fájlt, amiben meg tudja adni az adatbázis csatlakozáshoz szükséges adatokat, továbbá annak az MQTT Topic-nak a nevét, ahová a modul az átalakított adatot elküldi.

Az adatbázis csatlakozáshoz, illetve MQTT brokerre való csatlakozáshoz szükséges adatokat a vezérlő modul által létrehozott konfigurációs fájlból kapja meg. A lekért rekordok számát, illetve a lekérések gyakoriságát is a konfigurációs fájlból lehet definiálni. A modul az utolsó lekért rekord id-je alapján kéri le a következő rekordokat, így a fájl alapján a modul visszaállítható. Minden adatbázis minden táblája külön fájlban tárolja az utolsó id-t. A továbbított JSON formátumú üzenetben a lekért értékek mellett megjelennek az attribútumok nevei. (pl.: „tempAt2Meter”:”15”,”humidity”:”40”)

A modul csak a vezérléssel kommunikál, azonban ez a kommunikáció egyoldalú  
a felparaméterezés után, nem fogad további üzeneteket.

A modul a tároló modulban definiált mySQL osztályt használja, az osztályban definiált metódus készletből az összes metódust nem használja.



Mivel az osztály implementálja az MQTTCallback nevű osztályt, így az absztrakt metódusok felüldefiniálása kerülnek, ezen metódusokat a program nem használja, tehát nem szükséges programkódot mögé definiálnunk. Később a modul továbbfejlesztésére ezek a metódusok használatba fognak lépni. A modulnak ebben az osztályban két fő metódusa van, a kapcsolat kiépítés a brókerhez, illetve egy üzenet publisholása, a kapott topikra.

Kapcsolódás a brókerhez: Az adott objektum a megadott url alapján csatlakozik a központi brókerhez, a megadott felhasználónév, illetve jelszó segítségével. Ez a metódus egy szálat indít, innentől a kapcsolat a program futása során fennáll a bróker illetve a kliens között.

Posztolás egy topikra: A paraméterben átadott üzenetet az objektum posztolni fogja, a konfigurációs fájlból kapott topicra, ezt a topikot a vezérlés hallgatája, innen ő foglalkozik az üzenettel. A program könyvtárban előre megírt quality of service-t használja, amivel azt érjük el, hogy az adott üzenet pontosan egyszer kerüljön elküldésre, hiszen ha többször küldenénk, akkor a vezérlésnél duplikátumok keletkeznének. A protokollban az üzenetek karakter tömbként kerülnek továbbításra, így a kapott paramétert át kell kasztolni egy karakter tömbbé.

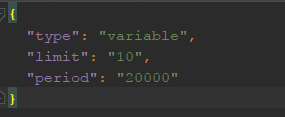
Program futása:

* változók deklarálása
* konfigurációs paraméterek értelmezése
* adatbázis, mqtt csatlakozás
* lekérés és posztolás

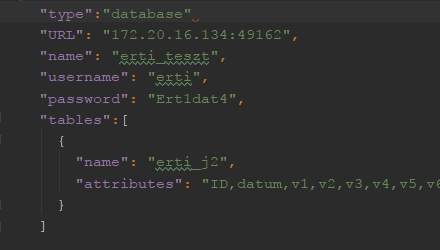
Kezdeti listák, változók deklarálása: A külső adatforrás csoportokból lekért rekordok számának, illetve a lekérések gyakoriságának tárolása változókban. Ezek az értékek a konfigurációs fájl értelmezése után felülíródnak. Egy üres lista létrehozása MySQL objektumok tárolására.



Ebben a paraméterben kerül átadásra a topic neve, amire a modul az üzeneteket küldi.



Ebben a paraméterben a futáshoz szükséges változók kerülnek átadásra.



Az adatbázis típusú paraméterben a cél adatbázishoz való csatlakozás,illetve lekéréshez szükséges adatok jelennek meg.

Konfigurációs paraméterek értelmezése:

A modul indításakor definiált értékek felülírása, a listák feltöltése. A konfigurációs fájlból kapott JSON típusú mezője alapján történik a megkülönböztetés. E mező alapján kerülnek hozzáadásra a megfelelő listához, vagy íródnak felül a megfelelő változók.

Adatbázis, MQTT csatlakozás:

A listákon való végig iterálás közben mindegyik adatbázishoz a megfelelő metódussal kiépül a kapcsolat. Ezek megvalósítására a mySQLDatabase osztályban definiált getConnection metódust használja a program. Mivel az adatbázisok a kapcsolódáshoz szükséges információt változóikban tárolják, így elegendő az őket tároló listákon végig iterálni.

Statikus adatokból történik meg a csatlakozás az MQTT brokerhez, mivel az adott gatewayen egy MQTT broker fog futni, aminek az adatai statikusan vannak megadva.

timer.scheduleAtFixedRate(period){

for(database in databases){

for(table in database.getTables()){

results=sourceDatabase.executeMySQLQuery(param)

for(record in results){

mqttClient.publish(recordToJson(record))

}

}

}

}

Lekérés és üzenetküldés:

Mivel a paraméterekben átadott összes adatforrásból az adatok betöltése, mind hisztorikusan, mind pedig valós időben (timer segítségével) szükséges, így az adatok lekérése betöltése foreach ciklusok egymásba ágyazásával történik.

Az adatbázis tábláin egy lekérdezést hajtok végre, majd a megkapott eredmények listán végigiterálok, a rekordot betöröm a kívánt JSON formátumba és ezt publisholom a megkapott MQTT topic-okra.