Összefoglalás

A mai világban az elektromos áram mindenhol megjelenik. Az autóiparban egyre nagyobb teret nyer, bármilyen munkát akarunk végezni, valamint otthonaikat már el sem tudnánk képzelni lámpa, mosógép, wifi, TV és egyéb háztartási eszközök nélkül. Ezért nagyon fontos az, hogy hogyan és mennyi áramot termelünk.

A két fő kategória, a megújuló és a nem megújuló között, azon kívül, hogy csak a megújuló terem újra emberi idő léptékkel, az is nagy különbség, hogy az egyes termeléseket, milyen pontossággal lehet megjósolni. A kőolajból és földgázból való energia előállítás során, pontosan meg lehet mondani azt, hogy adott mennyiségű fűtőanyagból mennyi elektromos áramot tudunk generálni. Ezzel ellentétben, viszont a szél sebességét és a napsütés mennyiségét nem lehet pontosan megjósolni.

Az energiatermelés fejlettsége mellett viszont, a tárolásra való megoldások le vannak maradva. Ez azért probléma, mivel a háztartásokban sem egy fix mennyiségű fogyasztás van. Ezen okok, miatt az energiatermelésnek szüksége van egy változó komponensre, amely követni tudja a változó fogyasztást is, ezek jelenleg többségében a gázmotorok, és melléjük csatlakoznak be egyre inkább a megújuló energiaforrások. Ezeknek a termelőknek, hogy mennyit kell termelnie egy központi szervezet, a MAVIR határozza meg. Ahhoz, hogy ezt a meghatározott célértéket tudják tartani, van szükség a szabályzókra. Mivel ők fogják tudni feldolgozni és az egyes erőművek között elosztani ezt az értéket.

A szabályozók tesztelése viszont nem egy egyszerű feladat, mivel legegyszerűbben éles környezetben lehetne tesztelni, viszont ez egyértelmű okokból nem a legszerencsésebb. Ezért szakdolgozatom témájaként egy olyan szimulációt valósítottam meg, mely különböző termelő vagy fogyasztó egységek működését másolja le, ez a digitális iker koncepciója, és ezeken lehessen tesztelni a szabályozók működését.

A szakdolgozatban be fogom mutatni a szabályzók működését, a szimulált egységeket ismertetem, valamint bemutatom a megvalósításhoz használt technológiákat, koncepciókat, majd a megoldásomat fogom részletezni és végezetül pár tovább fejlesztési lehetőséget fogok még bemutatni.

# Bevezetés

Napjainkban a fogyasztott áram mennyiségét nem lehet pontosan meghatározni, mivel például a háztartásokban nem tudjuk pontosan megmondani, hogy adott napon mennyit fogunk fogyasztani. Emiatt van szükségünk szabályzókra, melyek dinamikusan változtatják a telepek energia termelését.

A szakdolgozatom célja, hogy egy ilyen szabályzót lehessen kontrolált környezetben tesztelni, egy szimuláción keresztül. Ennek paraméteri, hogy mely termelő vagy fogyasztó egységek szerepeljenek a szimulációba, hogy mi legyen a milyen random értékkel induljon ezzel befolyásolva a végeredményeket, hogy valós idejű legyen-e a szimuláció, valamint egy objektum, amiben a konstans értékeket lehet beállítani.

A második fejezetben a szabályzó feladatát és működését fogom bemutatni és azt, hogy az általam megvalósított szabályzó milyen logika alapján működik. Ezek mellett bemutatom az egyes szimulált egységek feladatát és működését, mint például az invertert, a terhelőt (loadbank), a gázmotort és az akkumulátort.

Utána a szimulációkról fogok általánosságban írni, valamint bemutatom részletesen a digitális iker koncepcióját, majd kifejtem a kettő közötti különbségeket.

A negyedik fejezetben a használt technológiákkal fogom folytatni. A Kotlin nyelven írtam, így ezt fogom majd részletezni, belemerülve a nyelv által nyújtott DSL (Domain Specific Language)-be is. Utána a keretrendszert fogom bemutatni melyet használtam, valamint beszélek róla, milyen élményeim voltak vele. Ez egy nyitott forráskodú projekt, amely a Kalasim névre hallgat. Technológiák közé fog még tartozni a Gradle, IntelliJ IDEA, Koin , Kotlin Coroutines.

Ezek után kezdem el kifejteni a szimulációt és a benne lévő megoldásaimat, valamint futási eredményeket is mutatok majd diagrammok formájában.

A végén pedig kitérek, hogy milyen fejlődési lehetőségei vannak még a szimulációnak.

# Szabályzó és szimulált egységek

Ebben a fejezetben fogom bemutatni általánosságban a szabályzók feladatát, működését, és hogy honnét kap parancsot, valamint az irányított egységeket. Ezeket pedig diagramokkal kiegészítve és elmagyarázva. Itt fogom még részletezni az általam megvalósított teszt szabályzót is.

## Szabályzó

A szabályzónak alapvetően két oldala van. Az egyik oldalról van a MAVIR, a másik oldalról pedig az egyes napelemparok, szélturbinák és egyéb erőművek. Ezen két oldal közötti kapcsolatot a szabályzó valósítja meg.

### MAVIR

A MAVIR (Magyar Villamosenergia-ipari Átviteli Rendszerirányító) a magyarországi villamosenergiai-hálózat üzemeltetője. Feladatai közé tartozik a hazai villamos energiai-piac működtetése és ellenőrzése, az energiaellátás biztosítása, valamint a hálózat üzemeltetése karbantartása és fejlesztése1.

A graph showing different colored lines

Description automatically generated

**2.1 Ábra: 11.08-11.23 közötti szükséges árammennyiség2**

A 2.1-es ábrán látható, hogy egyes napokon mennyit fogyasztott az ország. A felső lila réteg az importált mennyiség, ha ezen túl lóg a sárga napelemek által termelt összeg, akkor exportról beszélünk. Ez az ábra szépen bemutatja a MAVIR azon feladatát, hogy az ország energiaellátását biztosítsa.

MAVIR a szabályzással azt akarja elérni, hogy a termelés és a fogyasztás egyenlőek legyenek. Ennek fontossága, hogy a villamosenergia-rendszereknek a frekvenciája állandó legyen. A Magyarországi frekvencia értéke 50 Hz. A termelés és fogyasztás kilengései tudják változtatni, ezt a 2.2 ábra mutatja be. Annak magyarázata, hogy mért szükséges az 50Hz túl mutat ezen a szakdolgozaton.

A diagram of a power line

Description automatically generated

**2.2 Ábra: A termelés és fogyasztás hatása a frekvenciára4**

A szabályzásnak alapvetően három fő fajtája van. Én a szakdolgozatom keretein belül csak az egyiket fogom részletesen bemutatni. Az első fajta az elsődleges tartalék (FCR), ez harminc másodpercen belüli stabilizálásra szolgál. A következő, amit későbbiekben jobban a kifejetek, az automatikus kiegyenlítő szabályzási szolgáltatás (aFRR), ezt az elsődleges tartalék után minél hamarabb elindítják, ennek hét és öt percen belül kell, hogy lehessen aktiválni. Az utolsó kategória a manuális kiegyenlítő szabályzási szolgáltatás (mFRR), amit pedig tizenkét és fél percen belül.4

A graph of different colored triangles

Description automatically generated

**2.3 Ábra: Hogyan aktiválódik egymáshoz képest a FCR, az aFRR és az mFRR 10**

A graph showing a number of green and yellow bars

Description automatically generated with medium confidence

**2.3 Ábra: 11.17-11.18 közötti országos aFRR szabályzás 15 perces bontásban5**

A 2.3-as ábrán az látható, hogy az automatikus kiegyenlítő szabályzási szolgáltatás hányszor használták egy huszonnégy órás periódusban. Ha a nulla felett van akkor, fel szabályzásról beszélünk, tehát az országban többlet fogyasztás lépett fel. Ha pedig nulla alatt van, ennek az ellentétjéről beszélünk.

### Szabályzó kliensei

A piacon felelhetőek termelő és fogyasztó felek feladata, hogy meg mondják, hogy mennyit fognak termelni és/vagy fogyasztani.

A termelők ezt a predikált értéket kell betartaniuk mivel, ha ezt nem tudják megtenni akár büntetés is járhat érte. A szabályzónak is ez az egyik feladata, hogy ezeket az eltéréseket minimalizálja. Ennek elősegítésére, hogy az esetleges különbséget könnyebben ellehessen osztani erőműveket és akár fogyasztókat is egybe lehet rakni egy vagy több mérlegkörbe.

A mérlegkör a kiegyenlítő energia igénybevételének okozathelyes megállapítására és elszámolására és a kapcsolódó feladatok végrehajtására a vonatkozó felelősségi viszonyok szabályozása érdekében létrehozott, egy vagy több tagból álló elszámolási szerveződés3.

### Szabályzó feladata

Most, hogy a szabályzó két oldalát ismerjük, ahonnét kapja a bemenetei parancsait, valamint ahova továbbítja a kimeneti szabályzásokat. Így könnyeb lesz megérteni, pontosan, hogy is működik.

Az erőművek alapvetően csoportban vannak és ezek közösen vannak szabályozva. Ezek a csoportok az alapján alakulnak ki, hogy az adott park kit kér fel, hogy elvégezze számára a szabályzást. Ez az aggregátor fél később megy a MAVIR-hoz, hogy bejelentse mérlegkörét.

A szabályzó első rétege lebontja erőművekre, tehát egyes erőműnek összesen mennyit kell termelnie. A következő réteg pedig, már a parkon belül termelő és fogyasztó egységek szintjén fogja meghatározni, hogy az egy-egy elemnek mennyit kell termelnie. Ezen értékeket olvassa a szabályzó, aminek hatására az első szinten visszacsatolást kap ezzel pontosítva a szabályzást.

## Szimulált egységek

Ebben a fejezetben bemutatom az egyes egységeket, amiket a szimulációban meg kellett valósítanom és elmondom, milyen feladataik vannak.

### Inverter

Az inverter az egyenáramot váltóárammá alakítja. Esetünkbe ezeket napelemparkokban találjuk meg.

A napelemek alacsony feszültségű egyenáramot generálnak. Ez nem megfelelő a háztartási gépek és berendezések számára. Az inverter megnöveli a feszültséget és váltóárammá alakítja át az energiát, így ezt már közvetlenül fogyasztható. Ezen az eseten kívül még, ha a közműhálózatba tápláljuk vissza a termelt áramot, akkor is ennek az eszköznek a feladata a hálózatnak megfelelő állapotú áram előállítása6.



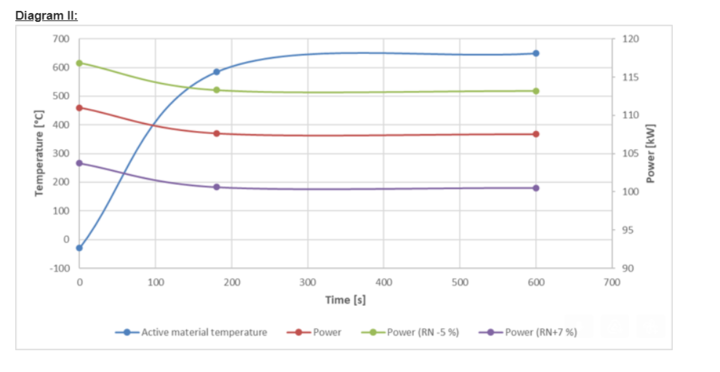
**2.5 Ábra: Napelem melletti inverter7**

Az invertereknek két része van. Az első a DC/DC konverter, mely az egyenáramot még először egyenárammá alakít és eltárolja. A második része a DC/AC konverter, amely már az egyenáramból előállítja a váltóáramot.

### Terhelő (Loadbank)

A terhelőegység egy olyan eszköz, amelyet arra terveztek, hogy elektromos terhelést biztosítson a generátorok, akkumulátorok és feszültségmentesítő tápegységek teljesítményének teszteléséhez és validálásához. Fő célja egy elektromos terhelés szimulálása azzal a céllal, hogy az ne legyen random és kiszámíthatatlan, ez által tervezetten lehessen tesztelni.8

A szabályzás szempontjából viszont, ez segíteni fog, hogy minél gyorsabban el lehessen érni a megfelelő termelés értékét leszabályzás esetén. Ha jön egy leszabályzási parancs akkor elindul és a termelt áram egy részét elfogyasztja, hogy az erőműből a hálózatra parancsértékhez közelebbi érték menjen ki.



**2.6 Ábra: Terhelők ellenállásának nagysága hőmérséklet függvényében**

A 2.6- os ábrán látható a terhelők ellenállása a hőmérséklettel arányosan. Hideg induláskor láthatóan, nagyobb a felvett teljesítmény és ekkor kisebb a generált ellenállás is. Felmelegedés után tud beállni konstans áram felvételre, hőmérsékletre és generált ellenállásra. Ennek következménye, hogy a terhelően belüli komponensek nem indulhatnak egyszerre, mivel túl nagy lenne a felvett teljesítmény. Ezeket a követelményeket megkellett valósítanom a szimulációban.

### Gázmotor

Gázmotor hasonló szerepet tölt be, mint az inverterek, tehát egy termelő egység, viszont felépítésben teljesen eltérnek. A gázmotor egy belsőégetésű motor, ami gáz halmazállapotú tüzelőanyaggal működik.



**2.7 Ábra: Gázmotor**

Több jellegzeteségei is van melyet a szimulációban meg kellett valósítani. Az első ilyen, hogy csak akkor indul el, ha egy bizonyos százalék fölött fog termelni. A másik pedig, hogy a motornak van felmelegedési ideje, ameddig nem termel áramot.

### Akkumulátor

Az akkumulátor különlegessége, hogy mindkét szerepet feltudja venni, tud fogyasztó és termelő egység lenni. Így ez az eszköz szabályzás szempontjából egy hasznos elem, hiszen bevethető le és fel szabályzáskor is.

## Saját szabályzó

A szimuláció bemutatására és tesztelésére létre kellett hoznom egy saját szabályzót. Ez a már elmondottakkal megegyező felépítésű, tehát felső réteg felel, hogy a parkoknak összesen mennyit kell termelniük. A második réteg pedig azért, hogy az egyesével az egységeknek, mennyit kell termelniük, vagy fogyasztaniuk.

A szabályzó, amire a szimuláció tervezve lett egy aFRR szabályzó, aminek egy tulajdonsága, hogy két másodpercenként lehet vele az erőműveknek parancsot küldeni.

### Erőmű szabályzó

Ezzel a MAVIR-t akarom szimulálni, ahogy adott időközömként küldenek parancsot a le vagy felszabályzásra. Ezen felül itt valósul meg az, hogy a parkokra bontsam a parancsban küldött összeget.

Ez egy külön program, melybe függőségként a szimulációt bele raktam. Saját kézzel tudok definiálni egységeket melyeket szimulálja. Valamint meglehet határozni, hogy szabályzás mikor történjen milyen értékkel. Azt, hogy mikor milyen értékkel kell dolgozzon azzal határozom meg, hogy hány másodperc telt el, nem pedig dátummal.

A szimulációt pedig külön coroutine-ban indítom el, ezzel új szálat indítva. A főszálon közben pedig 2 másodpercenként küldöm neki az éppen beállított értéket.

Ez egyértelműen egy le egyszerűsített verzió, melyben a lényeg az, hogy változásokra, hogy reagál a rendszer. Nincs visszacsatolás a kiolvasott értékre.

### Egység szabályzó

Az egység szabályzó feladat, hogy a parkokra bontott szabályzó értékeket, tovább ossza az inverterek, motorok, terhelők és akkumulátorok szintjére. Ez a valóságban az erőművekhez kihelyezett router feladata lenne. Erre, most nincs lehetőségem, hogy az egész router logikáját valahogy beültessem a rendszerbe, így megírtam a saját egyszerűsített verziómat.

Ez a szabályzó a szimuláción belül van megvalósítva.

Minden egységre külön kellett megvalósítanom a logikát, hiszen nagyon eltérő, hogy a parancsokra, hogy viselkednek. Az első az inverter volt, itt a legegyszerűbb módot választottam. Átlagot vontam a parkban fellelhető egységek kimeneti kapacitásából, majd ezek alapján egyenlően elosztottam, hogy mennyit termeljenek.

A következő a terhelők logikája volt. Ez akkor fog történni, ha a szabályozott érték kisebb, mint a jelenlegi célérték. Megnézi, hogy lehet-e terhelőt indítani, ha lehet akkor maximum értéket állítja be. Az, hogy elindíthatja-e, attól függ, hogy az az első, amit indít. Vagy pedig, hogy az előző, ami el lett indítva túllépet- e egy bizonyos százalékot az egység hőmérséklete.

A gázmotorok esetében, hasonló megoldást választottam, mint az inverterek esetében.

Az akkumulátornak hogyha a szabályozott érték nagyobb, mint az előző célérték, akkor elkezd termelőként viselkedni, tehát elkezdi magából kiengedni az eltárolt áramot. Viszont abban az esetben, ha már beállt vagy éppen leszabályzás történik, elkezd tölteni, egyes esetben a többi termelő egység termelését megnöveli annyival, amennyivel az akkumulátor tölteni tudja magát.

# Szimulációk

Ebben a fejezetben általánosságban fogok írni a szimulációk felhasználásáról és kifejtem pár fajtáját, majd rátérek az általam használt típusra. Ezek után pedig bemutatom a digitális iker koncepcióját.

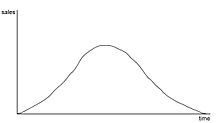
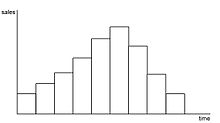
## Szimulációkról általánosságban

A számítógépes szimulációk egyre több és több iparágban nyernek szerepet, gondoljunk csak az egészségügyre, ahol például a sebészek könnyebben betudnak tanulni ennek segítségével. Gondolhatunk akár az autó versenyzésre is, ahol a pilóták drága benzin és egyéb nyersanyagok nélkül tudják megtanulni, a másnapi verseny pályáját. Ezen felül még az űr iparra is gondolhatunk, ahol egy szimuláció segítségével letudják tesztelni, hogy a rakéta sikeresen kifog-e jutni az űrbe. Esetünket nézve az energiaiparba szabályzó kódunkat tudjuk tesztelni anélkül, hogy félnünk kéne, hogy esetleg Budapesten áramkimaradás lenne.9

A szimuláció alapvetően egy valós világban létező folyamatnak vagy egy rendszernek a mása, melyet meglehet figyelni és valós adatokkal összehasonlítani. Általában teljesítmény növelésre, optimalizálásra, biztonság fejlesztésre és tesztelésre szokták használni.

A szimulációkat többféleképpen lehet kategorizálni. Az egyik módja például a kimenetel kiszámíthatósága alapján. Ez alapján lehet egy szimuláció sztochasztikus (véletlenszerű), tehát azonos paraméterekkel nehezen tudunk két ugyan olyan végeredményű futást reprodukálni. Ebből a szempontból másik lehetőség az, hogy a rendszer determinisztikus, tehát ha nem változtatjuk a változókat akkor a szimuláció azonos eredményt ad.

Másik lehetséges kategória az, hogy folytonosan megy vagy diszkrét lépsekkel halad. A folytonos szimuláció differenciálegyenleteket használ, még a diszkrét ütemezett eseményeket használ.



**2.7 Ábra: Bal oldalon diszkrét, Jobb oldalon folyamatos szimuláció ábrázolása11**

## Digitális iker (Digital Twin) koncepció

Ebben a fejezetben bemutatom a digitális iker koncepciójának történelmét, valamint leírom pontosan mit is foglal magába, fajtáit, majd leirom, hogy az én feladatomban, hogy jelent meg.12

### Történelme

A digitális iker koncepcióját David Gelernter Mirror Worlds című 1991-es könyve vetítette előre. A könyvében arról beszél, hogy a virtuális környezetek digitálisan tükrözik a valóságos világot, lehetővé téve az interaktív megfigyelését és manipulációját.

A koncepciót először 2002-ben Michael Grieves mutatta be a Society of Manufacturing Engineers konferencián. Ekkor Grieves, mint termékéletciklus-menedzsment modellként javasolta azt.

2010-ig sok néven ismerték, majd ekkor John Vickers a NASA mérnöke Roadmap reportjában „digital twin”-ként hivatkozott rá, ekkortól terjedt el ez a név.

### Digitális iker

A digitális iker egy virtuális reprezentációja egy fizikai rendszernek, amely folyamatosan frissül az aktuális adatokkal. A valóságban megtalálható félt lehet fizikai ikernek is nevezni. A digitális ikret a fizikai rendszer fejlesztésére, üzemeltetése és karbantartása során használják.

A digitális iker koncepciónak három fő eleme van. Ebből kettőt már említettem. Az első a fizikai iker, a valós világban felelhető rendszer vagy folyamat. A második a digitális iker, amely az előző szoftver formában. A harmadik pedig az adat, ami összeköti az előző kettőt.

A two airplanes in the sky

Description automatically generated with medium confidence

**2.7 Ábra:Digitális iker elemeinek illusztrációja13**

A digitális ikreknek három fajtája van, amely megmutatja, hogy a koncepciót milyen helyzetekben lehet használni. Az első a prototípus (DTP), amelyet akkor használnak mikor még a fizikai termék nem készült el. A következő a példány (Instance - DTI), ezt akkor használják mikor már létezik a termék és azon szeretnének különböző teszteseteket lefuttatni. Az utolsó pedig aggregátum (Aggregate - DTA), ez a fajta több DTI-t foglal magában, amelyek adatai és információi felhasználhatóak a fizikai termékkel kapcsolatos lekérdezésekre és prognosztikákra.14

Nem csak használat alapján lehet kategóriákra bontani, hanem az alapján is, hogy a rendszernek mekkora részét fogja virtuális térbe átütetni. A legkisebb egység a komponens (Component) szintű, ezzel a rendszer egy kis részét lehet megfigyelni. A következő az az eszköz (Asset) szintű, amely több komponenst foglal magába. Az utolsó előtti nagy egység a rendszer vagy egység iker (System or Unit), amely több eszköz összevonásával egy működő rendszert lehet létrehozni virtuális térben. A legnagyobb egység a folyamat (Process), melyben egész gyárakat tudunk akár létrehozni és megnézni, hogy az egész, hogy fog együtt működni.15

A koncepciónak van pár tulajdonsága, amely megkülönbözteti, más technológiáktól. Az első ilyen a kapcsolódás. Gondolva arra, hogy a digitális iker a valóság és virtuális világ között kapcsolatot alakít ki. Ez lehet akár két irányú, vagy csak egy irányú. itt gondolva arra, hogy a létező elem kihatással van a digitálisra és ez visszafele is igaz lehet.

A következő jellemző, az adatokra vonatkozik, ez a homogenizáció. Lényege, hogy a fizikális egységről sok adat tárolása nehéz, ezért az ezekkel való munka nem egyszerű. Ellentétes oldalán viszont a digitális megfelelőjére könnyen lehet nagy mennyiségű adatot tárolni, melyeket közös formára is tudunk hozni, ezzel az adatokat még függetlenítjük is a valós egységtől, ezzel jobban páraméterezhető és tesztelhető lesz.

Az adatokhoz kapcsolódó tulajdonság még, hogy a szimulálás során, megtudjuk mondani, hogy több adatot állítson számunkra elő, mint a valóságos egyed, ezzel könnyebben tudjuk megtalálni, a hibákat és több információt is tudunk gyűjteni.

A harmadik tulajdonság az újra programozhatóság. Ezt úgy kell értelmezni, mivel a fizikai egységet program formájában létrehozzuk, így az olcsón módosíthatóvá válik. Ezzel kapcsolatban van a modularitás, amely segít abban, hogy meglehessen figyelni, hogy mely modulok cserélésével a rendszer, hogyan változik.

## Digitál iker és szimulációk összehasonlítása

Habár mind a szimuláció is és a digitális ikret is arra használjuk, hogy létrehozzunk egy digitális modellt, mellyel helyettesíteni tudjuk egy rendszert, a digitális iker sokkal gazdagabb tanulmányozás szempontjából.

A fő különbség a kettő között a méret. A szimulációk során általában csak egy folyamatot figyelünk meg, még a digitális iker tud magában több szimulációt futtatni, ezzel több folyamat vizsgálatát lehetővé téve, valamint az egymásra való hatást is mutatva. Így ez inkább egy környezetként fogható fel.

Másik különbség a valós idejű adatok felhasználása, melyet egy szimuláció általában nem használ fel, de digitális iker esetén, például mikor a szenzorok által beolvasott adatot felhasználja a virtuális más, az olyan adatokat küldhet vissza a valós rendszerbe, mely befolyásolja annak működését.

Konklúzióként a digitális iker is egy szimuláció, tehát tulajdonságaival jellemezhető, viszont összetettebb, ez által komplexebb feladatok elvégzésére alkalmasabb.

## Megvalósítás

Itt szeretném leírni milyen jellemzői vannak megoldásomnak, szimulációs és digitális iker szempontból is.

Szimulációt nézve egy determinisztikus diszkrét esemény vezérelt rendszerről fogunk beszélni.

Digitális iker felől pedig egy DTA-ról beszélünk, melyben az egyes DTI-k a szimulált inverterek, terhelők, gázmotorok és akkumulátorok. Valamint egy rendszer szintű ikerről tudunk beszélni, mivel a termelő egységek virtuális másolatát valósítottam meg, így például a teljes szabályzó nem került bele.

# Felhasznált technológiák

Ebben a fejezetben felsorom milyen technológiákat használtam fel a szimuláció megvalósítására, azokat részletezem és kiemelem, milyen elemei voltak melyeket feltudtam használni.

## Kotlin

A Kotlin egy modern, expresszív programozási nyelv, melyet a JetBrains fejlesztett ki. Nyilvánosságra 2011-ben hozták, 2012-ben pedig nyílt forráskódú lett, 2017-ben pedig az Android hivatalos nyelve lett. 16

A Kotlin olyan nyelvként szolgál, amely támogatja mind szerveroldali, mind a kliensoldali fejlesztést. JVM felett fut, ami lehetővé teszi a Java kódokkal való kompatibilitást, ennek köszönhetően rendelkezik a Java minden tulajdonságával, de szebb szintaxissal rendelkezik és több olyan beépített funkció van mely segíti a fejlesztők munkáját. Kliens oldali felhasználásában pedig, a Kotlin tud fordulni Javascript-re is.

### Kotlin DSL

A Kotlin egyes funkcióit közösen használva tudunk létrehozni Domain Specific Language-ket (DSL). Ezek akkor lehetnek hasznosak mikor komplex objektumokat szeretnénk létrehozni, vagy konfigurációk létrehozására. Nem könnyű őket létrehozni, de olvashatóság szempontjából mindenféleképpen megéri, mivel elrejti a kódunk komplexitását és sok sablonkódot.17

A DSL-hez szükséges Kotlin funkciók közé tartozik a függvény típusok. Ez a típus egy olyan objektumot reprezentál, melyet lehet függvényként használni. a deklarációja példaként, a Kotlin filter függvényét hozom.

inline fun <T> Iterable<T>.filter(

predicate: (T) -> Boolean

): List<T>

A példában a predicate függvény típusú. A zárójelen belüli a függvény paraméter, a nyíl utáni pedig, hogy milyen típussal térjen vissza. Ha csak egy bemeneti paraméter van arra it-ként lehet hivatkozni. Használatára is hozok példát.

val numbers = listOf(0,1,2,3,4,5,6,7,8,9)

val evenNumbers = lnumbers.filter{ it % 2 == 0}

// evenNumbers = [0, 2, 4, 6 ,8]

Másik szükséges Kotlin funkció az extension function (kiterjesztési függvény), melynek segítségével kitudunk egészíteni osztályok függvényekkel anélkül, hogy lekéne származni az osztályból, vagy valamilyen Dekorátor mintát kéne használni. Extension function tudunk függvény típusként is meghatározni, ez fontos eleme a DSL-nek.18

Ezt a 2 funkciót használva, valamint kiegészítve Builder mintával tudunk létrehozni DSL-t.

class Sample(private val value) // Sample osztály

class SampleBuilder{ // SampleBuilder builder Sample classnak var value: Int? = null

fun build(): Sample{

return Sample(value)

}

}

fun sample(init: SampleBuilder.() -> Unit): Sample{ // DSL függvény

return this.init().build()

}

val newSample = samlpe{ //DSL függvény teszt

value = 1

}

Ezt a szimuláció konfigurációs objektumához használtam.

### Kotlin Coroutine

A Kotlin coroutineok egy hatékony és kifejező aszinkron programozási eszköz, amely lehetővé teszi a szálak nélküli, non-blocking kód írását. A coroutineok segítségével könnyedén kezelhetjük az aszinkron műveleteket, például hálózati hívásokat vagy adatbázis műveleteket, anélkül, hogy blokkolnánk a főszálat. A suspend kulcsszó segítségével felfüggeszthetjük a futást egy függvényen belül, és a launch vagy async függvények segítségével hozhatunk létre coroutineokat.

Ezt arra használtam, hogy a szimuláció elindítása után tudjak, szabályzási parancsokat küldeni, amik befolyásolták termelés mennyiségét.

### Gradle

A Gradle egy erőteljes, rugalmas nyílt forráskódú build-automatizált rendszer és projektépítő eszköz, amely széles körben használt a szoftvertervezési és fejlesztési folyamatokban. A Gradle támogatja a többnyelvű projekteket, például Java-t, Kotlin-t, és még más programozási nyelveket, és lehetővé teszi a felhasználók számára, hogy hatékonyan építsenek, teszteljenek és telepítsenek alkalmazásokat.19

Az eszköz erőssége a moduláris és konvencionális projektstruktúrák könnyű támogatása, valamint a jól konfigurálható és testre szabható felépítési folyamatok lehetősége. A Gradle továbbá képes automatizálni a projekt függőségek kezelését, valamint támogatja az inkrementális fodítást, ami gyorsítja a fejlesztési folyamatokat.

Alternatívája lehetne a Maven, de mivel a Gradle újabb verziója Kotlin-t használ így ez adta magát.

### Kotlin Data Frame

A Data Frame egy absztrakció a strukturált adatokkal való munkához. Alapvetően ez egy 2 dimenziós tábla különböző típusú és elnevezett oszlopokkal. Hasonlóan, mint egy SQL tábla.20

Legnagyobb erősége nem az absztrakcióban rejlik, hanem a Data Frame könyvtár által megvalósított műveleteken, amik megkönnyítik a munkát az ilyen típusú adatokkal.

A szimuláció közbeni log-ok .csv fájlba való kiírásához használtam.

### IntelliJ IDEA

Az IntelliJ egy integrált fejlesztő környezet (IDE), amit Java-ban írtak, abból a célból, hogy számítógépes programokat fejlesszenek benne Java, Kotlin és egyéb JVM alapú nyelveken. Kotlin-hoz hasonlóan ez is a JetBrains fejlesztése.

Ebben a fejlesztőkörnyezetben sok hasznos funkció található, melyek segítik a fejlesztő munkáját. Ezen funkciók közé tartozik az integrált adatbáziskezelő és git.

IntelliJ IDEA-t használtam a fejlesztéshez, ami egyértelmű választás volt, hiszen a Kotlin-nal, ez a legkompatibilisebb, valamint ezt használtam, az egyetem és munka közben, Java és Kotlin fejlesztéshez is.

## Kalasim

Ebben a részben bemutatom a keretrendszert, melyet a szimuláció elkészítéséhez használom. Először általánosan beszélek róla, majd bemutatom, az általam használt elemeket.

### Általánosságban

A Kalasim egy Kotlin-ban írt nyílt forráskódú keretrendszer, mely egy diszkrét eseményszimulátor. Holger Brandl fejlesztette, akinek a Kotlin Data Frame elődjét is köszönhetjük. A keretrendszer nagyon jól kidolgozott dokumentációval rendelkezik. 21

Én a keretrendszer v0.11.5-ös verzióját használtam.

A diszkrét eseményszimulátor olyan eszköz, mely lehetővé teszi a sztochasztikus, dinamikus és diszkrét módon változó rendszerek dinamikus viselkedésének tanulmányozását. Ilyen rendszerek például gyárak, kikötők, repterek és vezérlés/szabályzás. Ezen kívül tágabb felhasználási körei közé tartoznak a termék tervezés, folyamat automatizálás és vizualizáció, projekt menedzsment és a digitális iker fejlesztés.22

A Kalasim azon szimulációs szakemberek, folyamatelemzők és ipari mérnökök számára készült, akiknek túl kell lépniük a meglévő szimulációs eszközök korlátain, hogy modellezés és optimalizálják üzleti szempontból kritikus felhasználási eseteiket.

Összehasonlításképpen más szimulációs eszközökhöz képest a Kalasim nem low-code és nem is no-code modell alapján működik, hanem code-first mentalitásban. Ez lehetővé teszi a változás követést, scaling-et, refaktorálást, CI/CD-t, unit tesztelést és még sok mást.

A Kalasim, mint már említettem Kotlin nyelven íródót. Egyik fő jellemzőjét használja ki a nyelvnek a Coroutine-et, mellyel a folyamatokat definiálják. JVM-en fut a teljesítmény és scaling miatt. Koin függőség injektálási keretrendszert használ. Valamint használja az Apache Common Math könyvtárat.

Azért ezt a keretrendszert választottam, mivel a fejlesztéshez választott nyelvem a Kotlin volt, valamint nagyon sok olyan funkciója volt, melyet feltudtam használni és miközben kerestem, ezt találtam a legjobb választásnak.

### Fő funciók

Kalasim egy generikus folyamat orientált diszkrét eseményszimulátor, így ehhez kapcsolódóan sok olyan funkció található benne, mely megkönnyíti a feladatok definiálását.

Az első fő funkció a szimulációs entitások. Ezek a Component névre hallgatnak. Bennük lehet folyamatokat definiálni, amelyek vagy aktiválásra vagy pedig folyamatosan időközönként futnak le.

object: Component() {

override fun process() = sequence {} // aktiválásra használt

override fun repeatedProcess() = sequence{} // ismétlődő folyamat

}

Másik fontos funkció a gazdag függvény paletta, mellyel a folyamatokkal tudunk interaktálni. Négy ilyen függvény van, ezek közül nekem egyre volt szükségem, a hold-ra ez egy meghatározott szimulációs időre felfüggeszti a folyamatot.

object: Component() {

override fun process() = sequence {

hold(10.minutes) // felfüggeszti szimulációs időben 10 percre

}

A többi függvény a request ez Resource-hoz kapcsolódó feltételt ellenőriz, a wait ez adott állapothoz tartozó feltételt vár, az utolsó pedig a passivate, ezzel egy komponenst passzív állapotba lehet helyezni.

Másik fontos funkciója a Kalasimnak az események időzítése, melyek úgy tűnnek, hogy egyszerre futnak le viszont az egyszálú működés miatt a folyamatok egymás után egyesével fognak lefutni. A Kalasim egy priorityQueue-t használ ehhez.

A screenshot of a computer

Description automatically generated

**4.1 Ábra: Kalasim végrehajtási modell**

Valamint a Kalasim beépített montoring és statisztika gyűjtővel rendelkezik.

### Szimuláció konfiguráció

Szimulációk konfigurálásához is sok elemet nyújt a keretrendszer. Erre az Enviroment osztályt használja. Ezen belül belehet állítani, hogy egy Tick (szimulációs idő egy egysége), mennyinek felejen meg a valóságban. Valamint azt is állítható elem, hogy a komponenseket logolja-e.

Az alapvető szimuláció létrehozásához van lehetőségünk DSL-t használni viszont, ha összetettebb, számunkra megfelelő szimulációs környezetet szeretnénk, akkor az Enviroment osztályból kell leszármaznunk.

Fontos feladat volt még a random értékek kezelése. Erre is van beépített funkció az Enviroment osztályon belül lehet állítani a randomseed-et ezt, ha nem változtatjuk, akkor a szimulációk, mindig ugyan olyan eredménnyel futnak le.

### Nehézség

Ez igazából csak tapasztalat forrás. Ez volt az első, hogy úgy használtam nyílt forráskódú rendszert, hogy annak forráskódját megnéztem, és megpróbáltam úgy átalakítani, hogy azzal megoldjam a problémám. Ez a probléma a Koin függőség injektáló rendszer köré összpontosult.

Ez abból adódott, hogy a Koin minor verzió váltások között nem tart bináris és viselkedésbeli kompatibilitást. Számomra a Koin 3.3.3-as verziója kellett még a keretrendszer a 3.1.6-at használta.23

A különbség a verziók között a lazy loading-ban volt. Régi verzióban a module-ban való felsoroláskor megtörtént az inicializálás, az újabb verziókban viszont, első használatkor fog.

Ennek megoldását, nem a forráskódban találtam meg. Az lett a végeredmény, hogy .jar fájllá alakítottam a függőségei nélkül, majd a szükséges függőségeit utólag adtam, meg a saját projektemben.

1 - [Gyakran ismételt kérdések - MAVIR - Magyar Villamosenergia-ipari Átviteli Rendszerirányító Zrt.](https://www.mavir.hu/web/mavir/gyakran-ismetelt-kerdesek)

2 - [Szabályozási Adatok Kiegyenlítő és Nem Kiegyenlítő Szabályozás céljából - MAVIR - Magyar Villamosenergia-ipari Átviteli Rendszerirányító Zrt.](https://www.mavir.hu/web/mavir/szabalyozasi-adatok-kiegyenlito-es-nem-kiegyenlito-szabalyozas-celjabol) Erőművi termelés primmer források szerinti megoszlás és az import-exposrt szaldó – bruttó üzemirányítási mérések alapján

3 - [partner.mvm.hu/hu-HU/Nagykereskedelem/Merlegkor-menedzsment](https://www.partner.mvm.hu/hu-HU/Nagykereskedelem/Merlegkor-menedzsment)

4 - [Manuális Kiegyenlítő Szabályozási Szolgáltatás (mFRR) | Nano Energies: Tegye eredményesebbé vállalkozást az energia hálózat hatékonyabb kihasználásával](https://nanoenergies.hu/tudastar/manualis-kiegyenlito-szabalyozasi-szolgaltatas-mfrr)

5 - [Szabályozási Adatok Kiegyenlítő és Nem Kiegyenlítő Szabályozás céljából - MAVIR - Magyar Villamosenergia-ipari Átviteli Rendszerirányító Zrt.](https://www.mavir.hu/web/mavir/szabalyozasi-adatok-kiegyenlito-es-nem-kiegyenlito-szabalyozas-celjabol) – Aktiválás kiegyenlítő szabályzás céljából

6 - [Az inverter jelentése és felhasználási területei (eon.hu)](https://www.eon.hu/hu/blog/napenergia/inverter_jelentese.html)

7 - [Müllberg Speyer - 2 - Power inverter - Wikipedia](https://en.wikipedia.org/wiki/Power_inverter#/media/File:M%C3%BCllberg_Speyer_-_2.JPG)

8 - [Simplex: Load Bank Fundamentals (simplexdirect.com)](https://www.simplexdirect.com/loadbanksEducation.aspx)

9 - <https://en.wikipedia.org/wiki/Simulation>

10 - <https://nanoenergies.hu/tudastar/mari-picasso-es-terre>

11 - <https://en.wikipedia.org/wiki/Continuous_simulation>

12 - <https://en.wikipedia.org/wiki/Digital_twin>

13 - <https://www.techtarget.com/searcherp/definition/digital-twin>

14 - <https://www.twi-global.com/technical-knowledge/faqs/what-is-digital-twin>

15 - <https://www.ibm.com/topics/what-is-a-digital-twin>

16 - <https://en.wikipedia.org/wiki/Kotlin_(programming_language)>

17 – Marcin Moskala: Effective Kotlin BestPractices :: Item 35: Consider defining a DSL for complex object creation

18 - <https://www.baeldung.com/kotlin/extension-methods>

19 - <https://kotlinlang.org/docs/gradle.html#what-s-next>

20 - <https://kotlin.github.io/dataframe/overview.html#what-is-data-frame>

21 - <https://www.kalasim.org/>

22 - <https://www.kalasim.org/theory/> ???

23 - <https://github.com/InsertKoinIO/koin/issues/1369>