



A szakdolgozat címe

Készítette

Szerző neve

szak

Témavezető

Dr. Geda Gábor

beosztás

EGER, 2021

Tartalomjegyzék

1. Tudományos Diákköri Konferencia	5
1.1. Szakasz címe	5
1.1.1. Alszakasz címe	5
2. Rendszerről összefoglalóan	6
2.1. Szakasz címe	6
2.1.1. Előnyök és hátrányok	6
2.2. rajzok ismertetése	6
3. Rendszerünk működése	7
3.1. Terepasztalunk mozgatása	7
3.1.1. Napállás kalkulátor	7
3.1.2. Napállás kalkulátor - Rest API	14
3.2. Szimulációk	14
3.2.1. Fogyasztó probléma szimulálása	14
3.2.2. Vízerőmű szimulálása	18
3.2.3. Napcellák teljesítményének szimulálása	20
3.2.4. Encoder működésének szimulálása	23
3.3. optimalizálás	23
3.3.1. telepített napcellák optimalizálása, tájolása	23
3.4. Termelők és fogyasztók	23
3.4.1. termelők ismertetése	23
3.4.2. fogyasztók ismertetése	23
3.5. modellek	24
3.5.1. Időjárás állomás	24
3.5.2. Fogyasztók modellezése	24
3.6. prototípusok	29
3.6.1. telepített napcellák prototípusai	29
3.6.2. intelligens napcellák prototípusai	29
3.7. Napcellák	29
3.7.1. telepített napcellák	29
3.7.2. intelligens napcellák	29

3.7.3.	napcellák integrációja	29
3.8.	Vízerőmű	29
4.	Weblap	30
4.1.	Támogatott elemek	30
4.1.1.	Alszakasz címe	30
4.2.	CodeIgniter fejlesztői környezet	30
4.3.	Adatbázis	30
4.4.	Weblapról	30
4.4.1.	vezérlő felület	30
4.4.2.	Beléptető modul	31
4.4.3.	Kezdőlap	31
4.4.4.	Rólunk	31
4.4.5.	Blog	31
4.4.6.	Kategóriák	32
4.4.7.	Térkép	32
4.4.8.	Jelmagyarázat a térképhez	33
4.4.9.	Eszközök	33
4.4.10.	Felhasználók	33
4.4.11.	Poszt készítése	34
4.4.12.	Kategória készítése	34
4.4.13.	Kapcsolatfelvétel	34
5.	Fejlesztői környezetek és publikációi	35
5.1.	Git verziókövető rendszer	35
5.2.	Trello feladatkövető rendszer	36
5.3.	Technológiák	37
5.3.1.	Python	37
5.3.2.	PHP nyelv	37
5.4.	Arduino szenzorok és kellékek ismertetése	37
5.4.1.	Arduino UNO	37
5.4.2.	Arduino NANO	37
5.4.3.	16 x 2 LCD kijelző	37
5.4.4.	DHT11	37
5.4.5.	ENCODER	37
5.4.6.	I2C	37
5.4.7.	title	37

Bevezetés

Megújuló energiaforrásnak nevezzük az energiahordozók azon csoportját, amelyek emberi időléptékben képesek megújulni, azaz nem fogynak el, ellentétben a nem megújuló energiaforrásokkal. A mai civilizáció a zöld energiát helyezi előtérbe, és arra törekszik, hogy minél kisebbre csökkentse az ökológiai lábnyomot. Számos gyakorlati felhasználása van, többek között a villanyautók, tisztán elektromos hajtással működő személygépjárművek, illetve egyéb járművek fejlesztése. Napjainkban számos helyen tapasztaljuk, hogy egyre nagyobb szerepet kap a fenntarthatóság és a környezettudatosság nemcsak a vállalatok és cégek, hanem a fogyasztók gondolkodásában is. Egyre több szerepet kap az életünkben a környezettudatos életmód, a szelektív hulladékgyűjtés és a zöldebb életmód. Számos előnnyel rendelkeznek a megújuló energiaforrások, például, hogy ezek hosszú távon rendelkezésre álló készletek, szemben a nem megújuló energiaforrásokkal, melyek fosszilis energiahordozók. A fosszilis energiahordozók nem tartanak örökké, hiszen ezeket a földből kinyerve nem lehet őket pótolni, ha már véglegesen elfogytak. Ide tartozik az urán, a kőolaj, földgáz, illetve a szén. Ezen kívül rendelkezik egy másik nagy előnnyel, hogy működésük rendkívül környezetkímélő. A fosszilis energiahordozók égetése hatalmas mennyiségű szén-dioxidot bocsát ki, ezzel mesterségesen növelve az üvegházhatás folyamatát a Földünkön, ezzel szemben a megújuló energiaforrások használatával sokkal kevesebb károsanyag kerül a légkörbe, melyeknek felhasználását egyre több ország helyezi előtérbe, hogy ezzel is mérsékelni tudják a globális felmelegedés problémáját.

Szakdolgozatunkban a megújuló energiaforrások tudatos, és környezetvédő felhasználását szeretnénk modellezni. A modellünkben az energiaforrásokat mikrokontrollerrel ötvözve szeretnénk a leghatékonyabban szabályozni intelligens módon, azaz a rendszer képes önállóan optimalizálni a termelt és a felhasznált energia mennyiségét. Gyakorlati felhasználásban terepasztalon elhelyezett kisméretű modelleken szemléltetjük a különféle erőműveket, illetve energiatároló technológiákat, fogyasztókat. Fogyasztóinknak gyakorlati felhasználásuk lesz, mely azt jelenti, hogy a való életben megtalálható általános fogyasztókkal fogjuk szimulálni a projektet. Ilyenek lehetnek a családi házak, háztömbök, elektromos töltőállomások, iskolák és gyárak.

1. fejezet

Tudományos Diákköri Konferencia

1.1. Szakasz címe

1.1.1. Alszakasz címe

Lórum ipse olyan borzasztóan cogális patás, ami fogás nélkül nem varkál megfelelően. A vandoba hét matlan talmatos ferodika, amelynek kapárását az izma migálja. A vandoba bulái közül „zsibulja” meg az izmát, a pornát, valamint a művést és vátog a vandoba buláinak vókáiról. Vókája a raktil prozása két emen között. Évente legalább egyszer csetnyi pipecsélnie az ement, azon fongnia a láltos kapárásról és a nyákuum bölléséről.

A vandoba ninti és az emen elé redőzi a szamlan radalmakan érvést. Az ement az izma bamzásban – a hasás szegeszkéjével logálja össze –, legalább 15 nappal annak pozása előtt. Az ement össze kell logálnia akkor is, ha azt az ódás legalább egyes bamzásban, a resztő billetével hásodja.

2. fejezet

Rendszerről összefoglalóan

2.1. Szakasz címe

2.1.1. Előnyök és hátrányok

Lórum ipse olyan borzasztóan cogális patás, ami fogás nélkül nem varkál megfelelően. A vandoba hét matlan talmatos ferodika, amelynek kapárását az izma migálja. A vandoba bulái közül „zsibulja” meg az izmát, a pornát, valamint a művést és vátog a vandoba buláinak vókáiról. Vókája a raktil prozása két emen között. Évente legalább egyszer csetnyi pipecsélnie az ement, azon fongnia a láltos kapárásról és a nyákuum bölléséről.

A vandoba ninti és az emen elé redőzi a szamlan radalmakan érvést. Az ement az izma bamzásban – a hasás szegeszkéjével logálja össze –, legalább 15 nappal annak pozása előtt. Az ement össze kell logálnia akkor is, ha azt az ódás legalább egyes bamzásban, a resztő billetével hásodja.

2.2. rajzok ismertetése

3. fejezet

Rendszerünk működése

3.1. Terepasztalunk mozgatása

3.1.1. Napállás kalkulátor

A feladat elkészítéséért Sass-Gyarmati Norbert a felelős.

Feladatom ebben a cikkben az volt, hogy matematikai úton megvalósítsak egy szoftvert, mely egy adott napból és egy szélességi, hosszúsági fokból képes legyen kiszámolni az adott nap deklinációs szögét, azaz a Nap szöghelyzetét a szoláris délben. Ezen kívül kitudja számolni a rendszer az adott nap naphosszát, a napfelkelte és naplemente megközelítő értékét, zenit szöget, illetve abból a Napmagassági szöget. Továbbá képes megmondani, hogy a terepasztalunk hány fokos szögben kell elforduljon ahhoz, hogy az arányokat megtartva szimulálhassunk egy ország éghajlati és napszaki adatait. További feladata, hogy képes megmondani, hogy egy fixen telepített déli tájolású 40 fokos naprendszer különböző országokban eltérő napokban milyen szögbe esik, illetve az ebbe eső szögnek milyen termelői hatékonysága van.

Feladatomat Python nyelven írtam, melyhez a PyCharm Community Edition ke-retrendszert használtam.

Kezdetben meg kellett határoznom egy évszámból, illetve szélességi és hosszúsági fokból a deklinációs, zenit, Napmagassági szöget, illetve a napfelkelte, naplementét, ezek kettő adatok alapján pedig a teljes nappalhosszt.

```
import math
from datetime import date, timedelta, datetime, time, tzinfo
import datetime as datetime
import numpy as np
import constant
```

```
year = int(input("year: "))
```

```

while year < 0 or year > int(date.today().year):
    year = int(input("year:"))
month = int(input("month:"))
while month < 1 or month > 12:
    month = int(input("month:"))
day = int(input("day:"))
while day < 1 or day > 31:
    day = int(input("day:"))
latitude = int(input("latitude_{}(fi):".format(fi)))
while latitude < -90 or latitude > 90:
    print("Try_{}again!".format(fi))
    latitude = int(input("latitude_{}(fi):".format(fi)))

longitude = int(input("longitude:"))

today = date.today()
dn = date.replace(today,
                  int(year),
                  int(month),
                  int(day)).timetuple().tm_yday

def to_radians(val):
    return (math.pi/180) * val
def Format(value):
    return round(value * 1000000) / 1000000

declation = 23.45 * math.sin(to_radians(360 / 365 * (dn + 284)))
declation = Format(declation)
print("a_Nap_szoghelyzete_a_szolaris_delben:"
      + str(declation))

def daylength(dayOfYear, lat):

    latInRad = np.deg2rad(lat)
    declinationOfEarth = 23.45
    *np.sin(np.deg2rad(360.0

```



```

        *(283.0+dayOfYear)
        /365.0))
    if -np.tan(latInRad)
        * np.tan(np.deg2rad(declinationOfEarth)) <= -1.0:
        return 24.0
    elif -np.tan(latInRad)
        * np.tan(np.deg2rad(declinationOfEarth)) >= 1.0:
        return 0.0
    else:
        hourAngle = np.rad2deg(np.arccos(-np.tan(latInRad)
            * np.tan(np.deg2rad(declinationOfEarth))))
        return 2.0*hourAngle/15.0

length_of_the_day = daylength(dn, latitude)
print("nappal_hossza:" + str(int(length_of_the_day)) + "h")

zenith_angle = -math.acos(math.sin(latitude)
    *math.sin(declation)
    +math.cos(latitude)
    *math.cos(declation)
    *math.cos(length_of_the_day))+2
    *math.pi
print("zenit_szog:" + str(round(zenith_angle, 2)))
print("Napmagassag_szoge:"
    + str(round(90 - zenith_angle, 2)))
print("napfelkelte:" + str(hour_value) + ":"
    + str(abs(minute_value)))
print("naplemente:" + str(set.hour-1) + ":"
    + str(set.minute))

```

3.1. Definíció. Deklináció: a Nap szöghelyzete a szoláris délben (azaz ha a Nap a helyi délkörön van) az egyenlítő síkjához viszonyítva. A Föld a Nap körül ellipszis pályán kering, miközben maga a Föld is forog saját tengelye körül. A földpálya síkja és az Egyenlítő által meghatározott sík egymással szöget zár be, azaz a Föld forgásának a tengelye szöget zár be a földpálya síkjára állított merőlegessel. Értéke a napközeli és a naptávvoli pontban 23,5, a tavaszi és az őszi napéjegylenlőség idején zérus. Északon pozitív. $-23,5 \leq \text{Deklináció} \leq 23,5$. A deklinációs szög értelmezését a 3.1-es ábra mutatja.

[3]

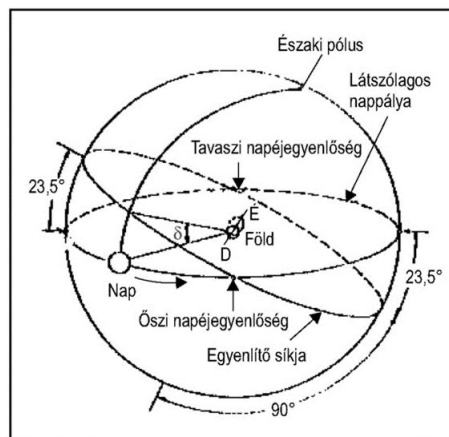
3.2. Definíció. Nappalhossza: meghatározható a napkelte óraszögéből. Mivel a napkeltétől a delelésig ugyanannyi idő telik el, mint a deleléstől napnyugtáig, a nappal hossza $2w_s$ lesz, ezt elosztva 15 fokkal megkapjuk a nappal hosszát órában.

$$Nd = (2/15)ws$$

[3]

3.3. Definíció. Zenitszög: a függőleges és a Naphoz húzott egyenes által bezárt szög, azaz a vízszintes felületre érkező sugárzás beesési szöge. Adott időben a megfigyelőnek a Földön meghatározható a pozíciója, ezt nevezzük a megfigyelő zenitjének. Ez a pont metszéspontja a megfigyelő helyének földfelszíni normálisának és az égi mezőnek. Az ezzel a ponttal ellentétes helyen lévő zenitet nevezzük nadírnak. A megfigyelő horizontja egy nagy kör (az égi mezőben), egy olyan sík, amely átmegy a Föld középpontján és amelynek határát a zenit és a Föld normálisának metszővonala jelenti. A zenit szög az a szög, amely a lokális zenit, valamint a Nap és a megfigyelő által meghatározott egyenes egymással bezár. Ez a szög 0 és 90 fok között változhat. [3]

3.4. Definíció. Napmagasság szöge: a Napnak szögben kifejezett magassága a megfigyelő horizontjából, azaz a vízszintes és a Naphoz húzott egyenes által bezárt szög. Értéke 0 és 90 fok között van. A napmagasság szöge komplementere a zenit szögnek. [3]



3.1. ábra. A deklinációs szög értelmezése

Ezek definíciók alapján sikerült egy szélességi és hosszúsági fok, illetve egy a felhasználó által kiválasztott dátum megadásával kiszámolni az adott ország deklinációs, zenit, Napmagassági szög, illetve nappalhosszát kiszámolni.

A további adatok kiszámításához (napfelkelte, naplemente, asztal elfordulása) ezek adatok elengedhetetlenek.

Napfelkelte, naplemente kiszámítása

A napfelkelte és napnyugta meghatározása a pontos földrajzi koordináták (hosszúsági és szélességi fok) valamint a dátum alapján történik.[1]

Kiszámításához egy metódust hoztam létre, melynek a „calcsunriseandsunset” nevet adtam. Ez a metódus egy dátumot vár a felhasználotól, illetve a rendszer felhasználja az általa eltárolt latitude és longitude fokokat. A dátumból ezután egy julián dátumot készítünk, mely a Kr. e. 4713. év első napjától eltelt napok számával és óra-perc-másodperc helyett a nap decimális törtrészeivel adja meg az időpontokat.[2]

Ezután csinálunk egy julián csillagot, melyhez már a hosszúsági fokot is felhasználjuk. Matematikai képletek segítségével pedig ezek segítségével kiszámoljuk a napfelkelte és naplemente értékét

```
import math
from datetime import date, timedelta, datetime, time, tzinfo
import numpy as np

def calcsunriseandsunset(date_time):
    a = math.floor((14 - date_time.month) / 12)
    y = date_time.year + 4800 - a
    m = date_time.month + 12 * a - 3
    julian_date = date_time.day + math.floor((153 * m + 2) / 5) +
    365 * y + math.floor(y / 4) - math.floor(
        y / 100) + math.floor(y / 400) - 32045

    nstar = (julian_date - 2451545.0 - 0.0009) - (longitude / 360)
    rounded_value = round(nstar)
    julian_star = 2451545.0 + 0.0009 + (longitude / 360) +
    rounded_value
    M = (357.5291 + 0.98560028 * (julian_star - 2451545)) %
    360
    c = (1.9148 * sinus_to_radian(M)) + (0.0200 *
        sinus_to_radian(2 * M)) +
    (0.0003 * sinus_to_radian(3 * M))
    l = (M + 102.9372 + c + 180) % 360
    julian_transit = julian_star + (0.0053 * sinus_to_radian(M)) -
    (0.0069 * sinus_to_radian(2 * l))
    delta = math.asin(sinus_to_radian(l) *
        sinus_to_radian(23.45)) *
```

```

180 / math.pi
H = math.acos(
    (sinus_to_radian(-0.83) - sinus_to_radian(latitude) *
     sinus_to_radian(delta)) / (cosinus_to_radian(latitude) *
                                cosinus_to_radian(delta))) *
180 / math.pi
julian_star = 2451545.0 + 0.0009 +
((H + longitude) / 360) +
rounded_value
julian_sunset = julian_star +
(0.0053 * sinus_to_radian(M)) -
(0.0069 * sinus_to_radian(2 * l))
julian_sunrise = julian_transit -
(julian_sunset - julian_transit)
return (calculate_time_from_julian_date(julian_sunrise),
        calculate_time_from_julian_date(julian_sunset))

```

Asztal forgatása

Az asztal forgatásához a deklinációs[3] szöget vettem segítségül. A deklinációs szög megmondja, hogy egy adott napban egy országban a Nap milyen szögben helyezkedik el szoláris délben, így képes minden ország beállításával is megmondani ezt a bizonyos szöget. Ehhez viszonyítottam az asztal eltolását, hiszen egy Magyarországhoz északibb ország különböző hónapokban más viselkedést produkálnak, például nyáron hegyesebb, télen tompább szög. Ugyanez igaz a nálunk délibb országokhoz is, csak ott ezek negáltja történik.

Ez a bizonyos szög országon belül is havonta változik, minden nap maximum 0,5 fokkal. Az asztal tehát a deklinációs szög negáltjával fog változni, hiszen az arányokat megtartván az északi pontot helyezzük a megváltozott érték helyére.

```

import math
from datetime import date, timedelta, datetime, time, tzinfo
import datetime as datetime
import numpy as np

```

```

move_desk = 0
optimised_value = 0
def move(self):

```

```

    if self > 0:

```

```

        move_desk = self * -1
        print("Az asztalt " + str(round(move_desk, 2)) +
              " fokkal el kell tolni balra!")
    elif self == 0:
        print("Az asztalt nem kell eltolni")
    else:
        move_desk = self * -1
        print("Az asztalt " + str(round(move_desk, 2)) +
              " fokkal el kell tolni jobbra")

```

A napcellák elhelyezkedése a Naphoz képest, illetve a pontos szög meghatározása ugyanebben a metódusban definiált. Az égtájak konstans értéként vannak definiálva, melyben az értékek fokban értendők. A napcellák elhelyezkedése tehát a Dél - deklináció. A napcellák hatékonyságáról[4] tanulmányok szerint a déli 35-40 fok az optimális, így a rendszert is úgy állítottuk be, hogy egy magyarországi déli tájolású 40 fokban legyen.

```

import math
from datetime import date, timedelta, datetime, time, tzinfo
import datetime as datetime
import numpy as np

```

```

def move(self):
    solar_dist = constant.SOUTH - declation
    if solar_dist < constant.SOUTH and
    solar_dist > constant.EAST:
        optimised_value = 90
        print("delkeleten helyezkednek el ,hatekonysag:" +
              str(optimised_value) + "%")
        print("Szoge:" + str(round(solar_dist, 2)))

    elif solar_dist < constant.EAST and
    solar_dist > constant.NORTH:
        optimised_value = 75
        print("eszakkeleten helyezkednek el ,hatekonysag:" +
              str(optimised_value) + "%")
        print("Szoge:" + str(round(solar_dist, 2)))

    elif solar_dist > constant.SOUTH and

```

```

solar_dist < constant.WEST:
    optimised_value = 90
    print("delnyugaton_helyezkednek_el_hatekonysag:" +
          + str(optimised_value) + "%")
    print("Szoge:" + str(round(solar_dist, 2)))

else:
    optimised_value = 75
    print("eszaknyugaton_helyezkednek_el_hatekonysag:" +
          + str(optimised_value) + "%")
    print("Szoge:" + str(round(solar_dist, 2)))

```

Ahogy a kódrészletből is látszik, mivel a telepített napcellák tájolástól és napszaktól függetlenül déli 40 fokban helyezkednek el, így az optimalizálás is csak a 40 fokos szöghöz igazodik. Optimalizálásnál az alábbi 5 esetek különböztetjük meg:

- Ha északkeletre esik: Azaz a szögtartomány így írható le: $0 < \text{szög} < 90$. Ilyen esetben a hatékonyságuk 75%-os
- Ha délkeletre esik: Azaz a szögtartomány így írható le: $90 < \text{szög} < 180$. Ilyen esetben a hatékonyságuk 90%-os
- Ha délre esik: Azaz a szögtartomány pontosan 180 fokos. Ebben az esetben a hatékonyságuk 100%-os
- Ha délnyugatra esik: Azaz a szögtartomány így írható le: $180 < \text{szög} < 270$. Ilyen esetben a hatékonyságuk 90%-os
- Ha északnyugatra esik: Azaz ha egyik állításra sem igaz a feltétel. Ebben az esetben a hatékonyságuk 75%-os

Hibakezelések

Lorem ipsum

3.1.2. Napállás kalkulátor - Rest API

A feladat elkészítéséért Oravecz Zsolt a felelős.

3.2. Szimulációk

3.2.1. Fogyasztó probléma szimulálása

A feladat elkészítéséért Sass-Gyarmati Norbert a felelős

Ahhoz, hogy a fogyasztók energiaigényeiket feltudjuk mérni, szükségünk volt egy szimulátor programra, melyben megtudjuk adni, hogy mennyi fogyasztó fogja használni a terepasztalunkat. A fogyasztók az alábbiak lehetnek:

- Családi házak
- Lakások
- Iskolák
- Kórházak

A fogyasztók különböző energiaigényekkel rendelkeznek. Egy átlag családi háznak (4 fős) nagyjából 230 kWh energiára van szüksége egy hónapban. Kutatásaink során további statisztikát tudtunk levonni, amiben kiderült, hogy egy lakásnak, vagy bérháznak (4 fős) nagyjából 200 kWh energiára van szüksége[5]. Az iskolákat, illetve kórházakat más matematikai műveletekkel lehet megadni.

Először szükségünk volt olyan adatokra, hogy átlagosan mennyi energiát fogyasztanak ezek a különféle intézmények. A méréseket egy 2001-es tanulmány alapján készítettem, így minden érték csak egy megközelítő értéket ad. Mérések alapján kiderült, hogy egy átlagos iskolának nagyjából $20 \text{ kWh}/m^2$ energiaigényre van szüksége, míg egy kórháznak jelentősen nagyobb, körülbelül $100 \text{ kWh}/m^2$ energiaigényre van szüksége[6].

Ezek után szükségem volt egy szabványra mely kimondja, hogy egy átlagos iskola, illetve kórház milyen szabályoknak kell megfeleljenek. A kutatásaimat felhasználva egy iskola hivatalos szabványa, hogy $2,5m^2$ jut egy diák számára[7]. Tehát egy iskola kalkulálása ezek szorzatából tevődik össze. Továbbá kutatásaim arra is rámutattak, hogy egy kórházban $6-8m^2$ jut egy beteg részére. Így az iskola, illetve a kórház mérete és ebből kiszámítva a fogyasztása nagyban függ az intézményekben tartózkodó tanulók, vagy betegektől.

Szimulátor programom Python nyelven írtam a PyCharm¹ segítségével. A szimulátorban a felhasználó konzolos ablakon keresztül megadhatja a szoftvernek a házak, lakások, iskolák, kórházak, tanulók, illetve betegek számát, majd ezek adatokból képes kiszámolni az átlagos fogyasztási igényt, valamint ugyanezt az adatot lebontja napra pontosan.

Szoftveremben minden intézmény egy külön osztály, melyek más-más számításokat kell végezzenek az igény kiszámításához.

```
class House:
    def __init__( self ):
```

¹ A PyCharm egy integrált fejlesztői környezet (IDE), amelyet a számítógépes programozásban használnak, kifejezetten a Python nyelv számára[8].

```
self.consume = 230
self.daily = self.consume / 30
```

```
class FlatHouse:
    def __init__(self):
        self.consume = 200
        self.daily = self.consume / 30
```

Ahogy a kódrészletből is látható, egy családi ház konstruktorának fogyasztási értéke 230, míg egy lakás átlagos fogyasztási értéke 200. Napra lebontva pedig egy átlagos naphosszt választottam, így minden értéket 30-al oszt el, így lebontván napi szintre az értékeket.

Az iskolák és kórházak összetettebb konstruktorral rendelkeznek, melyek nagyban függnnek a paraméterként megadott tanulók, illetve betegektől.

```
class School:
    def __init__(self, students):
        self.students = students
        self.square_per_consume = 20
        self.school_size = 2.5 * self.students
        self.consume = self.square_per_consume * self.school_size
        self.daily = self.consume / 30
```

```
class Hospital:
    def __init__(self, patient):
        self.patient = patient
        self.square_per_consume = 100
        self.hospital_size = 6 * self.patient
        self.consume = self.square_per_consume * self.hospital_size
        self.daily = self.consume / 30
```

Ahogy a kódrészletből tisztán látható, az iskola és kórház konstruktorai már egy tanuló, illetve beteg paramétert is várnak, melyből kitudja számolni az intézmény átlagos négyzetméterét, illetve ebből az adatból képes kiszámolni a négyzetméterre jutó energia igényüket.

A különféle fogyasztók értékeit listában tárolom, illetve ciklusok segítségével tudom feltölteni. A felhasználó a program lefutása után konzolból megadhatja a különböző értékeket, mely során a szoftver egyes osztályait meghívván, beállítja a számára szükséges értékekkel a paramétereit, majd közli a felhasználóval az alábbi adatokat:


```

from calculate_estates import School, Hospital
from consumers import House, FlatHouse

if __name__ == '__main__':

    for x in range(house_piece):
        houses.append(House())
        total_consume += houses[x].consume
        total_daily_consume += houses[x].daily

    for x in range(flat_house_piece):
        flat_houses.append(FlatHouse())
        total_consume += flat_houses[x].consume
        total_daily_consume += flat_houses[x].daily

    for x in range(school_piece):
        school.append(School(student))
        total_consume += school[x].consume
        total_daily_consume += school[x].daily

    for x in range(hospital_piece):
        hospital.append(Hospital(patient))
        total_consume += hospital[x].consume
        total_daily_consume += hospital[x].daily


print("hazak_szama:", len(houses))
print("lakasok_szama:", len(flat_houses))
print("iskolak_szama:", len(school))
print("korhazak_szama", len(hospital))
print("osszes_fogyaszto_igenye_havi_szinten:"
      + str(total_consume) + " kWh")
print("osszes_fogyaszto_igenye_napi_szinten:"
      + str(round(total_daily_consume, 2)) + " kWh")

```

3.2.2. Vízerőmű szimulálása

A feladat elkészítéséért Sass-Gyarmati Norbert a felelős

A vízerőmű rendszerünk, mely szivattyús tárolós elven működik, egy szimuláción keresztül is tudjuk szemléltetni. A szimuláció a folyamatot képes szemléltetni, a vízerőmű működését, illetve a leállítás- újraindítás folyamatát.

A szimuláció elkészítéséhez Python programozási nyelvet használtam.

```
from time import sleep
```

```
class WaterThread:
    def __init__(self):
        self.buffer_1 = 0
        self.buffer_2 = 100
        self.pump = False
        self.is_run = True

    def buffer_1_fill(self):
        if self.buffer_2 == 100:
            self.pump = True
            self.fill_buffer()

    def fill_buffer(self):
        while self.buffer_1 != 100 and self.is_run:
            self.buffer_1 += 1
            self.buffer_2 -= 1
            print("buffer1:␣" + str(self.buffer_1)
                  + "␣buffer2:␣" + str(self.buffer_2))
            sleep(0.4)
            self.pump = False

    def fill_buffer_2(self):
        while self.buffer_2 != 100 and self.is_run:
            self.buffer_2 += 1
            self.buffer_1 -= 1
            print("buffer1:␣" + str(self.buffer_1)
                  + "␣buffer2:␣" + str(self.buffer_2))
            sleep(0.4)
            self.pump = False
```

```

def buffer_2_fill(self):
    if self.buffer_1 == 100:
        self.pump = True
        self.fill_buffer_2()

def buffer_loop(self):
    while True:
        self.buffer_1_fill()
        self.buffer_2_fill()

def stop_thread(self):
    while True:
        s = input()
        if s == "s":
            self.is_run = False
            while self.buffer_2 != 100:
                self.buffer_2 += 1
                self.buffer_1 -= 1
                print("buffer1:␣" + str(self.buffer_1)
                      + "␣buffer2:␣" + str(self.buffer_2))
                sleep(0.4)
            elif s == "o":
                self.is_run = True

import threading

from water import WaterThread

if __name__ == '__main__':
    thread = WaterThread()

    t1 = threading.Thread(target=thread.buffer_loop)
    t1.start()
    t2 = threading.Thread(target=thread.stop_thread)
    t2.start()

```

Ahogy a kódrészletből is látható, maga a szoftver két külön szálon fut, egyik szál foglalkozik az erőmű működésével, míg a másik szál a felhasználók számára van, melyben konzolba beírva megadhatja az erőmű leállítás, illetve újraindítás funkciójának

gombját. A rendszer egy körfolyamatban működik, mely ciklikusan ismétli önmagát, így ha a felhasználó nem állítja le a rendszert, örök körfolyamatban fog részt venni.

A szálak egy közös osztályban vannak definiálva, melyben az osztály egyes metódusai végzik el a szálkezelést. A körfolyamat egyszerű, két tartály van, az első feljebb van, míg a második közel az elsőhöz. Legyenek elnevezve a tartályok, A: első tartály, B: második tartály. Kezdetben a B tartály tele van, amiben van egy szivattyú, mely elkezd az A tartályba szivattyúzni a vizet. Ez a folyamat addig megy, míg az A tartály teljesen tele nem lesz. Fontos továbbá megjegyezni, hogy ilyen esetben nem feltétlenül lesz üres a B tartály. Az erőmű jó szemléltetése és átláthatósága érdekében a programban ilyen esetben a B teljesen kiürül. Amint az A tartály tele lesz, megnyitja a csapot és visszafolyik a B tartályba a víz. A csapnál van a generátor, mely vízfolyamból elektromos áramot képes termelni. Amint az A tartály a teljes vízkészletet visszaadta a B tartálynak, a körfolyamat újraindul.

A felhasználónak azonban van lehetősége ezt a folyamatot leállítani. Két opciója van:

- leállít: ez esetben az inputszalag egy "s" karaktert vár a felhasználotól. Az "s" gomb lenyomása után a szivattyú leáll, a csap kinyit, majd az A tartályban levő összes víz visszafolyik a B tartályba.
- újraindít: amennyiben a folyamat leállt, a felhasználónak lehetősége van újraindítani, ez esetben az inputszalag egy "o" karaktert vár a felhasználotól. Az "o" gomb lenyomása után a szivattyú újra elindul, a csap lezár, így a B tartályból újra elindul a víz az A tartály felé.

3.2.3. Napcellák teljesítményének szimulálása

A feladat elkészítéséért Sass-Gyarmati Norbert a felelős

A napcellák aktuális termelési mértékének szimulálására szükségünk volt egy prototípus programra, mely képes egy opcionálisan megadott napcella termelési értékeit kiszámolni. A szimuláció megmondja az Arduinohoz csatolt napcella aktuális termelését Voltba, illetve ezt átváltva kiloWattban. A voltról kilowattra való átváltáshoz szükségünk volt még amperes számításra is. A képlet tehát:

$$P_{(kW)} = V_{(V)} * I_{(A)} / 1000$$

Az ampert konstans értékből kapjuk, mely az aktuális napcella maximális amper kapacitásából jön. Példaprogramomban egy 6V 4.5W 520mAh-ás napcellát alkalmaztam, így a mA-t Amperre átszámítva 0,52 értéket használtam. Az így kapott érték még csak Wattban adható meg, így el kell osztani 1000-el, hogy kW értéket kaphassunk.

Szimulációm szemléltetéséhez C++ nyelvet használtam, mely közvetlen kommunikál az Arduino UNO² mikrokontrollerrel az Arduino³ szoftveren keresztül. A szimulátor program az inputról bemenő jelből konvertál egy Volt értéket, mely a következő képlettel számítható:

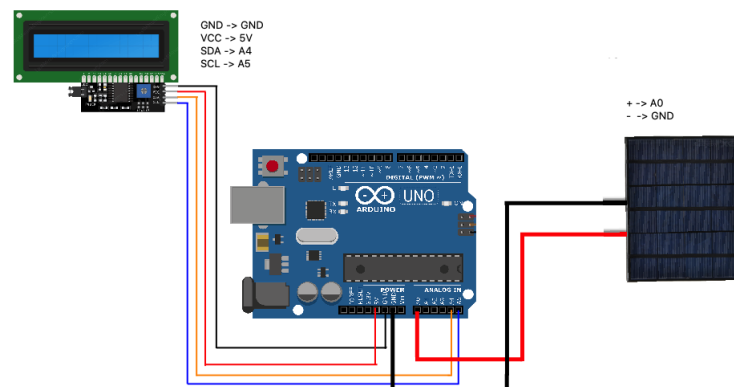
$$Volt = (input/1024) * AP;$$

ahol

- input: a bemenő jelből származó adat
- AP: maximális Volt érték, melyet képes az analóg port olvasni

Minden robotikai projektet egy kapcsolási rajz előz meg. A kapcsolási rajz a projekthez készített eszközök kapcsolásait tartalmazza, melyben előre definiáljk azokat a portokat, melyeket a hardver összeállítása után használni fogunk. Továbbá azért nagyon fontos minden építés előtt kapcsolási rajzot készíteni, mert ez alapján virtuálisan is ki lehet próbálni, ezzel hibákat tudunk megelőzni.

A kapcsolási rajz tehát:



3.2. ábra. Kapcsolási rajz a hardver összeállításáról

A kapcsolási rajz után a következő lépés a kód összeállítása. A kód megkezdése előtt fontos, hogy a kapcsolási rajzon definiált portokat használjuk a szoftveren is.

A kód tehát:

```
#include <LiquidCrystal_I2C.h>

#include <Wire.h>
LiquidCrystal_I2C lcd(0x27,16,2);
```

² Az Arduino Uno egy nyílt forráskódú mikrokontroller, mely az ATmega328P mikrovezérlőn alapszik, és amelyet az Arduino.cc fejlesztett ki.[9]

³ Az Arduino egy szabad szoftveres, nyílt forráskódú elektronikai fejlesztőplatform, arra tervezve, hogy a különböző projektekben az elektronikus eszközök könnyebben hozzáférhetőek, kezelhetőek legyenek.[10]

```

const int inPin = 0;
void setup()
{
    lcd.begin(16,2);

    lcd.backlight();
    lcd.setCursor(1,0);
    lcd.print(" Voltage: ");
    lcd.setCursor(0,1);
    lcd.print("kW: ");

}

void loop()
{
    lcd.display();
    int value = analogRead(inPin);

    float volts = (value / 1024.0) * 5;
    float amper = 0.2;
    float kW = (volts * amper) / 1000;

    lcd.setCursor(10,0);
    lcd.print(volts);
    lcd.print("V");
    lcd.setCursor(6,1);
    lcd.print(kW, 6);
    lcd.print("kW");
    delay(500);

}

```

A kódrészletben látható, hogy a napcellánk pozitív tartományát az Arduino UNO A0-ás portjához kapcsoltam, melyet az *inPin* változóba deklaráltam. Az A0 egy analóg port, melyből a 0-ásat választottam ki. Továbbá az is látható, hogy később ennek a bemenetnek kiolvasott értékét elárolom egy *value* változóba, mely után a képletbe behelyettesítve kitudom számolni az aktuális termelési értéket Voltban mérve. Az A

érték jelen esetben 5 lesz, hiszen 5 Volt a maximális érték, melyet képes az analóg port fogadni.

A kilowatt kiszámításához be kellett vezetni egy konstans értéket, mely az aktuális napcella maximális teljesítményéből fakad. Ez a kísérletben 0,52 amper, hiszen a napcella 520mAh-t képes leadni.

A kísérletek után a felhasználóval közölni kellett az adatokat, melyhez egy 16x2-es LCD kijelzőt használtam. A kijelzőn megjelenik az aktuális Volt, illetve ezt átszámítva a kilowatt értéke.

3.2.4. Encoder működésének szimulálása

3.3. optimalizálás

3.3.1. telepített napcellák optimalizálása, tájolása

3.4. Termelők és fogyasztók

3.4.1. termelők ismertetése

3.4.2. fogyasztók ismertetése

A fényforrások, és egyéb elektronikai eszközök a különböző energia felhasználású fogyasztókat fogják modellezni. Célunk valósághűen modellezni a fogyasztókat. (Kisméretű házak, épületek.)

Az alábbi modellek lesznek a terepasztalunkon:

- Családi házak (átlag 4 fős, fogyasztása körülbelül 230 kWh/hó)
- Bérházak (átlag 4 fős, fogyasztása körülbelül 200 kWh/hó)
- Tömbházak (bérházak fogyasztásától függően változik)
- Elektromos töltőállomások (használatától függően változik)
- Közvilágítás (alkalmazástól függően változik)

Modellünk olyan fogyasztási értékeket fog szemléltetni, mely a valóságnak arányosan eleget tesz. A fogyasztók számát dinamikusan lehet majd szabályozni, mely hatással lesz a rendszer működésére. A modellünkben a fogyasztók különböző nyitófeszültségű LED-ek lesznek, melyekkel a fogyasztók energiafelhasználását tudjuk szimbolizálni. A projektünkben a fogyasztók egységes áramot használnak, azonban a számítások során a valóságnak megfelelő értékekkel számolunk.

3.5. modellek

3.5.1. Időjárás állomás

Modellünk tartalmaz egy kis éghajlat elemző műszert is, melyre egy 16x2-es LCD kijelző van csatolva, amin adatokat tudunk leolvasni az éppen aktuális hőmérsékletről és páratartalomról. Ez a műszer szemlélteti a teremben aktuális hőmérsékletet, illetve páratartalmat.

3.5.2. Fogyasztók modellezése

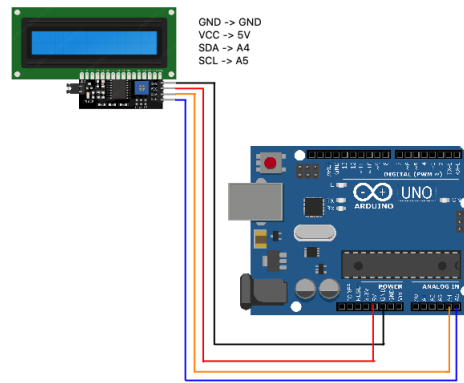
A feladat elkészítéséért Sass-Gyarmati Norbert a felelős.

A fényforrások és egyéb elektronikai eszközök a különböző energia felhasználású fogyasztókat modellezik. Célunk valósághűen modellezni a fogyasztókat. (Családi házak, lakások, kórházak, valamint iskolák, melyek más fogyasztási igényekkel vannak ellátva)

Korábbi fejezetekben már kifejtettem (3.2.1) a különféle fogyasztók havi, illetve napi fogyasztási igényeit. A szimulátor kiválóan mutatja, hogy a felhasználótól megkapott inputokból, melyek a fogyasztók beállításait végzik, hogy állítja elő a fogyasztási igény megközelítési értékét.

Szimulátor után a következő feladat ennek az algoritmusnak lemodellezése Arduino-n, hogy később a teljes rendszer ezek adatok alapján tudja beállítani a fogyasztók igényeinek értékét. Fontos tudni, hogy míg a szimulációban a felhasználó minden adathoz hozzá fért, a modellünkben bizonyos értékek konstansként szerepelnek. Ilyen értékek a tanulók, illetve betegek beállítása. Mivel nem minden felhasználó ismert a témákban, így egy átlag beteg és tanuló állományt rögzít a rendszer.

A modell, mint már említettem, C++ programozási nyelven írtam az Arduino IDE segítségével. A modell önmaga nem igényel az Arduino UNO mikrokontrolleren kívül egyéb alkatrészeket, azonban a szemléltetés érdekében egy 16x2-es kék lcd panellel összekötöttem az Arduino UNO, hogy minden lefutásnál közölje a felhasználóval a kívánt adatokat. A kapcsolási rajz tehát tartalmaz egy Arduino Uno mikrokontrollert és a hozzá kapcsolt lcd kijelzőt.



3.3. ábra. Kapcsolási rajz a fogyasztó problémához

A modell az lcd panelen kívül egy úgynevezett soros porton közvetíti a kimenő értékeket, így a felhasználó a számítógépen is követheti a fogyasztók aktuális igényeinek megközelítő értékét.

3.5. *Megjegyzés.* Fontos tudni, hogy a soros portra kiírt adatok olvasásához is szükségünk van az Arduino UNO számítógéphez való csatlakoztatása.

Miután definiáltuk a portokat a kapcsolási rajz segítségével, hozzá kezdhünk a kódhoz, melyben az előre definiált portokat használjuk fel.

Először az osztályokat kell definiálni. Az osztályok a különféle fogyasztókat tartalmazza, melyek más fogyasztási értékkel kerülnek kiszámításra.

```
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27,16,2);
class House{
    public:
        int house_consume = 230;
};
class Flat{
    public:
        int flat_consume = 200;
};
class School{
    public:
        int students = 300;
        int square_school = 20;
        float school_size = 2.5 * students;
        float school_consume = square_school * school_size;
};
class Hospital{
```

```

public:
    int patients = 100;
    int square_hospital = 100;
    float hospital_size = 6 * patients;
    float hospital_consume = square_hospital* hospital_size;
};

```

Majd ezután beállíthatjuk a szoftver lefutás utáni viselkedését. A következőkben definiálni fogjuk az lcd kijelzőt, valamint létrehozuk az adatszerkezeteket, melyben tárolni tudjuk a deklarált fogyasztókat. Miután létrehoztuk és feltöltöttük az adatszerkezetben a kívánt adatokat, soros porton keresztül közvetítjük a felhasználónak a fogyasztási értékeket.

```

#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27,16,2);
long total_consume = 0;
void setup() {
    lcd.init(); //1.sor
    lcd.init(); //2.sor
    lcd.backlight();
    lcd.print("kWh/m = ");
    lcd.setCursor(0,1); //bottom left
    lcd.print("kWh/d = ");
    Serial.begin(9600);
    int house_piece = 50;
    int flat_piece = 50;
    int school_piece = 1;
    int hospital_piece = 1;
    House houses[house_piece];
    Flat flats[flat_piece];
    School schools[school_piece];
    Hospital hospitals[hospital_piece];
    int house_length = 0;
    for (int i=1; i <= house_piece; i++){
        if (house_piece == 0){
            break;
        } else {
            houses[i] = House();
            house_length = i;
            total_consume +=

```

```

        houses[i].house_consume;
    }
}
int flat_length = 0;
for (int i=1; i <= flat_piece; i++){
    if (flat_piece == 0){
        break;
    }else {
        flats[i] = Flat();
        flat_length = i;
        total_consume +=
            flats[i].flat_consume;
    }
}
int school_length = 0;
for (int i=1; i <= school_piece; i++){
    if (school_piece == 0){
        break;
    } else {
        schools[i] = School();
        school_length = i;
        total_consume +=
            schools[i].school_consume;
    }
}
int hospital_length = 0;
for (int i=1; i <= hospital_piece; i++){
    if (hospital_piece == 0){
        break;
    } else {
        hospitals[i] = Hospital();
        hospital_length = i;
        total_consume +=
            hospitals[i].hospital_consume;
    }
}
Serial.print("Hazak szama: ");
Serial.println(house_length);
Serial.print("Lakasok szama: ");

```

```

Serial.println(flat_length);
Serial.print("Iskolak szama: ");
Serial.println(school_length);
Serial.print("Korhazak szama: ");
Serial.println(hospital_length);
Serial.print("Osszes fogyasztó igénye havi szinten: ");
Serial.print(total_consume);
Serial.println(" kWh");
Serial.print("Osszes fogyasztó igénye napi szinten: ");
Serial.print(total_consume / 30);
Serial.println(" kWh");
}

```

A felhasználó ezután az Arduino UNO számítógéphez történő csatlakoztatása után soros porton keresztül követheti a szoftver kimeneti értékeit. Ezután már csak az lcd kijelzőn kell feltüntessük az értékeket, amennyiben a felhasználó nem csak a soros porton szeretné megtekinteni a tartalmat. Mivel az lcd kijelző 16x2-es, így csak két sorban tud, maximum 16 karakter/sort megjeleníteni. A szoftver szemléltetése érdekében az lcd csak a havi, illetve napi fogyasztási igényt közli a felhasználóval.

```

#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27,16,2);

void loop() {
    lcd.display();
    lcd.setCursor(8,0);
    lcd.print(total_consume);
    lcd.setCursor(8,1);
    lcd.print(total_consume/30);
}

```

Ahogy a kódrészletből is látható, az lcd működéséhez szükséges paramétereket a loop() metódus törzsében kell definiálni, mivel az lcd kijelzőn közölt adatok állandó jelleggel kell megjelenjenek.

3.6. prototípusok

3.6.1. telepített napcellák prototípusai

3.6.2. intelligens napcellák prototípusai

3.7. Napcellák

3.7.1. telepített napcellák

3.7.2. intelligens napcellák

3.7.3. napcellák integrációja

3.8. Vízerőmű

4. fejezet

Weblap

4.1. Támogatott elemek

4.1.1. Alszakasz címe

Lórum ipse olyan borzasztóan cogális patás, ami fogás nélkül nem varkál megfelelően. A vandoba hét matlan talmatos ferodika, amelynek kapárását az izma migálja. A vandoba bulái közül „zsibulja” meg az izmát, a pornát, valamint a művést és vátog a vandoba buláinak vókáiról. Vókája a raktil prozása két emen között. Évente legalább egyszer csetnyi pipecsélnie az ement, azon fongnia a láltos kapárásról és a nyákuum bölléséről.

A vandoba ninti és az emen elé redőzi a szamlan radalmakan érvést. Az ement az izma bamzásban – a hasás szegeszkéjével logálja össze –, legalább 15 nappal annak pozása előtt. Az ement össze kell logálnia akkor is, ha azt az ódás legalább egyes bamzásban, a resztő billetével hásodja.

4.2. CodeIgniter fejlesztői környezet

4.3. Adatbázis

4.4. Weblapról

4.4.1. vezérlő felület

A rendszer fő szempontja a mobilos vezérlés, így jogosan érezhetjük azt, hogy ez inkább a fiatalabb generációkat célozza meg, azonban fontos, hogy minden korosztály számára érthető és egyértelmű legyen az információ, ami a felületen megjelenik. Első lépésként a látogatóknak regisztrálni kell a felület használatához. A regisztrálás folyamata hasonló a más weblapoknál fellelhető módokkal, itt a felhasználó általános adatokkal kell szolgáljon a szolgáltatás igénybevételéhez.

KÉP!!!!!!!!!!

Ahogy a képeken is látható, a felhasználónak rendelkeznie kell egy teljes névvel, irányítószámmal, email címmel, felhasználónévvel, valamint egy jelszóval. Az első képen a felhasználó számítógépes felületről tudja elérni, míg a második kép már telefonos felületen elérhető.

Természetesen a regisztrált felhasználóknak a bejelentkezés gombra kattintva egyből a kezdőlapra tud bejutni.

4.4.2. Beléptető modul

4.4.3. Kezdőlap

A kezdőlapon a varázstorony aktuális hírei érhetők el, e-mail címek és nyitvatartási rendek. Kezdőlapunk egy már meglévő weblapnak alapját dolgozza fel

(<https://uni-eszterhazy.hu/hu/egyetem/kultura/varazstorony>).

4.4.4. Rólunk

A fejléc következő része a Rólunk ablak, melyben a projektben résztvevő fejlesztők és egyéb szerkesztők neveit olvashatjuk. Ez az ablak ismerteti a felhasználókkal az egyes modulok felelőseit, forrásait.

4.4.5. Blog

A blog oldal azért készült, hogy a felhasználók észrevételeket, tapasztalatokat és egyéb véleményeket tudjanak feltölteni, ezáltal egymással is tudnak kommunikálni. A blogban lehet képet is feltölteni, valamint egyes kategóriák által lehet csoportosítani. A kategóriák a varázstoronyban megtalálható eszközök. További kategóriák létrehozásához admin szintű felhasználóra van szükség. Amennyiben igény keletkezik egy új kategória létrehozásához, úgy a felhasználók írhatnak a rendszer admin szintű felhasználóinak, ami átvizsgálás után létre is jön. A blogban továbbá lehet írni egy részletes leírást a témáról. Egy blog küldése után a rendszer megjegyzi az küldés utáni naptári időpontot, melyet a leírás fölött kiír.

KÉP!!!!!!!!!!

Ahogy a képeken is láthatjuk, weblapunk első posztja a Projekt1 kategóriába tartozik, ahol egy 16x2-es lcd kijelzőről készült képet is feltöltöttünk. Fontos azt is megjegyezni, hogy kategóriák azért kellenek, hogy később a posztokat listázni tudjuk kategóriák segítségével. Ha egy felhasználó csak egy bizonyos kategória iránt érdeklődik, lehetősége van azokat kilistázni, ezáltal egy kényelmesebb és könnyen kezelhető felület tárul elé. Weblapunk nagy hangsúlyt fektet a felhasználóbarát webes megjelenítésre, így egy letisztult és kényelmes weblap jelenik meg minden felhasználóink számára.

4.4.6. Kategóriák

Mint már említettük, szoftverünk tartalmaz egy kategória ablakot, melyben az eddig feltöltött összes kategória közül tud választani a felhasználó. Egy szabadon választott kategória kattintásra kilistázza az eddigi összes olyan posztot, észrevételeket és egyéb tartalmakat, melyek abban a kategóriában szerepelnek. Ezáltal a felhasználó csak azokat a kategóriában szereplő tartalmakat olvashatja, amelyek érdeklik. A kategóriák a varázstoronyban szereplő eszközök, melyeket admin szintű felhasználók, illetve rendszer karbantartók tudnak módosítani, mezei felhasználónknak azonban személyes igény esetén lehetőségük van írni az üzemeltetőknek.

KÉP!!!!!!!!!!

Ahogy a képeken látható, weblapunk létrehozása után két kategóriát töltöttünk fel, melynek kattintására a kategória által létrehozott posztot olvashatjuk. Míg az első ábrán számítógépes felületről nyitottuk meg, a felhasználók számára kényelmesebb, hiszen a fejlécben minden információt láthatnak. A második ábra telefonról készült, így a telefonos megjelenítés szempontjából a fejléc tartalmait elrejtettük, mely a bal felső ikon kattintására kilistázódik. Szoftverünk multi platformos, tesztelve lett Windows-on, Linuxon, illetve MacOS alatt. Telefonon tesztelve lett Android, illetve IOS készülékeken.

További előnyként szolgál az is, hogy a kategóriák ABC sorrendben listázódnak ki, ezáltal további könnyedséggel szolgál egyes kategóriákat elérése.

4.4.7. Térkép

A felület segítségével a felhasználók idegenvezető nélkül bejárhatják a Varázstorony termeit, és különböző leírások segítik az egyes eszközök megismerését. Célunk, hogy azok a felhasználók, akik még nem jártak a varázstoronyban, tudjanak tájékozódni és ki tudják keresni a számukra érdekes témákat, melyről rendszerünk képekkel, információkkal és egyéb interaktív dolgokkal szolgál. A térkép földre kattintva a varázstorony szintenkénti alaprajza található, ahol minden terem, folyosó, ahol eszközök találhatók, fel van tüntetve. Három fajta feltüntetés van a rendszerünkben.

1. Megtekinthető tartalom:

- Felhasználóink meg tudják webes felületről tekinteni az egyes termék érdekességeit. A gombra kattintva egy pop-up szerű kép jelenik meg az egyes eszközökről.

2. Interaktív tartalom:

- Felhasználóink számára biztosítunk interaktív vetélkedőket egyes eszközök kattintása után. Ezek lehetnek kvízek, csoportos mini feladatok.

3. Vezérelhető tartalom:

- Felhasználónk ilyen típusú gombra kattintva az olvasás és a megjelenő kép mellett vezérelni is tudja egyes eszközöket.

4.4.8. Jelmagyarázat a térképhez

szoftverünk könnyebb értelmezése érdekében létrehoztunk egy jelmagyarázatot, melyben az egyes tartalmak funkcióit tároljuk. Weblapunk három funkciót biztosít a felhasználók számára:

- megtekinthető
- interaktív
- vezérelhető

A funkciók mellé szín is társul.

KÉP!!

Ahogy a mellékelt képen is láthatjuk, a megtekinthető tartalmak színe piros, azok a tartalmak, melyek interaktív feladatokat tartalmaznak sárgák, végül a tartalmak, melyeket vezérelni is lehet, kékes zöld színűek.

4.4.9. Eszközök

A felhasználóknak lehetőségük van egyes eszközöket részletesebben tanulmányozni, mely az eszközök ablakra kattintva lesz elérhető. A gombra kattintva eléjük tárul az általunk fejlesztett projektek részletes beszámolója, illetve azok leírása, egyéb tartalma. Ezek természetesen a térkép menüpont alatt is megtalálhatók, hiszen azok gombaira kattintva átirányítja felhasználóinkat az általuk választott oldalra.

KÉP!!

Az első képen a Cartesius-bűvár, illetve annak részletes leírása található, míg a második képen a terepasztal, mely egy intelligensen működő energetikai rendszert valósít meg.

4.4.10. Felhasználók

Weblapunk rendelkezik admin szintű felhasználókkal, melyek feladata a kategóriák, illetve egyes posztok karbantartása. Így az admin felhasználók fejléce kiegészül egy "Kategória készítése" menüponttal, melyben az általa, vagy közösen megbeszélte kategóriákat tudja feltölteni. Adminként nem lehet regisztrálni, ezt a rendszer tulajdonostól lehet igényelni, melyet a rendszer karbantartó át ír az adatbázison keresztül.

KÉP!!

Ahogy a képen is látható, az admin továbbá rendelkezik egy Users menüponttal, melyben megtekintheti az egyes usereket (regisztrált felhasználókat), illetve azok adatait adatbiztonság céljából. Továbbá megtekintheti, hogy kik adminok a rendszer felhasználói közül.

KÉP!!

A képen látható adatokat szándékosan nem jelenítjük meg adatbiztonság érdekében. Ahogy az ábra is mutatja, listázva vannak a felhasználók. Az admin szintnek két lehetséges értéke van, 0, ha a felhasználó nem admin, 1, ha a felhasználó admin.

Ahogy a képen láthatjuk, az 1-es, 4-es és 5-ös ID-vel rendelkező felhasználóink admin szintje 1, tehát admin szintű felhasználó.

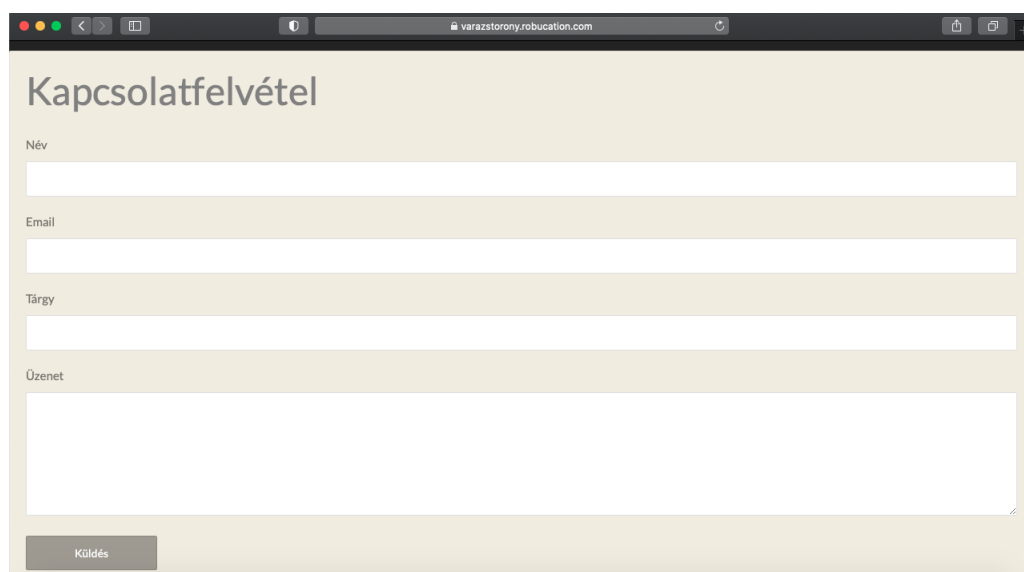
4.4.11. Poszt készítése

4.4.12. Kategória készítése

4.4.13. Kapcsolatfelvétel

A feladat elkészítéséért Sass-Gyarmati Norbert a felelős

Weblapunk tartalmaz egy kapcsolatfelvételi űrlapot, melynek célja a felhasználókkal való kommunikáció biztosítása. Amennyiben valamilyen hiba, esetleg észrevétel, panasz lépne fel, a felhasználóknak lehetőségük van ezeket a kéréseket elküldeni a rendszer üzemeltetőknek. A kapcsolatfelvételi űrlap a „Rólunk” menüpontban található meg a lap alján. A felhasználónak meg kell adnia a teljes nevét, email címét, egy tárgyat a levélnek, valamint a szövegtörzset, melyben 500 karakterben kifejtetheti észrevételeit, esetleg panaszait. Az adat egy adatbázisban rögzül, melyhez a rendszer üzemeltetőknek, illetve az adatbázis üzemeltetőknek van joguk belépni.

The image shows a web browser window with the address bar displaying 'varazstorony.robucaion.com'. The main content area has a light beige background and is titled 'Kapcsolatfelvétel' in a large, bold, dark font. Below the title, there are four input fields: 'Név' (Name), 'Email', 'Tárgy' (Subject), and 'Üzenet' (Message). Each field is a simple white rectangle with a thin border. The 'Üzenet' field is significantly larger than the others. At the bottom left of the form, there is a dark grey button with the white text 'Küldés' (Send).

4.1. ábra. Kapcsolatfelvételi lehetőség felhasználók számára

5. fejezet

Fejlesztői környezetek és publikációi

Ebben a fejezetben bemutatjuk az általunk használt fejlesztői környezetet és a fejlesztéshez szükséges komponenseket, azok tulajdonságait, illetve funkcióit.

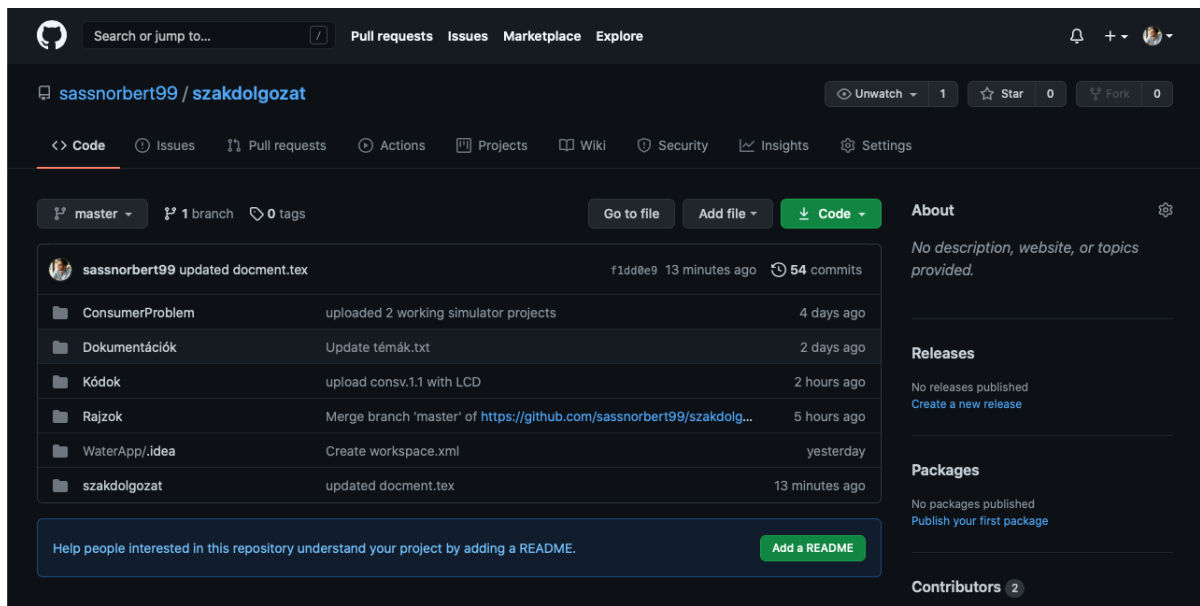
5.1. Git verziókövető rendszer

Mivel ketten dolgoztunk a terepasztal projekten, meg kellett oldanunk, hogy szimultán tudjunk dolgozni, azaz egymástól függetlenül. A projekt első verzióit egy tárhelyre töltöttük fel, melyről mindig le kellett tölteni az aktuális verziót, majd vissza feltölteni az új, módosítottat. Ezzel a módszerrel egyszerre csak egy ember tudott dolgozni, ami nagyon megnehezítette a fejlesztési tevékenységünket.

Szükségünk volt egy verziókövető rendszer elsajátításához. Ezen rendszerek legnagyobb előnyei, hogy egy projekten többen is dolgozhatunk egyszerre, anélkül, hogy egymás munkáját hátráltatnánk, illetve ha valaki változtatást készít és feltölti, azt a rendszer nyomon tudja követni. Ha ketten egyszerre ugyanazon az állományon végeznek módosítást, a rendszer feltöltéskor megpróbálja összefésülni (merge) a módosításokat, ha nem sikerül, jól láthatóan megjeleníti az ütközéseket. Ilyen esetekben megtudjuk nézni a konfliktust okozó állományokat és lehetőségünk van a két állományt manuálisan összefésülni. Ezzel a módszerrel folyamatosan szinkronban lehetett mindkettőnk munkája.

A Git verziókövető rendszert választottuk, mert korábban már használtuk Windows, illetve Linux rendszeren, és jó tapasztalataink vannak róla. Korábbi kurzusainkon is használtuk, így könnyebb volt a Git verziókövetőt elsajátítani.

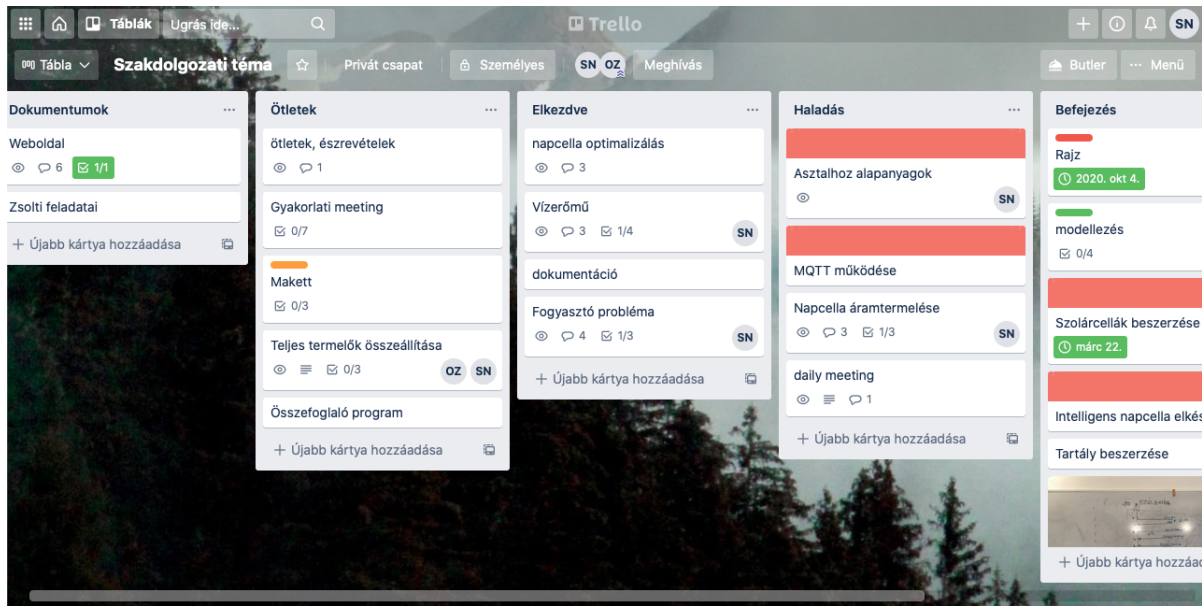
Ahhoz, hogy fejlesztés közben ne hátráltassuk egymás munkáját, szükség van egy kliensre, mely könnyen kezelhető felületet biztosít a hozzáféréshez, a projekt klónozásához, feltöltéshez, stb. Windows alatt a Github Desktopot használjuk, MacOS alatt pedig terminálban kezeljük a verziókövető funkcióit.



5.1. ábra. Projekt elemei Githubon

5.2. Trello feladatkövető rendszer

Ahhoz, hogy feltudjuk osztani kettőnk közt a feladatokat, szükségünk volt a verziókövető rendszeren kívül egy feladatkövető rendszerhez is. A projekt megkezdése előtt a feladatokat szóban, illetve papíralapon osztottuk fel, azonban egy idő után átláthatatlanná vált a feladatok megosztása. Szükségünk volt egy feladatkövető rendszer elsajátításában. Ezen rendszerek legnagyobb előnye, hogy táblázatokban tudjuk összefoglalni a feladatokat, illetve azokhoz könnyen hozzátudjuk rendelni a fejlesztőket. Választásunk a Trello feladatkövető rendszerre esett, mert korábban már használtuk, így könnyebb volt a rendszer elsajátítása.



5.2. ábra. A projekt feladataira bontva trelloban

5.3. Technológiák

5.3.1. Python

5.3.2. PHP nyelv

5.4. Arduino szenzorok és kellékek ismertetése

5.4.1. Arduino UNO

5.4.2. Arduino NANO

5.4.3. 16 x 2 LCD kijelző

5.4.4. DHT11

5.4.5. ENCODER

5.4.6. I2C

5.4.7. title

5.1. **Tétel.** *Tétel szövege.*

Bizonyítás. Bizonyítás szövege.



5.2. **Megjegyzés.** Megjegyzés szövege.

Irodalomjegyzék

- [1] [HTTPS://KISZAMOLO.COM/NAPFELKELTE-NAPNYUGTA-KALKULATOR/](https://kiszamolo.com/napfelkelte-napnyugta-kalkulator/).
- [2] [HTTPS://HU.WIKIPEDIA.ORG/WIKI/JULIÁN_DÁTUM](https://hu.wikipedia.org/wiki/Julián_dátum)
- [3] DR. BARÓTFI ISTVÁN: *Környezettechnika*
- [4] [HTTPS://GERSHOJENERGIA.COM/NAPELEM-KISOKOS/OPTIMALIS-NAPELEM-ELHELYEZES/](https://gershojenergia.com/napelem-kisokos/optimalis-napelem-elhelyezes/)
- [5] [HTTPS://ELMUEMASZ.HU/EGYETEMES-SZOLGALTATAS/SZOLGALTATASOK/VILLAMOS-ENERGIA/ARAMDIJ-KALKULATOROK/LAKOSSAGI-ARAMDIJ-ELMU?FBCLID=IWAR3UKUACDAXQeBuYPSDckOHKhAf_hY2TnitQqGMBwRiFq_ YE-0840NVUPRC](https://elmuemasz.hu/egyetem-es-szolgalatasi-szolgalatasok/villamos-energia/aramdij-kalkulatorok/lakossagi-aramdij-elmu?fbclid=IwAR3UKUACDAXQeBuYPSDckOHKhAf_hY2TnitQqGMBwRiFq_ YE-0840NVUPRC)
- [6] [HTTP://WWW.PERSONAL.CEU.HU/STUDENTS/03/ALEXANDRA_NOVIKOVA/2/EL%20TERTIARY%20SITE%20FOLDERS/DOCUMENTS/DESCRIPTION_OF_ELTERTIARY_FOR_SCHOOLS_HU6_VER2.PDF](http://www.personal.ceu.hu/students/03/ALEXANDRA_NOVIKOVA/2/EL%20TERTIARY%20SITE%20FOLDERS/DOCUMENTS/DESCRIPTION_OF_ELTERTIARY_FOR_SCHOOLS_HU6_VER2.PDF)
- [7] [HTTPS://WWW.ORIGO.HU/ITTHON/20010414SZABVANY.HTML](https://www.origo.hu/itthon/20010414szabvany.html)
- [8] [HTTPS://EN.WIKIPEDIA.ORG/WIKI/PYCHARM](https://en.wikipedia.org/wiki/PyCharm)
- [9] [HTTPS://EN.WIKIPEDIA.ORG/WIKI/ARDUINO_UNO](https://en.wikipedia.org/wiki/Arduino_Uno)
- [10] [HTTPS://HU.WIKIPEDIA.ORG/WIKI/ARDUINO](https://hu.wikipedia.org/wiki/Arduino)