



**Budapesti Műszaki és Gazdaságtudományi Egyetem**  
Villamosmérnöki és Informatikai Kar  
Automatizálási és Alkalmazott Informatikai Tanszék

Markovics Gergely, Nagy Viktor  
**ÖNÁLLÓ LABORATÓRIUM**

EKLER PÉTER

BUDAPEST, 2021

# 1 Bevezetés

## 1.1 Téma választás és aktualitása

Az általunk választott téma, egy olyan szoftver, melyet a plázák/bevásárló központok kamerái segítségével tudunk használni. A program képes a kamera képéből kiolvasni az előtte elhaladt személyek életkorát és nemét. Ezeket az adatokat el tudja tárolni az adott kamerára vonatkozóan, majd felhasználva ezen adatokat képes megjósolni egy általunk megadott jövőbeli időpontra, hogy mekkora valószínűséggel fog a pláza adott területén tartózkodni a kívánt kor- és nemcsoport. Ezen tudást felhasználva sokkal effektívebben lehet „targetált” hirdetések elhelyezni a központ különböző területein.

Az elmúlt évben ez fontossá is vált, ugyanis a központok látogatottsága jelentősen visszaesett a pandémiás időszak alatt, így megfontoltabban kell bánnia a fenntartónak az összeggel, amit például reklámokra lehet szánni. Az elkészített program ezen probléma feloldására és megkönnyítésére készült el.

## 1.2 Technológia megválasztása

A megoldás több technológia segítségével is megvalósítható. (Java, Python) Azonban, mi a Python mellett döntöttünk, ugyanis a külső könyvtárak alkalmazása egyszerűbb ezen a nyelven. Emellett azok a Machine Learningen alapuló könyvtárak, amelyekre szükségünk volt a feladat megoldása során (OpenCV, Facebook Prophet) jóval támogatottabb Pythonban, mint Javában.

A frontend pedig egy webalkalmazás. Tehát a HTML, CSS, Javascript és JQuery technológiákat használjuk fel. A webserver futtatásához pedig a Flask nevű web frameworkot használtuk.

## 1.3 Dolgozat szerkezete

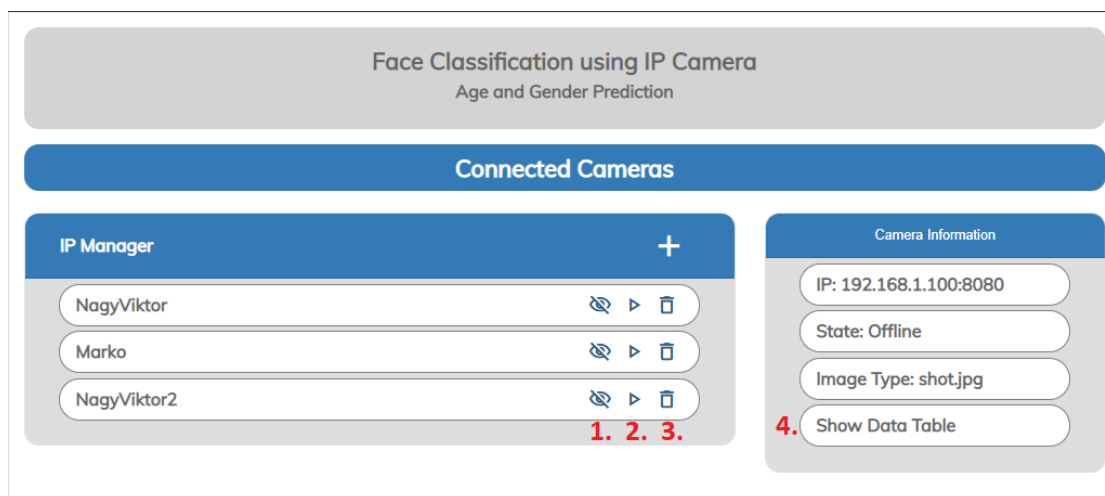
A program 3 nagyobb részből áll össze. Van egy backend, egy frontend és egy adatbázis. Úgy találtuk a legcélravezetőbbnek bemutatni a program felépítését, ha ezen 3 nagy csoportot bontjuk témakörökké. A munkamenet folyamatát is így tervezzük felosztani. Nagy Viktor fogja írni a backend részét, míg Markovics Gergely a frontend részét, azonban ellenőrizzük folyamatosan a másik munkáját. A dolgozat egyéb részeit közösen végezzük.

## 2 Program felépítése

### 2.1 Program használata és leírása

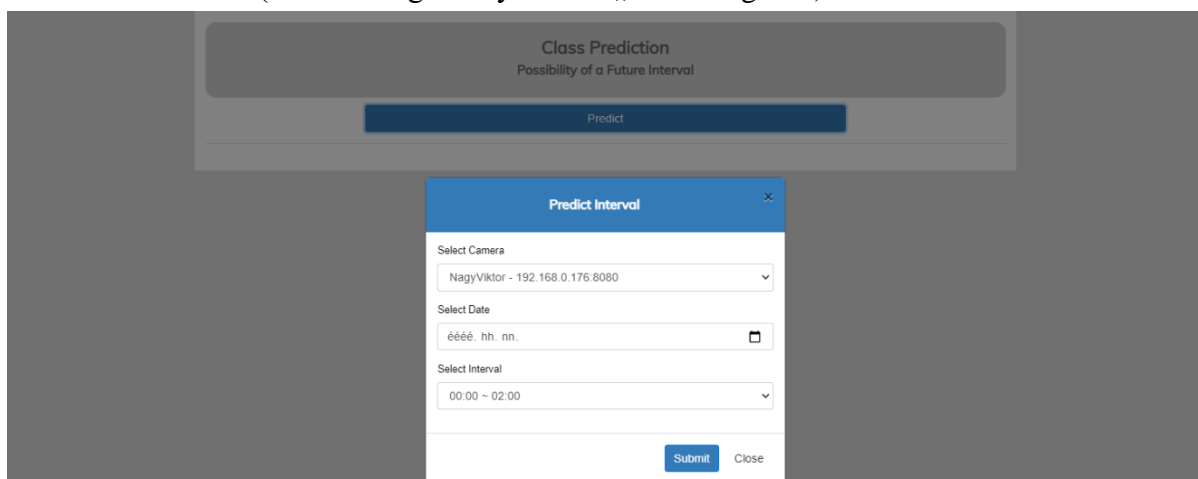
A weboldalnak két fő felülete létezik, van egy „Camera Manager oldal, melyen az általunk a rendszerhez adott kamerákat tudjuk kezelni (indít, leállít, vizsgál stb...). A 2. oldal pedig egy „Prediction” oldal, ahol ki tudunk választani egy időpontot és egy kamerát, majd ezen adatok segítségével elkészíti a predikciót.

#### 1. felület kinézete:



- 1. input:** Ezzel a gombbal meg lehet tekinteni, az adott kamera képét. Csak akkor megnyomható, ha a kamera elindított állapotban van.
- 2. input:** Ennek a gombnak két állapota van (elindított és leállított), a képen látott állapotban a kamera nincs elindítva.
- 3. input:** A kamerát kitörlésére szolgáló gomb.
- 4. input:** Meg lehet tekinteni a kamera ez idáig összegyűjtött adatait.

#### 2. felület kinézete: (Miután meg lett nyomva a „Predict” gomb)



**Select Camera input:** Az első oldalon hozzáadott kamerák közül lehet választani.

**Select Date input:** Egy jövőbeli időpontot lehet kiválasztani. (év, hónap, nap)

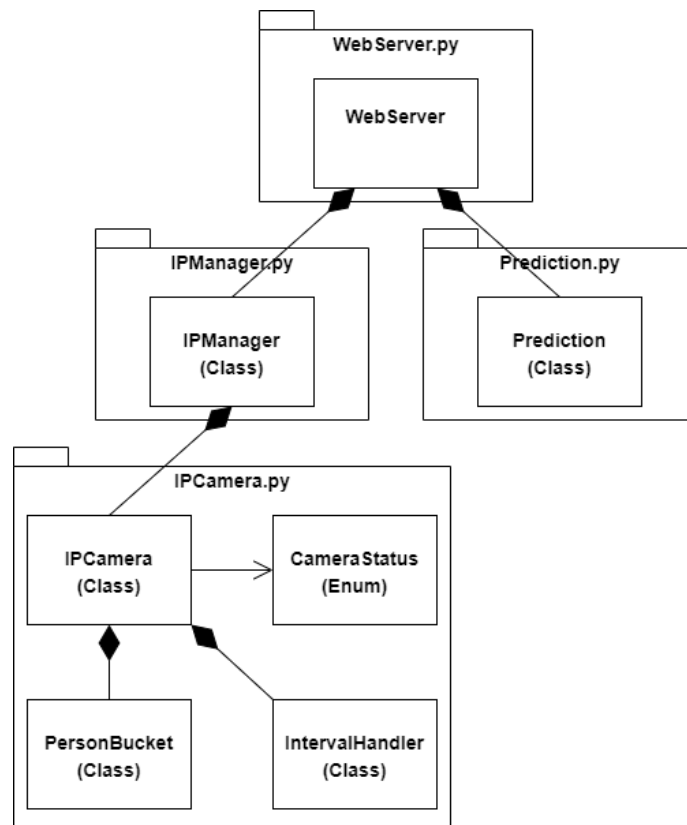
**Select Interval input:** A 2 órás intervallumok közül lehet pontosítani az időpontot.

**Submit input:** Véglegesíti a beadott adatokat, majd kiszámolja az információkat amire szükségünk van.

Röviden így épül fel a program, de a későbbiekben pontosítva lesz, hogy hogyan működnek és mit csinálnak az adott elemek.

## 2.2 BackEnd

### 2.2.1 Felépítése röviden



A program, mint ahogy az az ábrán is látszódik 4 darab Python fájlra osztható fel.

Az IPCamera.py egy kamerát jelképez és alap feladatai közé tartozik a kamerához kapcsolódó adatok tárolása, indítás/leállítás, kameraadatok kiírása adatbázisba és egyéb dolgok, amik részletezve lesznek a későbbiekben.

A fájlban belül több osztály is helyet foglal, ezek a kamera kisegítő osztályai, van egy, ami a kamera státuszát jelenti (CameraStatus), van egy, ami a kamera által megfigyelt személy adataira vonatkozik (PersonBucket) és van egy IntervalHandler, ami az időintervallumokkal kapcsolatos műveleteket és adattárolást segíti elő.

Az IPManager.py egy olyan „réteg” melynek az a feladata, hogy a kamerákat listában tárolja, ezen a listán műveleteket lehessen végrehajtani (hozzáadás, kitörlés) és a kamerákon van műveleteket központosítsa.

A Prediction.py a Prediction osztályt foglalja magába, ez az osztály végzi a meglévő adatok alapján a predickió műveletét.

Végezetül pedig maga a Webserver foglal helyet az ábra tetején, ugyanis ő kapja az utasításokat a honlapról, majd ezeket routekon keresztül továbbítja a kívánt osztálynak. A fontosabb routekról bővebben lesz szó a 2.2.5-ös pontban.

## 2.2.2 MI technológiák leírása

A szoftver alapvetően két fontosabb mesterséges intelligenciát használ. A program ugye kamerák képét dolgozza fel és az azon lévő emberek arcát kell, hogy tudja felismerni, így elengedhetetlen egy gépi látást segítő könyvtár.

Ehhez mi az OpenCV nevű ingyenes és nyílt forráskódú könyvtárat használtuk. A programunkhoz tudnia kellett az arcfelismerést, és ehhez az OpenCV támogatást ad, hogy egy úgynevezett CascadeClassifier metódus segítségével betanítsuk a mesterséges intelligenciát arra, hogy felismerje az arcokat. Ehhez neki szüksége van egy xml re, így az opencv github oldaláról felhasználtuk a haarcascade\_frontalface\_default.xml-t. (az arc felismerésen túl még sok fajta xml létezik, pl: test, szem, mosoly és egyebek)

Azonban nekünk nem volt elég, hogy felismerje az arcot, tudnunk kellett az arc alapján az adott ember korát és nemét, ugyanis ezekről gyűjtjük az adatok melyet később felhasználunk. Ezúttal más módon készítettük el felismerést. Itt az OpenCV nek az úgynevezett „caffemodel” -ekből való tanulási módszerét használtuk. A folyamat ugyanaz, mint az arcfelismerésnél, csak itt máshonnan tanítja be magát. (age.caffemodel és gender.caffemodel)

```
self.age_model = cv2.dnn.readNetFromCaffe("Backend/Models/age.prototxt", "Backend/Models/age.caffemodel")
self.gender_model = cv2.dnn.readNetFromCaffe("Backend/Models/gender.prototxt",
                                              "Backend/Models/gender.caffemodel")
self.haar_detector = cv2.CascadeClassifier("Backend/Models/haarcascade_frontalface_default.xml")
```

De ezeket az adatokat fel is kell tudnunk dolgozni, így ehhez is egy mesterséges intelligenciát választottunk, még pedig a Facebook Prophet nevű szintén nyílt forráskódú és ingyenes könyvtárat. Ez számunkra azért volt jó választás, mivel nekünk egy olyan szoftver kellett, ami képes a periódikusan ismétlődő adatokat előrejelezni. A szoftvernek meg tudtuk adni, hogy a heti és évi szezonalitást figyelembe vegye, meg lehet adni, hogy a magyarországi ünnepnapokat is beleszámolja.

Ezen két könyvtár adja a program alapját szóval megpróbáltunk minél több időt fordítani a tanulmányozásukra.

## 2.2.3 Kamera architektúra

### 2.2.3.1 Kamerák felépítése

Ahogy a 2.2.1-es ábrán is látszódott, ez a rész 3 fontosabb osztályból fog állni és egy enumból. Azonban ebben a részben egy átfogó képet próbálok meg adni, hogy a kamerák, hogyan működnek.

Egy IPCamera létrehozásakor meg kell adni 4 paramétert. Az adott camera IP címét (url), nevét (name), státuszát, hogy éppen indított vagy leállított állapotban van (status) és az kép típusát (imgType), ez arra van, hogy egy IP-n megosztott kamera képét a cím után megadott image typeal lehet elérni, esetünkben /shot.jpg.

```
class IPCamera:
    def __init__(self, url, name, status, imgType):
```

Indításkor még pár fontos dolgot betölt a program, mint például a kor és nem, meg az időintervallumra vonatkozó adatok (2.2.3.3). Létrehozza annak a kameravizsgálati függvénynek a Threadjét, ami akkor kerül elindításra, ha a kamera státusza „Started”. Ezeken felül betölti a szükséges modelleket a program, ami alapján működik az arc/nem/kor felismerés.

```
self.cameraThread = threading.Thread(target=self.ipcamFaceDetect, args=())
self.writeThread = threading.Thread(target=self.writeCSV, args=())
self.age_model = cv2.dnn.readNetFromCaffe("BackEnd/Models/age.prototxt", "BackEnd/Models/age.caffemodel")
self.gender_model = cv2.dnn.readNetFromCaffe("BackEnd/Models/gender.prototxt",
                                             "BackEnd/Models/gender.caffemodel")
self.haar_detector = cv2.CascadeClassifier("BackEnd/Models/haarcascade_frontalface_default.xml")
```

Ebben a fejezetben szeretnék beszélni a kamerakép arcfelismerő részéről.

A függvény az elején megkapja a pontos elérési útvonalat a kameraképhez. Innentől kezdve addig fogja menteni az adatot amíg nem kapja meg az utasítást hogy le kell állnia.

```
def ipcamFaceDetect(self):
    urlshot = "http://" + self.url + "/" + self.imgType
    x = threading.Thread(target=self.saveImage, args=())
    x.start()
    while not self.stopped:
```

A függvényen belüli lényeges rész még a felismerés, itt a kamera adott frame-ét átalakítja szürkére, majd ezen lefuttatja az arcok felismerését. Majd végigmegy egy ciklussal az összes felismert arcon és az azokról talált adatokat összegzi és lementi.

```

if frame is not None:
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = self.haar_detector.detectMultiScale(gray, 1.2, 5)

    for face in faces:
        x, y, w, h = face
        detected_face = frame[int(y):int(y + h), int(x):int(x + w)].copy()
        img_blob = cv2.dnn.blobFromImage(detected_face, 1, (224, 224), self.MODEL_MEAN_VALUES, swapRB=False)

```

### 2.2.3.2 Kamera indítás, leállítás és adatbázis műveletek

A kamerákat arcfelismerő függvényei külön szálakon futnak. Tehát ha megérkezik az IPManagerből a hívás, hogy el kell indítani a kamerát akkor előtte elvégez pár ellenőrzést, hogy a kamera indítható állapotban van-e. Elsőnek le kell ellenőriznie, hogy online van-e a kamera, ehhez egy 5 mp-s timeoutot adtunk, ha addig nem válaszol akkor úgy vesszük, hogy offline állapotban van. Aztán leellenőrzi, hogy a kamera Paused állapotban van-e és a Thread sincsen elindítva ugyanis ezek is kizáró okok az indításra. Ha ezen mind átment akkor a már a konstruktorban létrehozott Threadet el lehet indítani.

Mind az indításkor mind a leállításkor szinkronban kell tartani a kamerának a státuszával. Erre figyeltünk mindenhol, hogy ne legyen inkonzisztencia.

```

def pauseCameraThread(self):
    if(not self.cameraAlive()):
        self.status = CameraStatus.Paused
        self.stopped = True
        return False
    if (self.status == CameraStatus.Started):
        self.status = CameraStatus.Paused
        self.stopped = True
        return True
    return False

```

Az adatbázisban tároljuk az ipcímekhez tartozó adatokat és egy listát ami a kamerákat és adatait tárolja amire szükség van egy kamera betöltésekor. Az emberekről gyűjtött adatokat két óránként menti le. Tehát ha az Intervallum kezelő osztály igent mond a menthetőségre, akkor le is menti.

```

if (self.intervalHandler.isDataSaveable() and not self.writeThread.isAlive()):
    self.writeThread.start()

```

### 2.2.3.3 Kor, Nem és Idő intervallum adatainak kezelése

Ezen osztályokból lényegesebb az Intervallum kezelő osztály, ugyanis a kor és a nemet összefoglaló PersonBucket osztály tulajdonképpen csak egy tárolóként működik egy két egyszerűbb segédfüggvénnyel.

Az intervallumokat kétórás ciklusokban határoztuk meg. Ez arra kell, hogy a jóslásnál ezekből a intervallumokból kell majd választani. Tehát az adatbázisba is két óránként kerülnek

be adatok. A ciklus ami veszi a kamera képét az minden futáskor megkérdezi az IntervalHandlert, hogy menthető-e a kép, ami csak akkor fog igent visszaadni, ha belépett egy ilyen két órás ciklusba. Arra is figyeltünk, hogy ha útközben kifagy a kamera akkor az adott állapotát lementse az adatbázisba.

```
def isDataSaveable(self):
    dateNow = self.getIntervalDateNow()
    if (self.lastInterval == self.defaultDate):
        self.lastInterval = str(dateNow)
    if (str(self.lastInterval) != str(dateNow)):
        return True
    return False
```

#### 2.2.3.4 Kamerák kezelése az IPManager osztályban

Ez az osztály segít abban, hogy a kamerákat egységesen tudjuk kezelni, és a Webserver közvetlenül ennek az osztálynak a függvényeit hívja, hogy elérje a kamerát.

Feladata, hogy listában tárolja az adatbázisba lementett kamerákat, azokat indításkor beolvassa, vagy esetleges leállás esetén ki is mentse azokat.

```
iplist = df["ipaddress"].tolist()
namelist = df["name"].tolist()
statuslist = df["status"].tolist()
imgTypelist = df["imgType"].tolist()

for i in range(len(iplist)):
    ipcamera = ic.IPCamera(iplist[i], namelist[i], ic.CameraStatus(statuslist[i]), imgTypelist[i])
    clist.append(ipcamera)
```

Ezen felül még feladata a kamera indítása, leállítása, listához való hozzáadása, onnan a törlése.

```
def addCamera(self, ipaddr, name, status, imgType):
    self.persistCamera(ipaddr, name, status, imgType)

    ipcamera = ic.IPCamera(ipaddr, name, ic.CameraStatus(status), imgType)
    self.cameraList.append(ipcamera)
```

#### 2.2.4 Predikció tervezése

Ebben a bekezdésben arról lesz szó, hogy az előrejelzéshez milyen átalakítások kellenek az adattáblán, milyen egyéb paramétereket vesz figyelembe a Prophet a predikciónál és hogyan mi az, amit outputként kapunk pontosan.



#### 2.2.4.1 Adattábla műveletek

A predikció osztály a `getPrediction` függvényén keresztül kapja meg a szükséges információkat az előrejelzéshez, úgy, mint jóslandó időpont, és az IP címe a kamerának amelyik az adatokat gyűjtötte.

Tehát az első feladatok közé tartozik, hogy a programnak az adattáblát át kell alakítania olyan formátummá, amit be tud fogadni a Prophet.

```
time,age,gender
2019-02-02 00:00:00,"[0, 0, 0, 3, 5, 0, 3, 1]","[5, 7]"
2019-02-02 02:00:00,"[0, 1, 1, 1, 3, 2, 0, 1]","[4, 5]"
```

A feladat az, hogy a végeredmény táblában csak egy időpont és egy számoszlop legyen. Ez alapján tud egy választott időpontra egy számot visszaadni. Tehát lesz egy `ds` oszlopa az új táblának:

```
def loadCamera(self):
    self.df = pd.read_csv('DB/cameras/' + self.ip + '.csv')
    self.df['ds'] = pd.DatetimeIndex(self.df['time'])
```

És lesz egy `y` oszlopa. Ez az `y` oszlop fogja jelképezni azokat a számokat amelyek ugyanazon kategóriába/intervallumba esnek (1-1 predikció minden korosztályra és mindkét nemre), tehát összesen el kell végezni 10 predikciót, mert 10db különböző oszlopunk van.

```
if i < 8:
    predictdf['y'] = self.df['age'].apply(lambda x: self.getValue(x, i))
else:
    predictdf['y'] = self.df['gender'].apply(lambda x: self.getValue(x, i - 8))
```

#### 2.2.4.2 Előrejelzéshez használt paraméterek

A Prophet meglehetősen testreszabható, így mi is beállítottuk a számunkra szükséges paramétereket.

```
predictdf['cap'] = predictdf['y'].max() * 1.1
predictdf['floor'] = 0

self.models.append(Prophet(interval_width=0.95, daily_seasonality=False, weekly_seasonality=True,
                             yearly_seasonality=True, growth='logistic', holidays=self.holidays))
```

A kép első két sorában az látható, hogy a kiszámolt értéknek egy lehetséges maximumot és egy minimumot adtunk meg, ezt muszáj volt beadni, ugyanis enélkül az előrejelzés gyakran adott negatív értéket.

Ezután látszódik a Prophet létrehozása. Itt ami lényeges az a seasonality, a growth és a holidays. A különböző seasonalitykben meg lehet adni, hogy mit vegyen figyelembe (napi, heti, éves szezonalitást). A growth annak muszáj logisticnek lennie, ugyanis csak ebben a számolási módszerben lehet megadni a minimumot és a maximumot. A holidaysben pedig megadtunk egy saját listából beolvasott ünnepnap táblázatot (a Prophet oldalán megadtak egy generáló scriptet ami 40 évre legenerálta Magyarország ünnepnapjait), így elég csak azt a táblázatot beolvastatni.

```
def loadHolidays(self):  
    df = pd.read_csv('DB/holidays.csv')  
    self.holidays = df
```

#### 2.2.4.3 Kimenet

Kimenetként elsősorban egy json objektumot ad vissza a Webservernek a predikció.

```
json = {  
    "ageBuffer": predInfo.ageBuffer,  
    "agePercentBuffer": predInfo.agePercentBuffer,  
    "genderBuffer": predInfo.genderBuffer,  
    "genderPercentBuffer": predInfo.genderPercentBuffer,  
    "ageIntervalInfo": predInfo.ageIntervalInfo,  
    "genderIntervalInfo": predInfo.genderIntervalInfo,  
    "colorBuffer": self.colorBuffer  
}
```

Másodsorban viszont képeket ment le az adatbázisba, hogy onnan a weboldal később vissza tudja majd olvasni és látványosabb legyen az adatvizualizáció.

```
def createImages(self, id) :  
    if id >= 0 and id < 6 :  
        self.figures[id].savefig('DB/predPhotos/predImage' + str(id) + '.png')  
        plt.close(self.figures[id])
```

Erről nem írnék részletesebben, mert a 2.3.4.3 -as fejezetben előjön a Frontend szempontjából.

#### 2.2.5 Webserver tervezése

Ebben a bekezdésben arról lesz szó, hogy a webserver hogyan épül fel, mit használunk, mik a fontosabb útvonalak a szerveren és ezek hogyan működnek.

Sokat gondolkoztunk, hogy szervernek mit használjunk, ugyanis ez a magja az egésznek, innen lesz irányítva a teljes program. Végezetül a Flask nevű Pythonban írt „micro web framework”. Mikro webframeworknek van elnevezve, ugyanis nincs szüksége különleges eszközökre és könyvtárakra, de amire nekünk szükségünk volt, vagyis, hogy különböző

```
@app.route("/prediction/p", methods=['POST'])
def getPrediction():
    data = request.get_json()
    request.close()
```

útvonalakra külön műveleteket csináljon és képes legyen json fájlok fogadására, arra tökéletes választás volt.

A szerver az elindulásakor létrehoz egy-egy példányt az IPManager és a Prediction osztályokból, hogy a kamerákat és az előrejelzést lehessen vezérelni. Majd ezek betöltése után elindul a szerver. Itt automatikus főoldalra az index.html által visszaadott oldalra kerülünk.

#### A fontosabb elérési útvonalak:

- „/” és „/prediction/”

Ez a fő oldal elérési útvonala, és csak kirendereli az index.html-t. A másik főbb oldal ugyanezt a műveletet hajta végre csak a prediction.html-t rendereli.

```
@app.route("/")
def index():
    return render_template("index.html")
```

- „/s:<id>” „/p:<id>” „/d:<id>”

Ezek az útvonalak akkor hívódnak meg amikor a kamerán műveleteket hajtunk végre. Az s, p és d betűk jelentik a start, pause és delete parancsokat. Az id meg egyértelműen az adott kamera id-ját jelenti. Így a függvényeknek az a feladata, hogy továbbítsa az IPManagernek azt, hogy melyik kamerán és mit akarunk elvégezni. Ezután, ha ez sikeres volt akkor 200 as html kóddal visszatér, ha nem akkor 502 és „Camera cannot be found”

```
@app.route("/p:<id>")
def pauseCamera(id):
    response = ipm.pauseCamera(int(id))
    print("Pause ID: " + id)

    if (response):
        writeable = ipm.editCameraStatus(id, 0)
        if(writeable):
            return "OK", 200
        else:
            return "Camera cannot be written", 502
    else:
        return "Camera cannot be paused", 502
```

- „/prediction/p”

Akkor hívódik meg amikor a predikciós oldalon a submit gombot megnyomtuk, tehát elkezdjük lekérdezni az adott adatok szerinti előrejelzést. Itt lekérdezi a javascript által

```
@app.route("/prediction/p", methods=['POST'])
def getPrediction():
    data = request.get_json()
    request.close()

    date = datetime.strptime(data['date'], "%Y-%m-%dT%H:%M:%S.%fZ")
    date = date + timedelta(hours=2)
    predicted = pre.getPrediction(date.strftime("%Y-%m-%d %H:%M:%S"), data['camera'])
    print(predicted)
    return jsonify(predicted)
```

felküldött json-t, ami az adatokat tárolja. Ezek segítségével már meg lehet hívni a Prediction osztály getPrediction metódusát, aminek elküldjük a lekérdezni kívánt időpontot és a kamera IP címét.

- „/atstart/”

Ezt azért kell indításkor meghívni, hogy összehangban legyen az adatbázisban és az IPManagerben tárolt státusz a kamerának (vagyis, hogy elv van-e indítva vagy sem)

```
@app.route("/atstart")
def cameraCheck():
    for id in range(len(ipm.cameraList)):
        try:
            ipm.editCameraStatus(id, ipm.cameraList[id].status.value)
        except:
            return "Camera cannot be started", 502
    return "OK", 200
```

Ezeket a routeokat gondoltam fontosabbnak megemlíteni, ugyanis a többi vagy nem végez elég dolgot, vagy nagyon hasonló, mint már egy bemutatott útvonal által végzett művelet.

## 2.3 FrontEnd

### 2.3.1 Felépítése röviden

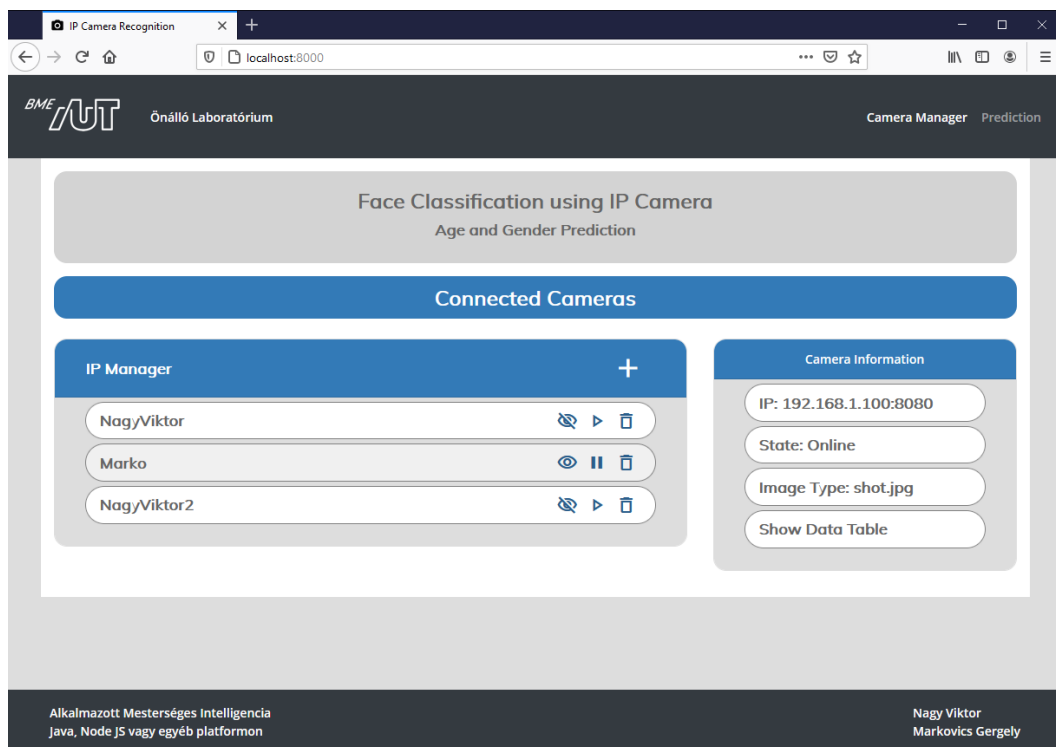
Az alkalmazás frontendje egy honlap, ami 2 felületből áll. Egyik felületen lehet kezelni a kamerákat, elindítani, megállítani és egyéb adatait megvizsgálni nekik. Másikon lehet a kamerák eddig lementett adatait felhasználva predikciókat végrehajtani, hogy a jövőben milyen értékeket várjon az adatbázis az emberek koráról, neméről, és ezek számosságáról.

### 2.3.2 Felhasznált eszközök

- A honlap használ bootstrap 4-et, annak nagyrészt cardbox és modal eszközeit.
- Használ google fontokat az ikonok megjelenítéséhez
- A honlap stílusait egy css fájlból olvassuk ki
- Használ JavaScriptet, két felülethez 1-1 javascript fájlt kötve
- A javascript fájlok erősen hivatkoznak ezentúl JQuery könyvtárra is.

### 2.3.3 Camera Manager leírása

#### 2.3.3.1 Megjelenített felület



### 2.3.3.2 Osztályok

A Camera Manager felület 1 osztályt és egy hozzá tartozó enumot tartalmaz:

- Camera osztály: Ez tartalmazza a kameráknak a felhasználói felületnek és vele a felhasználóknak fontos részeit, mint például, hogy hogy nevezték el a kamerát, vagy hogy mi az IP címe.
- CameraStatus enum: Ez az enum mondja meg az egyes kamerák állapotát, hogy éppen el van indítva vagy nem, vagy hogy most keresi az alkalmazás, hogy melyik a kettő közül.

```
function Camera(name, ip, status, imgType) {  
  this.name = name;  
  this.ip = ip;  
  this.status = status;  
  this.imgType = imgType;  
  this.beforestatus = CameraStatus.Paused;  
  this.canShow = false;  
}
```

```
const CameraStatus = {  
  Paused: 0,  
  Started: 1,  
  Pending: 2  
}  
Object.freeze(CameraStatus)
```

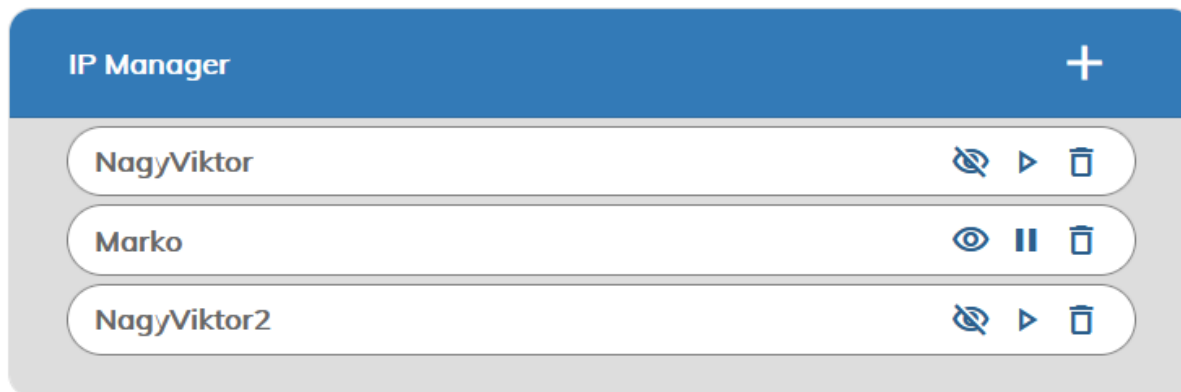
### 2.3.3.3 Inicializálás

Mivel a Camera Manager felület a honlap belépési pontja, ezért a honlapon használt adatok inicializálása is értelemszerűen itt történik. Erre szolgál az ip-script.js parseCameras() függvénye, ami a flask szerverünknek a „clist” route-járól lekérdezi a kamerák jsonben felküldött adatait. Itt ha a kamerának az van megadva, hogy ő aktív szeretne lenni, akkor kiegészítő lépésekre van szükség az egyszerű hozzáadáson túl. Meg kell figyelni, hogy a kamera valóban elindítható-e, erre szolgál a cameraStartable függvény.

```
if (c.status == CameraStatus.Started) {  
  var camera = new Camera(c.name, c.ip, CameraStatus.Pending, c.imgType);  
  cameras.push(camera);  
  cameraStartable(id, camera);  
} else {  
  var camera = new Camera(c.name, c.ip, c.status, c.imgType);  
  cameras.push(camera);  
}
```

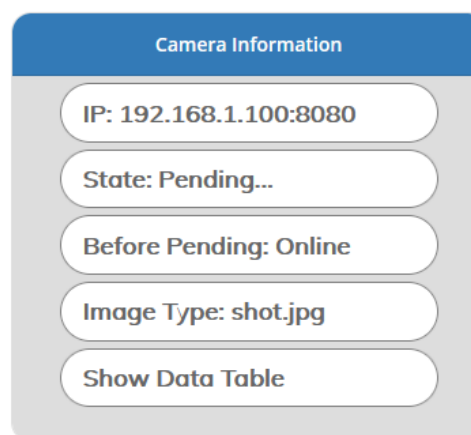
#### 2.3.3.4 Renderelés

Mivel a kamerák adatait a megjelenítő adatait átláthatatlan lenne egyesével változtatgatni, ha valami változott, ezért a Camera osztály adatait listában található adatok reprezentálását a honlapon egy erre kijelölt függvény kezeli, ez a renderCameras függvény. Annyi a feladata, hogy létrehozza az IP Manager cardboxban található listát az egyes kamerák adatai alapján, és annak gombjait bekösse.



#### 2.3.3.5 Kamera információk

Mint ahogy az első pontban is lehet látni, ha egy kamerára rákattintunk, akkor a Camera Information boxban megjelennek a hozzá tartozó információk. Ezt a funkciót a createCaminfo függvény valósítja meg. A másik boxhoz hasonlóan itt is egy listát hozunk létre, de nem az egyes kamerákról, hanem egy konkrét kamera információit rakjuk bele. Ezekbe az információkba tartozik bele a kamera IP címe, a jelenlegi státusza, Az útvonal a kamera IP címén, ahova a készített kép van mentve, és a szerver által eltárolt adatok megjelenítésére vezető fül. Ha szóba került a státusz, itt megemlíteném, hogy a szerver 5 másodpercet ad egy IP kamerának, hogy megmondja éppen aktívan üzemel, ezért a státusz lekérdezésének az eredménye is maximum ennyi időt tud követelni. Ameddig nem reagál, addig megjelenik egy kiegészítő információ az ablakban, hogy a keresés előtt milyen státuszú volt a kamera, ahogy az alábbi kép mutatja.



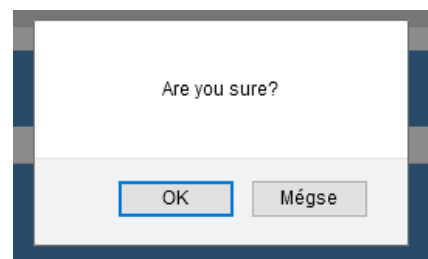
Az ablak egyetlen kicsit rejtélyes eleme a „Show Data Table”, ami egy olyan szolgáltatást ad, hogy egy modalban feljövő táblázatban láthatjuk az adott kamera csv fájljában lementett adatok utolsó 50 adatát. Így a felhasználó is láthatja, hogy pontosan mi folyik a

hátterben. Természetesen ez is a szerveren keresztül érkezik, amit a honlap AJAX lekérdezése a „data: {id}” útvonalon ér el. A táblázat így jön elő, ha rákattintunk az elemre:

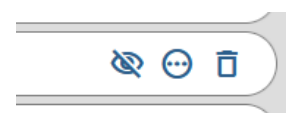
Camera Data Table											
Time	Age Data								Gender Data		
2021-01-01 22:00:00	1	0	1	8	12	9	10	3	18	26	
2021-01-01 20:00:00	2	3	9	26	42	34	23	19	65	93	
2021-01-01 18:00:00	4	5	9	31	61	27	45	39	84	137	
2021-01-01 16:00:00	3	13	14	28	63	41	52	50	84	180	
2021-01-01 14:00:00	5	9	19	38	64	49	44	49	109	168	
2021-01-01 12:00:00	5	4	10	30	52	28	40	25	75	119	
2021-01-01 10:00:00	1	7	11	9	34	28	35	25	68	82	
2021-01-01 08:00:00	0	6	10	8	17	12	10	9	29	43	
Data Headers:	0-6	6-12	12-18	18-26	26-36	36-48	48-60	60-100	Women	Men	

### 2.3.3.6 Kamerák gombjai

Ahogy fenti képeken látszik, az IP Manager ablakban az egyes kamerák nevei mellett 3 ikonnal jelzett gomb található. Jobbról az első egyértelmű, levesszük a kamerát a listáról, amit kezelni akarunk. Vannak ügyetlen felhasználók is, ezért ezt először egy alertdialog mögé rejtjük, hogy visszavonható legyen.

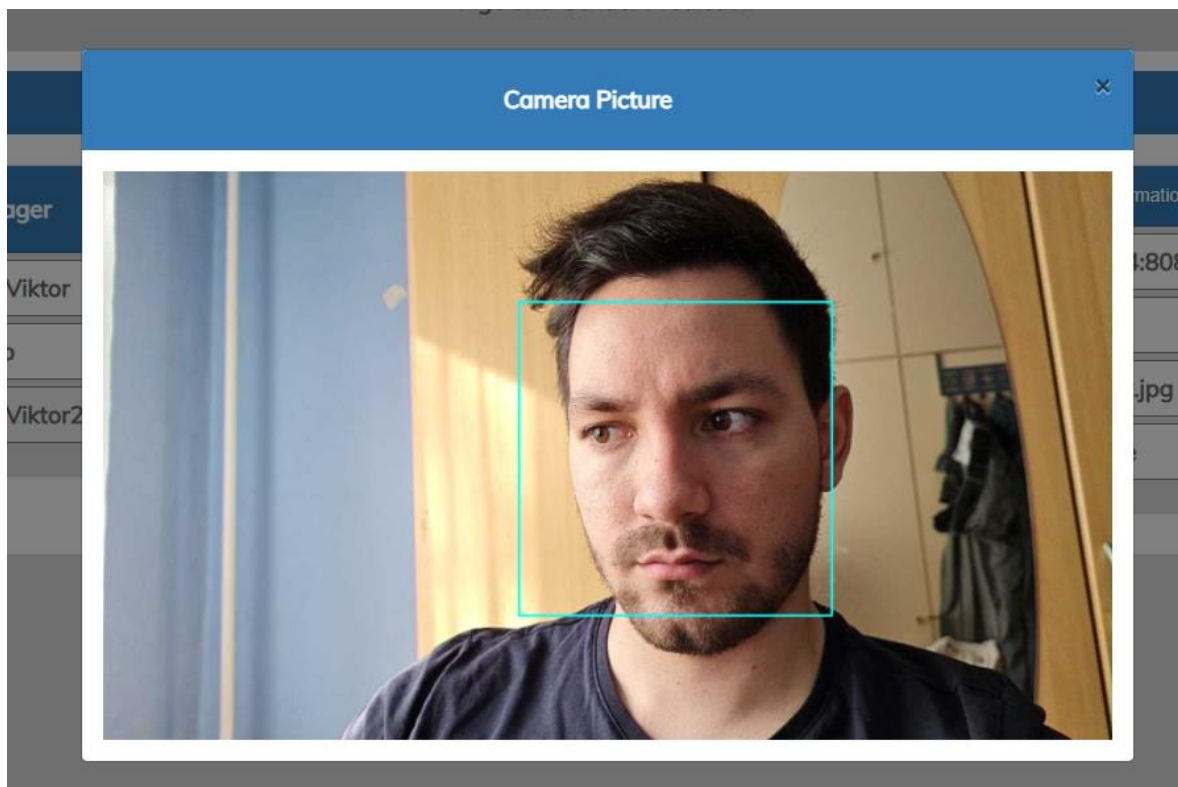


A középső elem is az, aminek látszik, ha a kameránk aktív, akkor a szerver el tudja rajta indítani az általa készített képek kezelését. Elindított állapotban a pause ikont jeleníti meg, leállított állapotban a start ikont a google fontjai között. Az 5 másodperces ellenőrzés itt is életben van, ennek jelölésére amíg keresi a kamerát, az ikon is pending ikonra vált.





Balról az első egy kicsit látványosabb funkció, ezzel meg tudjuk nézni, hogy éppen mi látható a kamerán egy modal ablakban. Ezt a képet másodpercenként frissíti, és a funkció csak akkor elérhető, ha a kamera kezelése el van indítva. Ezt jelezve az áthúzott szem ilyenkor sima szemre változik, és megnyomhatóvá válik. A fogott képen látható türkiz négyzet is a felismert arcok körül, mint az adott példán:



Ha valami olyat akarnánk tenni, amit nem szabad, akkor a snackbarok is megsegítenek.

Camera cannot start

### 2.3.3.7 Kamera hozzáadása

Még az a kérdés maradt hátra, hogy kamerát hogyan tudunk hozzáadni a listához. Erre van az IP Camera ablak jobb felső sarkában található plusz ikon, amire rákattintva megjelenik egy modal, ahol megadhatjuk a beállításokat. Értelemszerűen az alapbeállítások a név, ahogy a kameránkat nevezni akarjuk, és az IP cím porttal együtt, ahol megtaláljuk. Az esetleges nagyobb hibák elkerülése végett erre a két mezőre extra validációkat vezettünk be. Névre egy olyat, hogy a többi névhez képest egyedinek kell lennie. IP címnél óvatosabbnak kell lennünk, nem elég, hogy egyedi, de ugye a formátuma is fontos. IP kötelező formátumának azt adtuk meg, hogy 4 darab 0-255 közötti számnak kell egymást követnie köztük ponttal elválasztva, utána kettőspont után legalább 1 és legfeljebb 5 jegyű szám portnak. Ezeket a feltételeket a `checkname` és `checkip` függvény valósítja meg, és a validáció megszegését ezzel a függvénnyel jelzi a beviteli mezőn:

```
if (!unique) input.setCustomValidity('IP address must be unique');  
else if (!input.value.match(ipregex)) input.setCustomValidity('IP address must be valid');  
else input.setCustomValidity('');
```

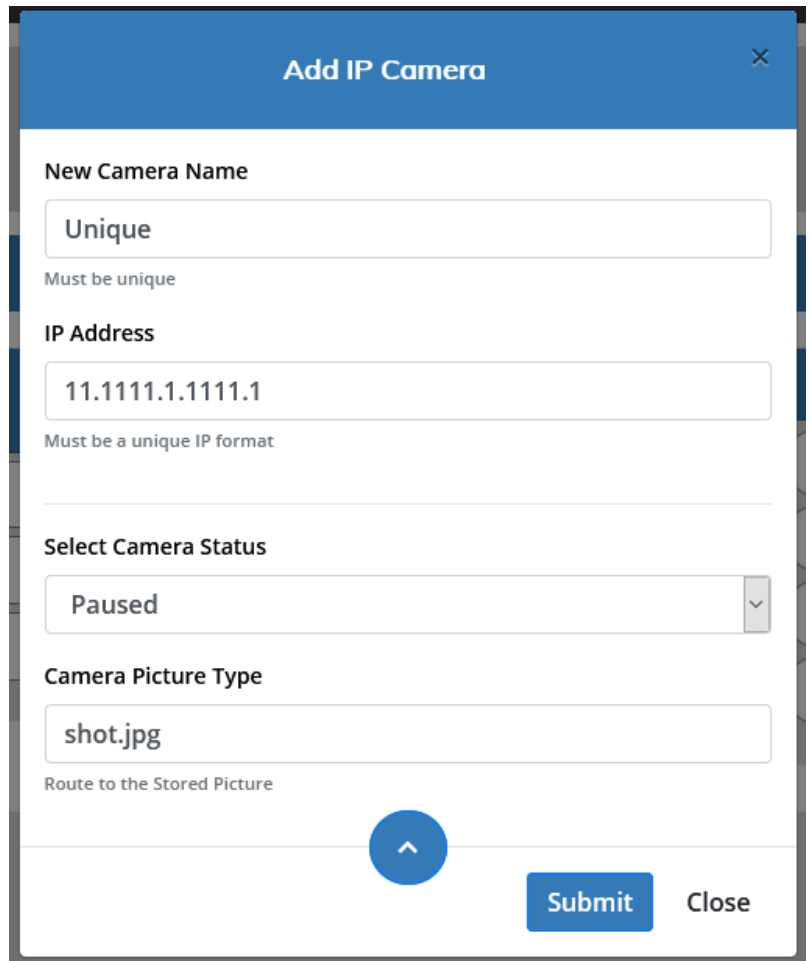
Egy-egy példa a megszegésre:

<p>New Camera Name</p> <input type="text" value="NagyViktor"/> <p>Must be unique</p> <div> Camera Name must be unique</div>	<p>IP Address</p> <input type="text" value="11.1111.1.1111.1"/> <p>Must be a unique IP format</p> <div> IP address must be valid</div>
--	---

Ezek voltak az általános beállítások a kameráknál, viszont ezenkívül lehet megadni speciális beállítások megadni, ha lenyitjuk őket a középen alul található gombbal. Ekkor 2 új beállítás jelenik meg. Megadhatjuk, hogy a kamera állapota milyen legyen létrehozáskor.

Kezdhethetünk rögtön elindított kamerával is. A második egy ennél fontosabb megoldás. Itt megmondhatjuk, hogy mi az az útvonal a kameránk címében, ahol a lementett kép található. Ez fontos a szerver számára, mivel itt keresi a képet, amit megfigyel. Az alkalmazás, amit használtunk tesztelésre a „shot.jpg” útvonalra mentette, ezért az alapbeállítás erre van rakva.

Ha mindent beállítottunk és a Submit gombra nyomunk, akkor el is indul a folyamat, hogy a kamera hozzá legyen adva a többi kamerához mind a szerveren, mind a honlapon.



### 2.3.3.8 Kommunikáció Predikció felületével

Mivel a kamerák változtatásával csak ez a fül foglalkozik, ezért felesleges a Predikció felületének is lekérdezni a kamerák állapotát, elég, ha a Camera Manager felület értesíti róla őt is. Ezért a kamerák adatait a Predikciós felület a Camera Managertől kapja LocalStorage-en keresztül, amibe JSON-be csomagolt kamera adatokat rakunk fel, és a Fogadó oldalon kibontjuk őket, így már a predikciókat is el tudjuk küldeni a megfelelő címekre a következő felületen.

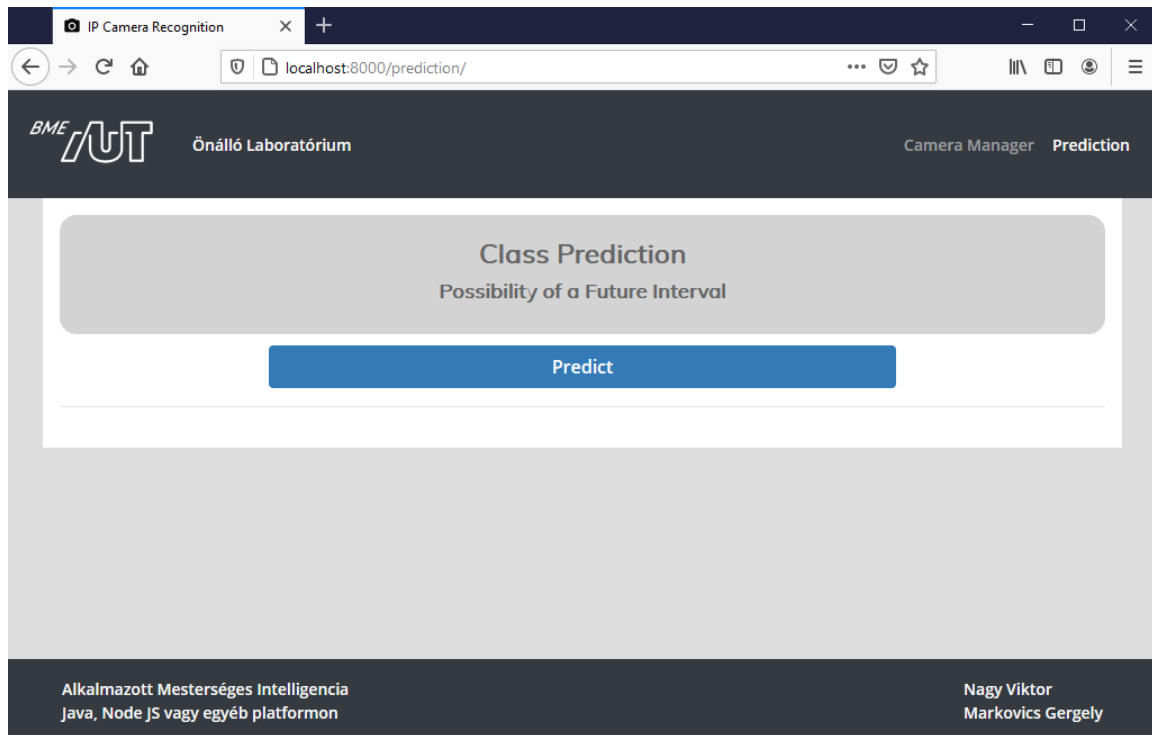
```
function saveCamerasToLocal() {  
    localStorage.setItem('cameras', JSON.stringify(cameras));  
}
```

Predikció javascript fájljában:

```
var cameras = JSON.parse(localStorage.getItem('cameras'));
```

## 2.3.4 Prediction leírása

### 2.3.4.1 Megjelenített felület



Ez a felület kicsit egyszerűbb, mivel egyetlen funkció megvalósítására van

### 2.3.4.2 Predikció indítása

Ahogy képen lehet látni, ezen a felületen egyetlen lehetséges lépésünk van, elindítani a predikciót a „Predict” gombbal. Ilyenkor megjelenik egy modal, amiben 3 dolgot választhatunk: melyik kamerára akarunk jósolni, milyen napra és a napon belül melyik intervallumra. A nap választásnál muszáj olyat választanunk, ami a jövőben van.

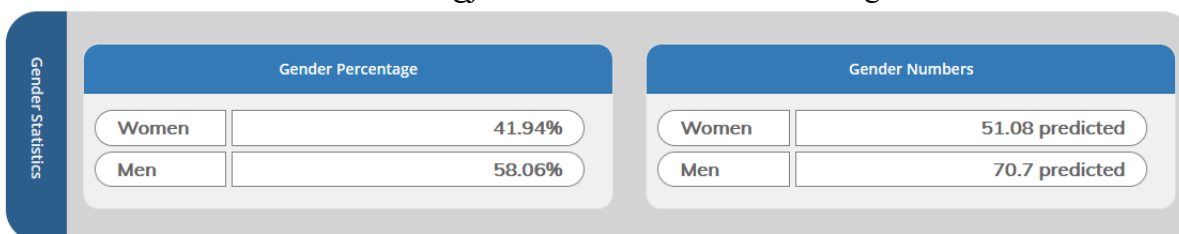
The modal is titled 'Predict Interval' and has a close button (X) in the top right corner. It contains three sections: 'Select Camera' with a dropdown menu showing 'NagyViktor - 192.168.0.176:8080'; 'Select Date' with a text input field showing 'éééé. hh. nn.'; and 'Select Interval' with a dropdown menu showing '00:00 ~ 02:00'. At the bottom right, there are two buttons: 'Submit' (blue) and 'Close' (gray).

### 2.3.4.3 Predikció eredménye

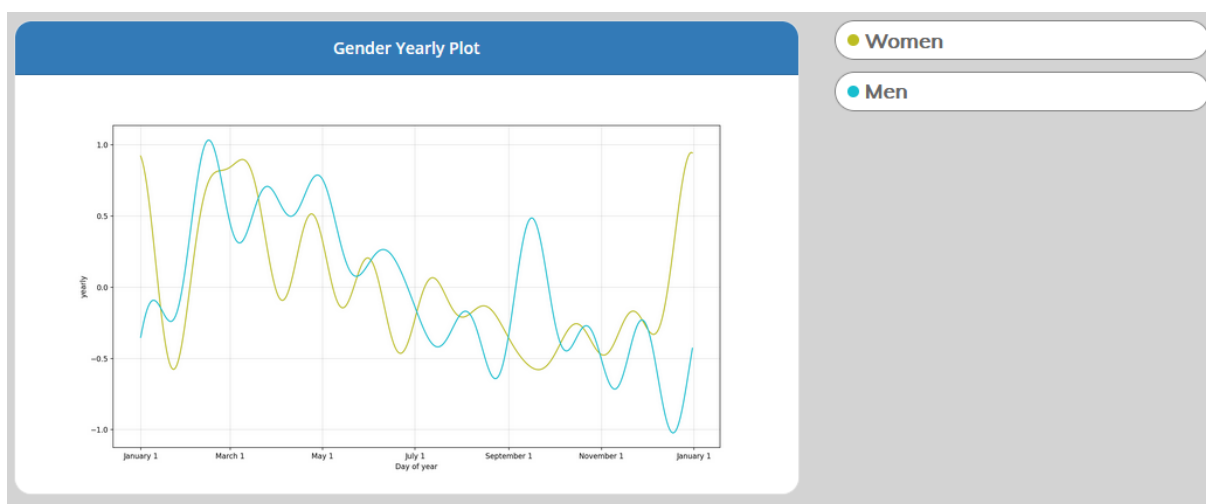
Mivel az adatbázis elég hosszú tud lenni, évek adatai lehetnek benne és azt is több változó szerint elemezzük, ezért ez a lépés sok időt vehet időbe, mostani példáinkkal olyan egy-másfél percre tart kiszámolni.

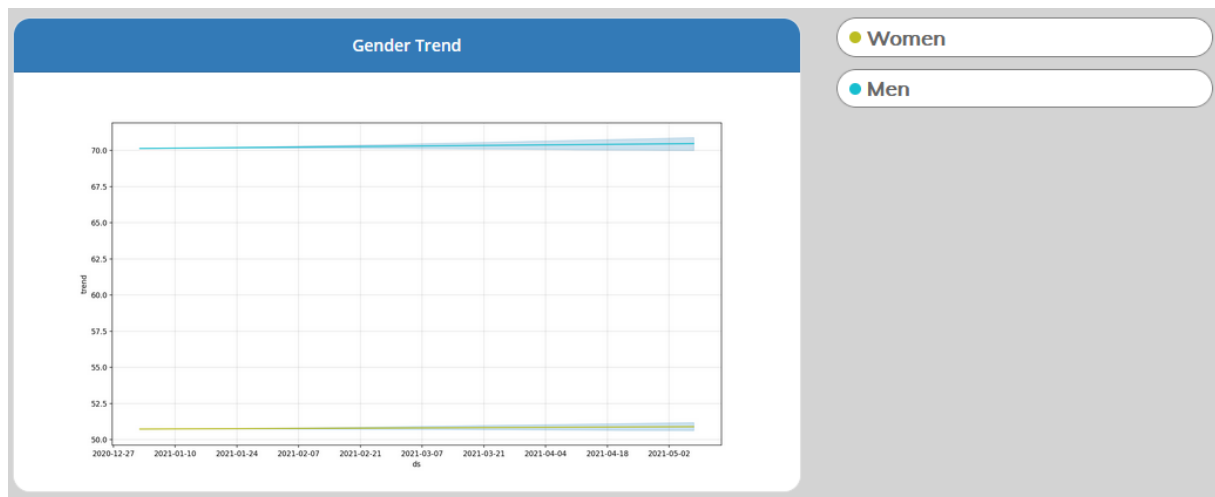
A predikció két lépésből áll: Először lekérjük a számossági adatokat, ami az időköltésesebb, ezután lekérjük ezek az adatok alapján a grafikonjaikat is. Az első a „prediction/p” útvonalon, a második a „prediction/img” útvonalon történik meg. A lekért adatokat külön kezeljük korbeli és nembeli adatokként, és 2-2 táblázat sorozatot hozunk létre. Az adatok feldolgozását a pred-script.jsben a loadData függvény valósítja meg, és a createList függvénnyel létrehozza a táblázatokat. Ezzel szimmetrikusan a képeket a createImage függvény helyezi el a weblapon.

A statisztikai adatoknál megjelenik százalékos és számosságbeli adatai is.



Ha a statisztikai adatok felkerültek, utána 3-3 grafikont töltünk be, és elhelyezzük mellette az adatok színekódját is az olvashatóság kedvéért. Létrehozunk egy grafikont az adatok trendjére, és az éves és heti ábrázolására is az adatoknak.





Mivel a példaadatokra nem található megbízható forrás, ezért a jóslást saját magunk által alkotott pszeudo adatokon kell elvégeznünk. Ezáltal nem tudjuk behangolni a beállításokat, hogy a lehető legjobban tükrözze a valóságot: Ezekből az okokból nem mondhatjuk, hogy a valóságban is megállná a helyét.