



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

Kis Tamás

BIZTONSÁGI KAMERA RASPBERRY PI ÉS JAVA ALAPOKON

Szakdolgozat

KONZULENS

Simon Gábor

BUDAPEST, 2016

Tartalomjegyzék

Összefoglaló	6
Abstract.....	7
1 Bevezető	8
1.1 A feladat értelmezése	8
1.2 A tervezés célja	8
1.3 A feladat indokoltsága	8
1.4 A szakdolgozat felépítése	9
2 A feladatkiírás	10
2.1 A feladatkiírás pontosítása	10
2.2 A feladatkiírás részletes elemzése	10
2.2.1 Alapkonstrukció.....	10
2.2.2 Megvalósítandó funkcionalitás	11
3 Előzmények.....	12
3.1 Hasonló megoldások.....	12
3.2 Következtetések	13
4 Tervezés	14
4.1 Döntési lehetőségek értékelése	14
4.1.1 Webszerver	14
4.1.2 Felhasználók kezelése.....	14
4.1.3 Kamera kezelése	15
4.2 Választott megoldások indoklása.....	15
4.2.1 Webszerver	15
4.2.2 Felhasználók kezelése.....	15
4.2.3 Kamera kezelése	16
5 Felhasznált technológiák	17
5.1 Szerveroldal	17
5.1.1 OpenCV	17
5.1.2 Gson.....	18
5.1.3 SIGAR	18
5.1.4 JavaBeans Activation Framework és JavaMail	19
5.2 Kliensoldal	20

5.2.1 jQuery	20
5.2.2 Bootstrap	21
6 Az elkészült megoldás felépítése	23
6.1 Szerveroldal	23
6.1.1 Modulok.....	23
6.1.2 Szolgáltatások	28
6.1.3 Logolás.....	30
6.1.4 Külső könyvtárak	30
6.2 Kliensoldal	31
6.2.1 HTML	31
6.2.2 JavaScript.....	32
6.2.3 Külső könyvtárak	33
6.3 Egyéb fájlok	38
6.3.1 content mappa	38
6.3.2 public mappa.....	38
6.3.3 conf.ini	38
6.3.4 .dll és .so	38
7 Az alkalmazás használta	40
7.1 State	41
7.2 Stream	41
7.3 Pictures.....	42
7.4 Settings.....	42
7.5 Logout	43
8 Értékelés	44
8.1 Teljesítmény tesztek	44
8.1.1 Erőforrás felhasználás.....	44
8.1.2 Terhelés teszt	45
8.1.3 Webes felület	45
8.2 Továbbfejlesztési lehetőségek	48
8.2.1 Jelenlegi funkciók optimalizálása.....	48
8.2.2 További funkciók implementálása.....	48
8.3 Tapasztalatok	49
9 Üzembe helyezés.....	50
9.1 Követelmények	50

9.1.1 Hardveres	50
9.1.2 Szoftveres.....	50
9.1.3 Raspberry Pi esetén.....	50
9.1.4 Alkalmas eszközök	51
9.2 Az alkalmazás elindítása.....	51
9.2.1 Paraméterek	52
Ábrajegyzék.....	53
Irodalomjegyzék.....	54

HALLGATÓI NYILATKOZAT

Alulírott **Kis Tamás**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2016. 12. 04.

.....
Kis Tamás

Összefoglaló

A feladat egy intelligens kamerarendszer szoftverének megalkotása volt, mely alkalmas egy Raspberry Pi miniszámítógépen is futni. Mindenképp egy olyan megoldással akartam előállni, ami Windowsos és Linuxos környezetben is megállja a helyét. A kamera szoftver lényegében egy webservert, így a böngészőből könnyedén lehet vezérelni és felügyelni a működését.

A technológiákat tekintve, backend részről a Javára esett a választásunk, mely alkalmas elég magas szintű programozásra és könnyedén futtatható bármilyen környezetben (cross-platform). A Java az 1.6-os verziótól kezdve támogatja HTTPServer készítését és futtatását, így adta magát a dolog, mivel egy JavaEE webservert beüzemelése sokkal több munkával járt volna mint ami a feladat megoldásához szükséges, és nem tudtuk volna kellő mértékben kihasználni a vele járó előnyöket.

Frontend részről az alap HTML és CSS mellett Bootstrap és jQuery is használva lett a könnyű fejlesztés végett. Ügyelve a tárolókapacitás végeességére, beállítható egy limit is a képek számát illetőleg.

Az így elkészült kamera képes képeket rögzíteni és tárolni, videót streamelni és mozgás esetén a tulajdonost értesíteni (e-mail). Az értesítő e-mailekben természetesen a mozgást kiváltó képkocka is elküldésre kerül.

A webes felületen megtekinthető a valós idejű videó stream, az eddig készített képek és lehetőségünk van az eszközt konfigurálni is. Mindezt egy teljesen reszponzív felületen, amit így egy okostelefon segítségével is könnyedén megtehetünk.

Abstract

My project was to create a software of an intelligent camera system, which is able to run on a Raspberry Pi minicomputer. My main goal was to create a solution which is usable on most of the operation systems like Windows and Linux. The camera software is basically a webserver, so it's easy to control and supervise from a browser.

Considering the technology, we decided to use Java for backend, which is applicable to high level programming and easily runnable on any platform. With Java (from v1.6), it's simple to create and run a webserver using the built-in `HTTPServer` class. Setting up a JavaEE webserver would have been more complicated and we wouldn't have been able to take advantage of the benefits associated with it.

Besides of basic HTML and CSS, Bootstrap and jQuery were also used for frontend because of the easy development. Considering the storage capacity, the user(s) can set a limit for the amount of the images.

The camera is capable to capture and store photos, stream videos and notify the owner (via e-mail) when motion detected. Naturally, the motion triggered frame will be also attached to the e-mail.

The website provides the possibility to watch real time stream, see previously captured photos and also configure the device. We can use all of these functions easily on a smartphone.

1 Bevezető

1.1 A feladat értelmezése

A feladat megoldása során egy olyan alkalmazást kell készíteni mely alkalmas az otthonunk védelmét biztosító feladatkört betölteni. A megoldás hardveres részét alkotó Raspberry Pi és a hozzá tartozó kamera, bárki számára könnyedén beszerezhető, így garantálva a széleskörű felhasználást.

A feladat előzetes specifikációja alapján a következő követelményeknek kell megfelelnie az elkészült megoldásnak:

- hardveresen kizárólag a Raspberry Pi miniszámítógép család modelljeit és az ezekkel kompatibilis hardverelemeket használhatja fel
- szoftveresen kizárólag nyílt forráskódú Java alkalmazásokat használhat fel
- Java nyelven kell íródnia
- reszponzív webes felületet kell nyújtania a kamera felügyeletére és vezérlésére
- képesnek kell lennie állókép készítésére és mozgókép streamelésére

1.2 A tervezés célja

A tervezés során az elsődleges feladat az volt, hogy az elkészült megoldás teljes mértékben eleget tegyen az előzetes feladatspecifikációnak. Ezen felül kiemelt figyelmet kapott az alkalmazás egyszerű használatának, könnyed üzembe helyezésének és a későbbiekben történő bővíthetőségének garantálása.

1.3 A feladat indokoltsága

Napjainkban egyre nagyobb igény mutatkozik a személyes értékek megfelelő védelmére és az otthonunk biztonságára is egyre nagyobb figyelmet fordítunk. Ezen alkalmazás megoldást nyújt bárki számára, hogy az otthonát illetve egyéb értékeit biztonságban tudja a nap bármelyik szakában.

1.4 A szakdolgozat felépítése

A továbbiakban a következő témakörökről lesz szó:

- a feladat pontosítása és részletezése
- hasonló megoldások
- saját megoldás felépítése
- felhasznált technológiák általános bemutatása
- elkészült megoldás bemutatása
- alkalmazás beüzemelése

2 A feladatkiírás

2.1 A feladatkiírás pontosítása

A feladat megoldása előtt pontosítani kellett bizonyos kritériumokat, amik alapján a későbbiekben a tervezés végbe mehet. A feladatkiírás alapján az alábbi témakörökben szabad kezet kaptam:

- Raspberry Pi-n futó operációs rendszer típusa (Windows, Linux)
- a Java alapú webservert alapja, megvalósításának módja (Apache Tomcat, egyéb megoldások)
- biztonság; autentikáció mikéntje (egy vagy több felhasználó, kezelésük módja)
- a kamerakezelés megoldása (képek fájlból vagy memóriából)
- biztonsági funkciók megvalósítása (mozgásérzékelés, értesítés)
- a webes UI kinézete, megjelenítése (JavaScript, HTML5, Bootstrap)

2.2 A feladatkiírás részletes elemzése

2.2.1 Alapkonstrukció

A feladat típusából adódóan két részre bomlott a feladat, szerveroldalra és kliensoldalra.

2.2.1.1 Szerveroldal

Szerveroldali részről első sorban azt kellett meghatározni milyen kész webservert alkalmazást használjak, aminek van Java API-ja. A feladatra természetesen több lehetőség is adott volt.

Fontos szerveroldali funkció még a kamera kezelése, amire olyan megoldást kell találni, ami relatíve gyors (a streamelés miatt) és ezen felül nem használ túl sok memóriát (a limitált erőforrások miatt).

2.2.1.2 Kliensoldal

A kliensoldal esetében nem kellett komolyabb funkcionalitást megvalósítani, így ami eldöntésre várt, hogy hogy nézzen ki és milyen funkciók legyenek kivezetve a felületre.

2.2.2 Megvalósítandó funkcionalitás

A kötelező funkciókon túl (kép készítése, videó streamelése), meghatározásra kerültek további funkciók, amik bővítik a kamera repertoárját.

2.2.2.1 Alap funkciók

Ezek azok a funkciók, amiket a feladatkiírás közvetve, vagy közvetlenül megkövetelt:

- biztonság: a kamera funkció csak bizonyos személyek számára engedélyezettek
- kamera vezérlése: a kamera állapotának megváltoztatása a webes felületen keresztül
- real-time streaming: a webes felületen lehetőséget kell biztosítani a felhasználó(k) számára a kamera élőképek megtekintésére
- fotó rögzítése: a felhasználónak képesnek kell lennie tetszőleges időpillanatban fénykép rögzítésére

2.2.2.2 Egyéb funkciók

Az alap funkciókon túl további lehetőségek biztosítása a kamera jobb használhatósága érdekében:

- mozgásérzékelés és értesítés: a kamera egy biztonsági funkciója, mely a megfigyelt területen történt mozgás esetén értesíti a kamera tulajdonosát és rögzíti az aktuális képkockát
- felhasználó kezelés: a felhasználó(k) adatainak módosítása
- kép limit: a limitált erőforrások miatt szükség lehet az elmentett képek számának limitálására
- rendszer monitorozása: az alkalmazás erőforrás felhasználásának megjelenítése

3 Előzmények

Fontosnak tartottam más fejlesztők munkáinak megismerését, és a témával kapcsolatos ismereteim elmélyítését mielőtt nekikezdttem volna az érdemi munkának.

3.1 Hasonló megoldások

A hasonló megoldások kutatása során kifejezetten olyan megoldásokat/ötleteket kerestem, melyek között szerepel Linuxos és Windowsos megoldás is, legyen az személyi számítógépre vagy Raspberry Pi-re tervezve. Ezen felül igyekeztem utána járni milyen lehetőségek adóttak a Raspberry Pi saját kamerájának felhasználására ([1], [2]).

Nagyon hamar kiderült, ahogy az várható is volt egy Java alapú rendszer esetén, hogy a Raspberry Pi-re és a személyi számítógépekre szánt megoldások közötti lényegi különbség elenyésző, így nem volt jelentősége külön egyik vagy külön másik platformra keresni megoldásokat. Ebből adódóan elegendő volt olyan lehetőségek után kutatni, ami alkalmas a Raspberry Pi natív kamerájának használatára.

Raspberry Pi esetén kifejezetten Windows operációs rendszerre írt megoldásokat nem találtam, hiszen mint kiderült nem támogatja jelenleg a Raspberry Pi-n futó Windows rendszer a Pi natív kameráját.

A Linuxos megoldások között az esetek nagy részében a Raspberry Pi saját kamerakezelő alkalmazásait hívják segítségül (raspistill és raspivid), majd az így készült fájlokat dolgozzák fel ([3], [4]).

Python nyelv esetén voltak könyvtárak a kamera kezelésére, viszont mivel a programozási nyelv kötött volt a feladatspecifikációban, így saját megoldást kellett kitalálnom.

3.2 Következtetések

- Windows specifikus megoldások elvetésre kerültek, mert így az alkalmazás képtelen lenne a Raspberry Pi natív kamerájának kezelésére
- kamera kezelése tekintetében találtam egy alternatívát az OpenCV külső könyvtár formájában melynek segítségével a képet közvetlenül a memóriába lehet tölteni, ezzel megspórolva a fájlműveleteket (Megjegyzés: az OpenCV eredetileg egy C/C++ könyvtárnak készült, viszont rendelkezik Java-s API-val)

4 Tervezés

A korábbi fejezetekben kiderült milyen lehetőségek jöhetnek szóba a feladat megoldása során, a továbbiakban bemutatom, hogy az adott megoldásoknak milyen előnyei és hátrányai vannak.

4.1 Döntési lehetőségek értékelése

4.1.1 Webszerver

A tervezés során meg kellett határozni, hogy a webszerver megvalósításának melyik módját választom. Az alábbi lehetőségek mind Java alapú, nyílt forráskódú megoldások:

- komolyabb JavaEE szerveralkalmazások (Tomcat, JBoss, Jetty...)
 - + gyors, jól optimalizált
 - + bevált, régóta használják
 - a feladat szempontjából túl bonyolult
- Java saját webszerver megoldása (HTTPServer)
 - + egyszerű, könnyen használható
 - komolyabb szerverfunkcionalitás megoldása nehézkes

4.1.2 Felhasználók kezelése

Fontos volt eldönteni, hogy a feladat megoldása során milyen módon oldjuk meg a felhasználók azonosítását (authenticáció):

- az alkalmazásból bővíthető, tetszőleges számú felhasználós környezet, SQL adatbázissal
 - + gyors
 - + tetszőleges számú felhasználó megoldható
 - feladat szempontjából túl bonyolult

- alkalmazásba/fájlba égetett, egy/keves számú felhasználós környezet
 - + egyszerű, kisszámú felhasználók esetén optimális
 - sok felhasználó esetén nehézkes a kezelésük

4.1.3 Kamera kezelése

Szükség volt meghatározni a kamera kezelésének módját az eddig összegyűjtött információk alapján:

- parancssoros megoldás (raspistill, raspivid)
 - + egyszerű használat
 - eltérő operációs rendszereken különböznek a parancsok
 - az elkészült kép egyből fájlba írásra kerül (felesleges fájlművelet)
- külső könyvtár (pl. OpenCV)
 - + egységes minden operációs rendszeren
 - + memóriába kerül a készített kép
 - + plusz funkciók (képfeldolgozás, képmanipuláció)
 - bonyolultabb használat

4.2 Választott megoldások indoklása

4.2.1 Webszerver

A webszerveres megoldások közül a Java saját `HTTPServer` osztályára esett a választásom, mivel a szerveroldali funkcionalitás nem követel meg komolyabb architektúrát és így könnyedén bővíthető az alkalmazás funkcionalitása.

4.2.2 Felhasználók kezelése

Mivel az alkalmazást egy időben egyszerre csak néhányan fogják használni és az adott felhasználókhöz nem lenne értelme külön beállításokat/adatokat tárolni, így megelégedtem az egy/limitált felhasználós rendszer megvalósításával, mert így jelentősen lecsökken a lehetséges hibák száma és a kis létszámú felhasználók esetén, azok kezelése is leegyszerűsödik.

4.2.3 Kamera kezelése

Mivel fontosnak tartottam, hogy az alkalmazás több platformon is megállja a helyét, és hasznosnak véltem egy külső képfeldolgozó könyvtár bevonását a projektbe a későbbi funkciókibővítés érdekében, így az OpenCV nyílt forráskódú függvénykönyvtár választása mellett tettem le a voksom.

5 Felhasznált technológiák

Az alábbiakban általánosságban bemutatásra kerülnek az általam felhasznált technológiák.

5.1 Szerveroldal

5.1.1 OpenCV

Az *OpenCV* (Open Source Computer Vision Library) egy open source gépi látáshoz és tanuláshoz használt szoftver könyvtár. Azért hozták létre, hogy egy egységes infrastruktúrát biztosítsanak a gépi látás alkalmazásoknak, és hogy felgyorsítsák gépi képfeldolgozást.

Több mint 2500 optimalizált algoritmust tartalmaz, melyek lefedik az alapvető gépi látáshoz és tanuláshoz szükséges eljárásokat. Ezen algoritmusok segítségével arcokat detektálhatunk és ismerhetünk fel, tárgyakat és emberi mozdulatokat azonosíthatunk, nyomon követhetjük a kamera és egyéb tárgyak mozgását, 3D pontfelhőt készíthetünk stereo-kamerával, képeket állíthatunk össze nagy felbontású képpé, egy adott képhez hasonló képeket kereshetünk ki képadatbázisokból, vörös-szemet távolíthatunk el egy adott képről, nyomon követhetjük a szemek mozgását, előre meghatározott markereket cserélhetünk ki a megjelenített képen 3D-s modellekre...stb.

A gyakorlatban, Izraelben őrzött területekre történő behatolások felderítésére, Kínában bányász eszközök megfigyelésére, Európában vízbefulladásos balesetek megelőzésére, Törökországban a kifutópályák tisztaságának ellenőrzésére, világszerte különböző termékek címkéjének ellenőrzésére és Japánban pedig gyors arc detektálásra használják.

Rendelkezik C, C++, Python, Java és MATLAB interfészekkel, és támogatja a Windowsos, Linuxos, Androidos és Mac OS-es felhasználását.

Az OpenCV natívan C++-ban íródott, és olyan sablonalapú interfésszel rendelkezik, mely zökkenőmentes használható fel STL tárolókkal. [5]

5.1.2 Gson

A *Gson* egy olyan Java könyvtár, mely képes Java objektumokat a nekik megfelelő JSON-ná alakítani. Ugyancsak alkalmas JSON stringeket a nekik megfelelő Java objektumokká alakítására. Tetszőleges objektumokkal tud dolgozni, az sem követelmény, hogy a használt osztályoknak a forráskódja rendelkezésre álljon.

Léteznek hasonló konverterek, viszont azok használatához szükséges annotációkat elhelyezni a konvertálandó Java objektumok osztályában, amit nehézkes megoldani, amennyiben nem áll rendelkezésre az adott osztály forráskódja. Ezen konverterek nagy része viszont nem nyújt teljes körű támogatást a generikus Java osztályokhoz sem. Ezzel szemben viszont a *Gson* tervezése során az imént említett problémák megoldása fontos szerepet kaptak.

A *Gson* tervezése, majd később fejlesztése során célul tűzték ki:

- `toJson()` és `fromJson()` függvények biztosítását, melyek segítségével könnyedén alakíthatóak Java objektumok JSON-ná oda és vissza
- a fejlesztő által nem módosítható osztályok konvertálásának támogatását
- a Java generikus osztályaira vonatkozó teljes körű támogatás garantálását
- személyre szabható konvertálási beállítások rendelkezésre bocsátását
- tetszőleges mélységig komplex objektumok konvertálását [6]

5.1.3 SIGAR

A *SIGAR* API egy egységes interfészt biztosít a rendszer adatainak eléréséhez, mint például:

- rendszer: memória, swap, processzor, load átlag, uptime, bejelentkezések
- folyamatok: memória, processzor, állapot, paraméterek, környezet, megnyitott fájlok
- fájlrendszer felismerése és hozzá tartozó metrikák
- hálózati interfészek detektálása, beállításainak adatai és hozzájuk tartozó metrikák

- TCP és UDP kapcsolatok adatai
- hálózati routing táblák

Ezek az adatok a különböző operációs rendszereken elérhetőek, viszont más-más módon.

A SIGAR egységes API-t biztosít a fejlesztőknek a használt platformtól függetlenül.

Az API magja (core) C-ben íródott, de rendelkezik az alábbi nyelvekhez tartozó bindingokkal: Java, Perl, Python, Erlang, PHP és C#. [7]

5.1.4 JavaBeans Activation Framework és JavaMail

A *JavaBeans Activation Framework*, vagy röviden JAF, lehetőséget nyújt a fejlesztők számára:

- típus nélküli adatok (arbitrary data) típusának meghatározására
- ezen adatok elérésére
- rajtuk végezhető műveletek meghatározására és a megfelelő osztály példányosításával ezen műveletek végrehajtására

Lehetőséget biztosít még típus nélkül adatokhoz tartozó típus létrehozására dinamikusan és egyedi adatokhoz műveletek párosítására.[8]

A *JavaMail* Java API segítségével e-maileket küldhetünk és fogadhatunk SMTP, POP3 és IMAP felhasználásával. [9]

5.2 Kliensoldal

5.2.1 jQuery

A *jQuery* egy gyors tömör és funkció-gazdag JavaScript könyvtár. Képes a HTML DOM bejárására és manipulálására, események kezelésére, HTML elemek animálására, és Ajax kérések küldésére. A sokoldalúságot és a módosíthatóságot kombinálva, a jQuery megkönnyítette emberek millióinak a JavaScriptben történő fejlesztést. Íme, néhány egyszerű példa az alkalmazására:

DOM bejárása és módosítása:

```
// a button elem kikeresése melyen rajta a continue class, majd a HTML
// tartalmának megváltoztatás
$( "button.continue" ).html( "Next Step..." );
```

Esemény kezelés:

```
// egy display:none-nal elrejtett #banner-message ID-jú elem
// megjelenítése, amikor a button-container ID-jú konténerben szereplő
// gombok egyike megnyomásra kerül
var hiddenBox = $( "#banner-message" );
$( "#button-container button" ).on( "click", function( event ) {
    hiddenBox.show();
});
```

Ajax:

```
// lekéri az időjárást a /api/getWeather URL-ről a zipcode=97201 query
// paramétert elküldve, majd a szerver válaszána sikeres megérkezés után
// a weather-temp ID-jú elemet feltölti a kapott válasz alapján
$.ajax({
    url: "/api/getWeather",
    data: {
        zipcode: 97201
    },
    success: function( result ) {
        $( "#weather-temp" ).html( "<strong>" + result
            + "</strong> degrees" );
    }
});
```

(Forrás: [10])

5.2.2 Bootstrap

A *Bootstrap* egy ingyenes open-source front-end keretrendszer, mely alkalmas weboldalak és webalkalmazások megjelenítésének személyre szabására. HTML és CSS alapú sablonokat tartalmaz a nyomtatási kép, a formok, a gombok és egyéb komponensek megjelenítésének módosítására. A többi keretrendszerrel ellentétben, kizárólag front-end fejlesztés megkönnyítésére alkalmas.

A Bootstrap kompatibilis az alábbi böngészők legfrissebb verziójának mindegyikével:

- Google Chrome
- Firefox
- Internet Explorer
- Opera
- Safari

A 2.0 verziótól kezdve támogatja a reszponzív web-design-t, azaz a weboldalak elrendezése követi a weboldalt megjelenítő eszköz képernyőjének méretét, legyen az asztali gép, tablet vagy akár mobiltelefon.

A 3.0 verziótól kezdve, a Bootstrap átvette a mobile-first design filozófiát, kihangsúlyozva a reszponzív web-design fontosságát.

A Bootstrap grid rendszere és reszponzív design-ja alapból 1170 pixel széles, természetesen ez a fejlesztés során módosítható. A biztosított eszközök több különböző képernyő méret (felbontás) kategóriát különböztetnek meg, hogy az eszközök mindegyikén megfelelő legyen a megjelenítés. A grid rendszerben meghatározott oszlopok szélessége a felbontás kategóriájától függően változik.

A Bootstrap az összes fontos HTML elemhez alapértelmezett stílust biztosít, melyek garantálják az egységes és modern megjelenítést a formázott szövegekhez, táblázatokhoz és a különféle formokhoz.

Ezek a stílusok előre implementálva vannak a Bootstrap CSS fájljában, használatukhoz egyszerűen csak el kell látni a HTML elemeket a megfelelő class nevekkal.

A Bootstrap ezen felül számos JavaScript komponenst is tartalmaz, melyek segítségével további elemekkel gazdagíthatjuk a felhasználó felületet, mint például dialógus boxok, tooltipok és carousel-ek. Ezen JavaScript komponensek nem csak új eszközöket biztosítanak, hanem a CSS által biztosított elemek funkcióját is bővítik, például így kaptak auto-complete funkciót az input mezők.

A 2.0-ás verziótól kezdve tovább bővült a JavaScript funkcionalitás (is) az alábbi eszközökkel: modalok, lenyíló menük, scrollspy, tabok, tooltipok, popover menük, alertek, gombok, collapse animáció, carouselek (slideshow) és typeahead kereső mezők. [11]

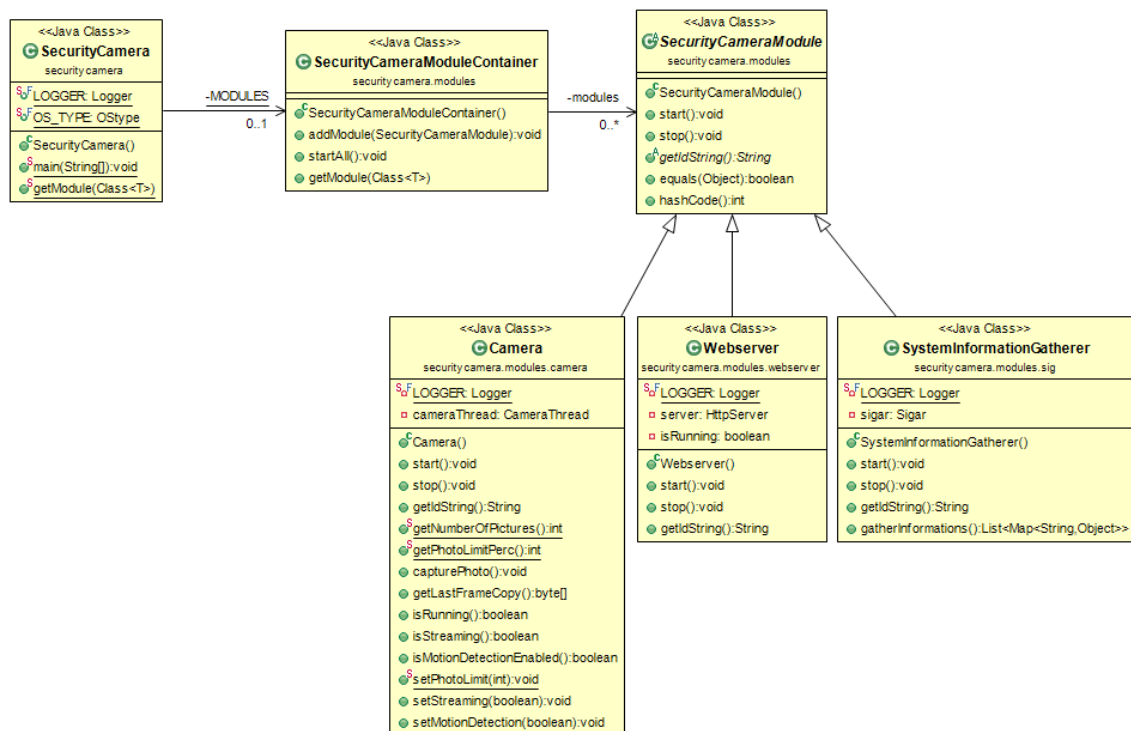
6 Az elkészült megoldás felépítése

6.1 Szerveroldal

6.1.1 Modulok

Igyekeztem egyfajta moduláris felépítést követni, így az egyes modulok fejlesztése, módosítása során nem szükséges az egész alkalmazást módosítani, elég csak az adott modulba belenyúlni. Ennek a felépítésnek további előnye, hogy egy-egy úgy funkciócsoport (modul) beépítése is nagyban leegyszerűsödik.

Éppen ezért, az alkalmazáson belül minden összetartozó funkció kód szinten egy modulnak (SecurityCameraModule) felel meg. Ezen a modulok kerülnek elindításra, megállításra, illetve ezeken a modulokon keresztül érjük el az alkalmazás funkcióit.



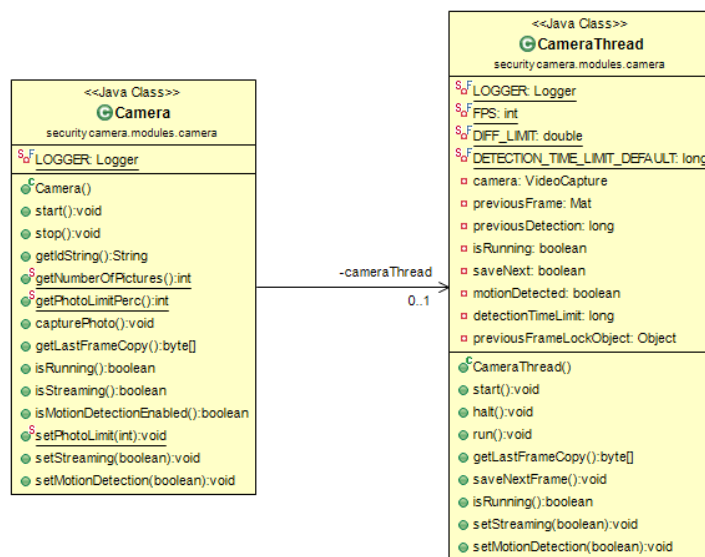
6.1. ábra – A megoldás során elkészített és felhasznált modulok

Az alkalmazáshoz három modult készítettem, de ez a későbbiekben gond nélkül bővíthető. Az alkalmazás elindítása során e modulokból hozunk létre egy-egy példányt, majd ezeket a beállításoknak megfelelően elindítjuk.

6.1.1.1 Camera

A Camera modul kiemelt szerepet kap az alkalmazás szempontjából, lévén hogy egy biztonsági kameráról beszélünk, így nem meglepő, hogy a funkcionalitás nagy része ide összpontosul.

Mivel a streamhez és a mozgásérzékeléshez folyamatos képkocka elérésre volt szükség, így magát a kamera „olvasását” egy külön szál végzi. A kamerát érintő kérések a Camera osztályon keresztül jutnak el ehhez a szálhoz, majd ugyancsak a Camera osztályon keresztül kerülnek kiszolgálásra.



6.2. ábra – Camera modul

A kamera bekapcsolása esetén egy az OpenCV csomag saját osztálya (VideoCapture) kerül példányosításra, mely a jelen paraméterezés szerint egyből csatlakozik az eszközhöz csatlakoztatott alapértelmezett kamerához.

```
// csatlakozás az alapértelmezett kamerához
camera = new VideoCapture(0);
```

A képfeldolgozás pipeline tervezési mintát követi, miszerint az alkalmazás a folyamat elején készít egy képet, majd a különböző szűrők és algoritmusok egymást követve érvényesítik a hatásukat. Jelenleg ez a felépítés nincs teljes mértékben kihasználva, viszont a későbbiekben remek alapot teremt a különböző képfeldolgozási eljárásoknak.

```
// egyetlen képkocka (Mat) elkérése a kamerától
Mat frame = new Mat();
camera.read(frame);
```


Maga a mozgásérzékelés is ebben a modulban kap helyet. A képkockák folyamatos elkérése (memóriába töltése) során, minden egyes ciklus végén csak a két utolsó képkockát tároljuk. Amennyiben a mozgásérzékelés aktiválva van, a ciklus végén az utolsó két képkocka között egy differencia számítás megy végbe.

```
double errorL2 = Core.norm(previousFrame, frame, Core.NORM_L2);
double difference =
    errorL2 / (double) (previousFrame.rows() * previousFrame.cols());
difference *= 100;
```

A két kép különbségének számítása pixel szinten történik, és végeredményben egy százalékos értéket kapunk. Amennyiben ez az érték meghaladja a küszöbértéket, mozgást detektáltunk. Ezt az értéket próbálgatás útján határoztam meg (1.75%). Fontos hogy ne legyen túl szigorú a detektálás, mert különben a különböző képen megjelenő zajokat, vagy akár a megfigyelt területen történő fényviszony változásokat (például ha beborul az ég) is mozgásnak érzékelné. Viszont arra is vigyázni kellett, hogy ne legyen túl magas ez az érték, mert akkor meg egy kellően lassan mozgó tárgyat/személyt nem detektálná.

A modul megvalósítása során előkerült egy hiba, ami egyre nagyobb memória fogyasztásban mutatkozott meg. Az egyre növvő memória igény memória szivárgásra enged következtetni, viszont a Java garbage collectorának kezelnie kellett volna ezt az esetet. A mivel a programban kód szinten nem találtam hibát, így a Java VisualVM segítségével kezdtem vizsgálni az alkalmazás memóriahasználatát. A különböző objektumok heapben foglalt méretét vizsgálva fény is derült a hiba lehetséges okára, ami kis utána olvasással be is igazolódott. A hiba oka az volt, hogy mivel az OpenCV alapjaiban C/C++-ban íródott, így egy képkocka adatait tároló Mat objektum plusz memóriát foglal az adatoknak, viszont magában az objektumban „csak” a terület memória címe van. Egy memória cím viszont relatíve kicsi, így a garbage collector nem siette el az objektum felszabadítását. Az alkalmazás viszont mivel élőképet kezel, így másodpercenként 25 ilyen objektumot foglalt.

A megoldása nem volt bonyolult, egyszerűen a már nem használt képkocká(ka)t a ciklus végén az OpenCV saját függvényével (release) felszabadítjuk, így a garbage collectornak már csak a Mat objektum felszabadításával kell törődnie.

```
if (previousFrame != null) {
    previousFrame.release();
}
previousFrame = frame;
```

6.1.1.2 Webserver

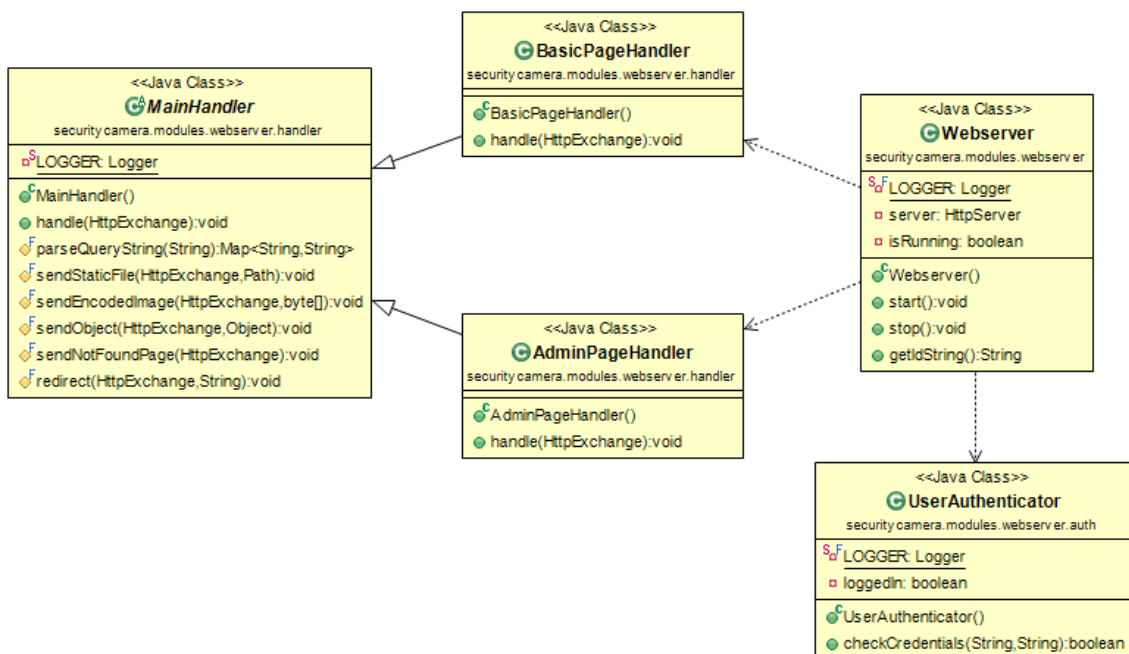
A 80-as porton érkező HTTP kérések kiszolgálásáért felelős modul. A beérkező kérések szétválogatásra kerülnek az URL és a kérés típusa (GET, POST) alapján, majd kiszolgálja a kérésnek megfelelően.

A felületekhez (jelenleg) kétféle jogosultsági szint társítható; publikus (bárki láthatja) vagy privát (csak bejelentkezés után látható). Az oldalak láthatóságának beállítása a Webserver osztályon történik. A felhasználók azonosítását (authenticációját) egy BasicAuthenticator végzi, az én esetemben ez az UserAuthenticator osztály. A basic authenticáció beállítása a handlererekhez a következőképpen történik:

```
server.createContext("/", new BasicPageHandler())
    .setAuthenticator(new UserAuthenticator());
server.createContext("/admin", new AdminPageHandler())
    .setAuthenticator(new UserAuthenticator());
```

Az authenticáció során a begépett jelszó SHA-256 hashelése után összehasonlításra kerül a beállításokban tárolt jelszóval.

Jelenleg egy felhasználó fér hozzá a privát oldalakhoz, mivel a feladat szempontjából ennyi is elegendő, viszont újabb felhasználók hozzáadása is megoldható.



6.3. ábra – Webserver modul

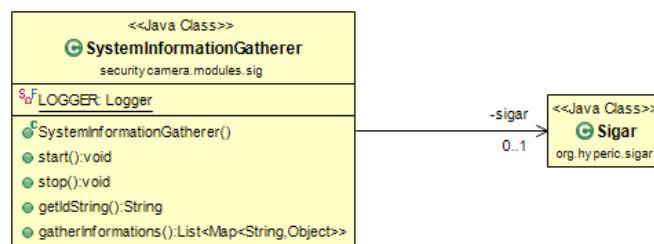
Az kérés kiszolgálásának módja a handlerekben került implementálásra. Azért volt szükség két osztályra, hogy a kód jobban áttekinthető legyen, és a későbbiek a funkciók bővítése se okozzon gondot.

Az *AdminPageHandler* kizárólag a /admin kezdetű URL-ekre érkező kéréseket kezeli (beállítások megjelenítéséért és módosításukért felelős weboldal), míg az összes többi kérés kiszolgálása a *BasicPageHandler*-en keresztül történik.

A két handler közös őse (MainHandler) valósítja meg az általános műveleteket, mint például az átirányítást, adatok küldését vagy a hibaüzenetek (404 Not found) továbbítását a kliens felé.

6.1.1.3 SystemInformarionGatherer

A rendszer monitorozásához, a különféle adatok (processzor és memória használat) elérését egy külső nyílt forráskódú könyvtárral oldottam meg (Sigar), melyet ez a modul foglal magába.



6.4. ábra – SystemInformarionGatherer modul

A használata nagyon egyszerű, a Sigar osztály példányosítását követően valósidejű adatokat kérhetünk az adott eszköz erőforrásainak állapotáról. Én az egyszerűség kedvéért csak a processzor adatokat (processzor gyártója és felhasználása) és a memória használatára vonatkozó adatokat kértem le az alábbi módon:

```

CpuInfo[] cpuInfoList = sigar.getCpuInfoList();
CpuPerc[] cpuPercList = sigar.getCpuPercList();

for (int i = 0; i < cpuInfoList.length; i++) {

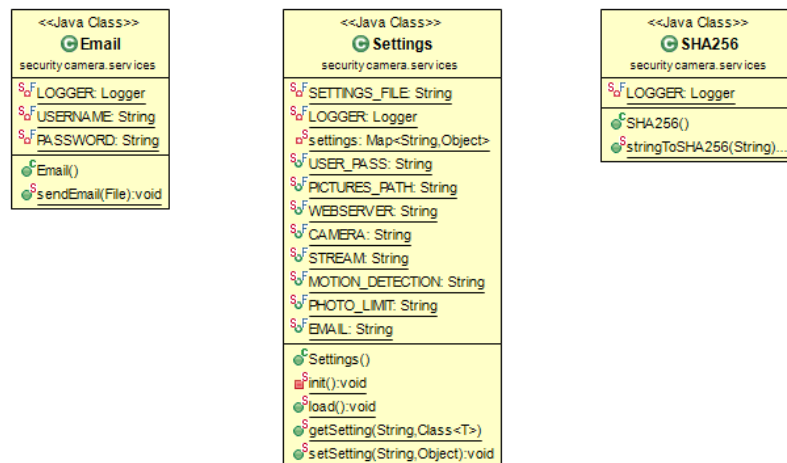
    CpuInfo cpuInfo = cpuInfoList[i];
    CpuPerc cpuPerc = cpuPercList[i];

    String cpuName = cpuInfo.getVendor() + " " + cpuInfo.getModel()
        + " (" + (i + 1) + ")";
    long cpuUsagePerc = Math.round(cpuPerc.getCombined() * 100);
}

Mem mem = sigar.getMem();
long memUsagePerc = Math.round(mem.getUsedPercent());
  
```

6.1.2 Szolgáltatások

Az olyan funkciókat, melyek nem szükségesek hogy folyamatosan fussanak, azaz a funkcionalitásukra csak időnként van szükség, egy-egy szolgáltatás foglal magába. Ezek a szolgáltatások mindentől független funkcionalitást valósítanak meg, melyeket tipikusan egy-egy statikus függvényen keresztül érünk el.



6.5. ábra – A megoldás során elkészített és felhasznált szolgáltatások

6.1.2.1 Email

Az e-mail elküldéséért felelős szolgáltatás. Jelenleg csak mozgás érzékelést követően van rá szükség.

Ahhoz hogy Java-ból e-mailt tudjunk küldeni, szükség volt egy mail szerverre, amit mivel nem akartam kézzel beüzemelni, így a Google e-mail szolgáltatására esett a választásom. A szolgáltatás igénybevételéhez szükség volt egy Gmail-es e-mail címre és a hozzá tartozó jelszóra.

Gyors regisztrációt követően már nem volt más teendő, mint összerakni magát az e-mailt, melyet mozgásérzékelés esetén elküldünk:

```
MimeMessage message = new MimeMessage(session);

message.setFrom(new InternetAddress(USERNAME));
message.addRecipient(Message.RecipientType.TO,
    new InternetAddress(emailAddress));
message.setSubject("Motion detected!");

BodyPart messageBodyPart1 = new MimeBodyPart();
messageBodyPart1.setText(
    "The security camera detected motion.
    The captured photo has been attached to this email.");
```

```

BodyPart messageBodyPart2 = new MimeBodyPart();
DataSource source = new FileDataSource(attachment);
messageBodyPart2.setDataHandler(new DataHandler(source));
messageBodyPart2.setFileName(attachment.getName());

Multipart multipart = new MimeMultipart();
multipart.addBodyPart(messageBodyPart1);
multipart.addBodyPart(messageBodyPart2);

message.setContent(multipart);

Transport.send(message);

```

Ahogy az a fenti kódrészletből jól látszik, kezdetben beállítjuk a feladót, a címzettet és az e-mail tárgyát, majd ezt követően két BodyPartot készítünk elő. Az elsőbe kerül az e-mail szövege, a másodikba pedig a mozgás érzékelésekor került kép. A két BodyPartot végül egyesítjük (MultiPart) és az üzenethez adjuk.

6.1.2.2 SHA256

A begépelt jelszó hashelése, majd összevetése az eltárolt jelszóval, az SHA256 szolgáltatásnak a segítségével történik.

Magát a hashelési eljárást nem kell túlzottan ismernie a fejlesztőnek (sem) ahhoz megkapjuk az átalakított jelszavat.

```

MessageDigest digest = MessageDigest.getInstance("SHA-256");

byte[] hash = digest.digest(s.getBytes(StandardCharsets.UTF_8));
String result = DatatypeConverter.printHexBinary(hash);

```

6.1.2.3 Settings

Az alkalmazás minden (újra)indítást követően betölti a legutolsó mentett állapotát beállításait. Az aktuális beállítások elkéréséhez vagy módosításához ez az osztály áll rendelkezésre.

Ugyancsak ez az osztály felelős:

- módosult beállítások fájlba mentéséért
- induláskor a legutóbbi beállítások betöltéséért

A beállítások egyszerűbb kezelhetősége érdekében a conf.ini fájlban szereplő adatok egy valid JSON-t alkotnak. Amennyiben ez a fájl törlésre kerül, akkor az alkalmazás kódjába „égetett” értékek kerülnek beállításra, így ha véletlenül elfelejtenénk a jelszavunkat, ezen fájl törlésével visszaállíthatjuk azt az alapértelmezettre (admin).

6.1.3 Logolás

Az alkalmazás futását befolyásoló minden fontosabb esemény logolásra kerül, így az esetleges hibák forrását egyszerűbb felderíteni.

Ezen feladat megoldására a Java saját loggerét használtam fel (*java.util.logging.Logger*), mely az 1.4-es verziótól a programnyelv része.

A logolás alapvetően a sztenderd kimenetre történik, viszont hogy később is vissza lehessen követni az alkalmazás logjait, így az alkalmazás megfelelő paraméterezett elindításával fájlba is kiíródik a log.

6.1.4 Külső könyvtárak

A szerveroldali funkcionalitás megvalósításához többféle külső könyvtárat felhasználtam, természetesen ezek a könyvtárak mind nyílt forráskódú projektek.

6.1.4.1 OpenCV

Sajnos a projektem során nem tudtam kihasználni teljes mértékben a tudását, de ez a későbbiekben, a projekt továbbfejlesztése során változhat. Jelenleg kizárólag a kamera kezelése miatt van rá szükség (kép készítése és a képkockák közötti különbség mértékének meghatározása).

6.1.4.2 Gson

Mivel a kliensoldali JavaScript objektumai gond nélkül válthatóak JSON-be és vissza, így a szerveroldalról kiszolgált dinamikus tartalmak JSON-ben érkeznek meg a kliensoldalra. Ezért volt szükségem egy olyan Java megoldásra, ami képes Java objektumokat JSON-né konvertálni.

A Gson felhasználását a projektem során az alábbi módon végeztem:

```
Gson gson = new Gson();
String data = gson.toJson(object);

OutputStream responseBody = exchange.getResponseBody();

exchange.getResponseHeaders().add("Content-type", "application/json");
exchange.sendResponseHeaders(200, data.length());
responseBody.write(data.getBytes());

...
```

6.1.4.3 SIGAR

Mivel a feladat megoldása során szükségem volt különböző rendszeradatokra, amiket a webes felületen meg is jelenítettem, így a SIGAR Java-s API-ja nagyban megkönnyítette a feladat megoldását.

6.1.4.4 JavaBeans Activation Framework és JavaMail

Az én feladatom szempontjából ez azért volt szükség a JAF-re, mert amikor e-mailt küldünk a tulajdonosnak arról, hogy mozgást érzékelünk, csatoljuk hozzá az elkészült képet is, ami így egy típusatlan adatfolyamként fog a címzetthez megérkezni.

Természetesen ehhez elengedhetetlen az, hogy az alkalmazás képes legyen e-mail küldésére, melyet a JavaMail külső könyvtár segítségével old meg.

6.2 Kliensoldal

A kliensoldal esetében, alapvetően statikus HTML fájlokról beszélhetünk, melyeket egy JavaScript kód tölt fel dinamikus adatokkal.

6.2.1 HTML

A statikus HTML oldalak megírásának során igyekeztem a legfrissebb technológiákat felhasználni.

Az oldalak mindegyikében megtalálható egy HTML5-ös tag (nav), mely az oldalak közötti navigálásra szolgál.

Az eszköz és az alkalmazás állapotát megjelenítő oldal (State) és a beállítások megjelenítésére szolgáló oldal (Settings) táblázatos elrendezést követ (table). Ezek a táblázatok kerülnek feltöltésre a JavaScript kód által.

A streamelést megjelenítő oldal (Stream), lényegében egy divbe ágyazot két img tagból áll. Ezeknek a kép tageknek (img) az attribútumai kerülnek változtatásra a stream nézése folyamán. Ez a két img tárolja a jelenleg megjelenített és a következőnek megjelenítendő képkockát. Mivel nagyon sűrűn cseréljük ki a képet az oldalon, így bizonyos böngészőknek nem is volt elég ideje betölteni és megjeleníteni a képet, amikor már ki kellett cserélni a következő képkockára. Ez a hiba a stream villogását okozta.

A hibát végeredményben úgy sikerült megoldani, hogy mindig két képkocka szerepel az oldalon, viszont az egyik nincs megjelenítve (`display: none`). Az újonnan érkező képkocka először megjelenítés nélkül kerül az oldal tartalmába, majd mikor a következő képkocka érkezik, megjelenítésre kerül. Így amíg nem megjelenített állapotból megjelenített állapotba kerül, van elég ideje a böngészőnek betölteni a képet.

Ennek a megoldásnak viszont ára van, a stream az élőképhez képest egy képkockányit késik. Ez 24 FPS esetén $\sim 41,6$ ms késést jelent, ami úgy gondolom elfogadható késleltetés ahhoz, hogy élvezhető minőségű legyen a stream.

Az előzetesen rögzített képeket megjelenítő oldal (Pictures) kizárólag `divek`ből és a megjelenített képek `img` tagjeiből állnak, melyek megfelelő elrendezéséért a CSS a felelős.

6.2.2 JavaScript

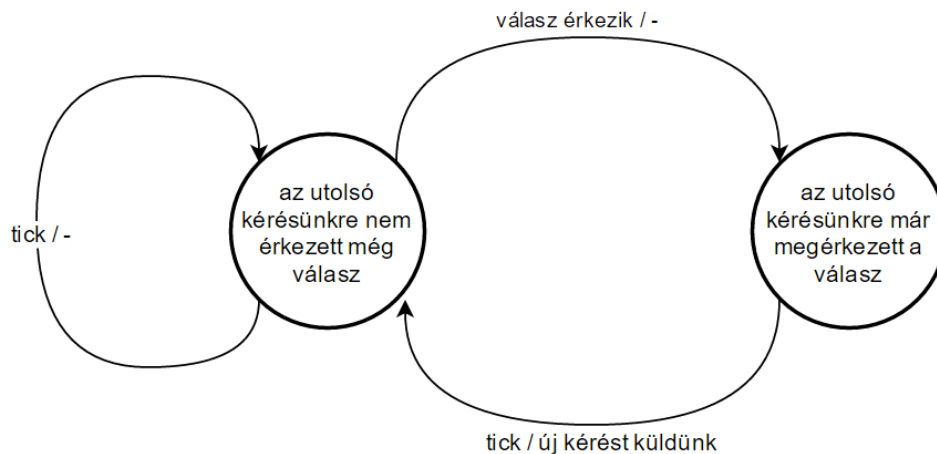
A kliensoldali funkcionalitás JavaScript nyelven íródott, mely feladata az oldal dinamikus tartalommal való feltöltésében és a betöltött adatok frissen tartásában nagyjából ki is merül.

Ezen felül a különböző gombok és kapcsolók funkciójai is JavaScript nyelven kerültek implementálásra, melyek aktiválását a nyelv AJAX kérések formájában továbbítja a webszerver felé.

A JavaScript funkcionalitás implementálása során, a streamet megjelenítő oldalon előkerült egy probléma, ami megoldásra várt. A gond az volt, hogy a JavaScript kód korábbi változataiban, a kliens folyamatosan, adott időközönként kérte el a következő képkockát, viszont amennyiben a szerver válasza annak terheltsége miatt késett, a kliens nem törődve a problémával, ettől függetlenül tovább bombázta a szervert kérésekkel. Ennek az lett a következménye, hogy az alapból kis késésből sokkal nagyobb késés született.

A probléma megoldása nem volt túl bonyolult, egyszerűen amikor kérnénk a következő képkockát, amíg még nem érkezett meg az előző kérésünkre a válasz, addig nem küldünk új kérést.

Az elkészült állapotgép az alábbiak szerint néz ki:



6.6. ábra – Streamelés folyamatának állapotgépe

A tick esemény másodpercenként ~24-szer következik be, megfelelő működés esetén ilyen időközönként lenne szükség új képkockára.

6.2.3 Külső könyvtárak

A feladat elvégzésének megkönnyítése érdekében kliensoldalon is felhasználásra kerültek különböző külső csomagok, melyek ugyancsak szabadon felhasználható könyvtárak.

6.2.3.1 jQuery

Az én szempontomból nagyon hasznosnak bizonyult, mivel a stream megjelenítése során, egy HTML elemen belüli képet cserélgetek, melynek gyorsan végbe kell mennie.

6.2.3.2 Bootstrap

Fontosnak tartottam, hogy az elkészült weboldal megjelenítése segítse a kamera vezérlését akár a számítógépünkről akár a telefonunkról használnánk azt, így mindenképp egy olyan megoldást szerettem volna megvalósítani, ami reszponzívan jeleníti meg a weboldalakat.

Így a Bootstrap grid rendszerét én is használatba vettem, melyek segítségével a képeket megjelenítő oldalon a képernyő felbontásától függően változik az egy sorban megjelenített képek száma.

```

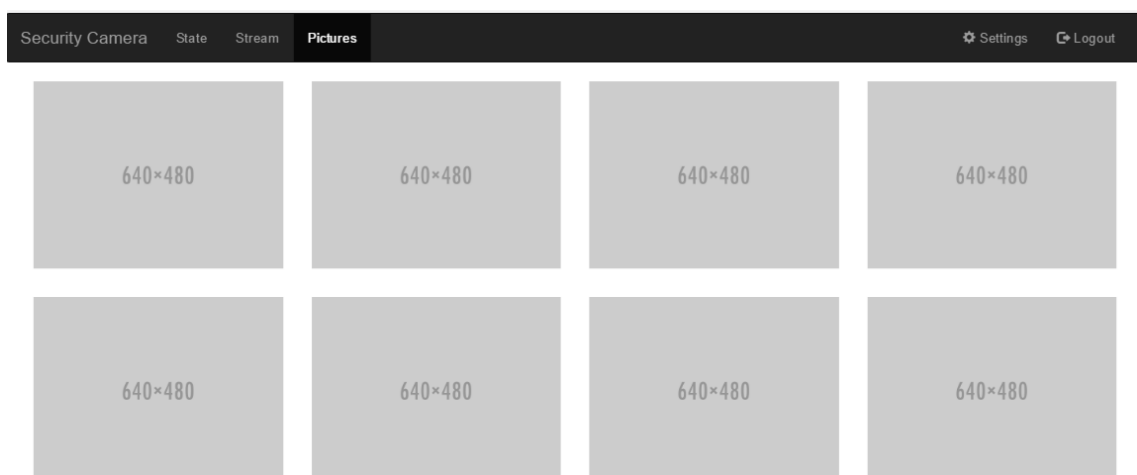
<div class="container">
  <div id="pictures" class="row">
    <div id="2016-11-28-22-16-34.jpg"
      class="col-lg-3 col-md-4 col-xs-6 center-block">
      <a href="public/pictures/2016-11-28-22-16-34.jpg">
        
      </a>
    </div>
    <div id="2016-11-28-22-16-33.jpg"
      class="col-lg-3 col-md-4 col-xs-6 center-block">
      <a href="public/pictures/2016-11-28-22-16-33.jpg">
        
      </a>
    </div>
    // további képek
    ...
  </div>
</div>

```

Mivel Bootstrap-ről van szó, így a megjelenítés szempontjából első sorban a classok számítanak. Jól látható, hogy a containeren belül minden egy sorban foglal helyet (row). Így a sorok tördelését a Bootstrapre bizzuk. Természetesen a különböző felbontások kategóriáira teszünk megkötéseket.

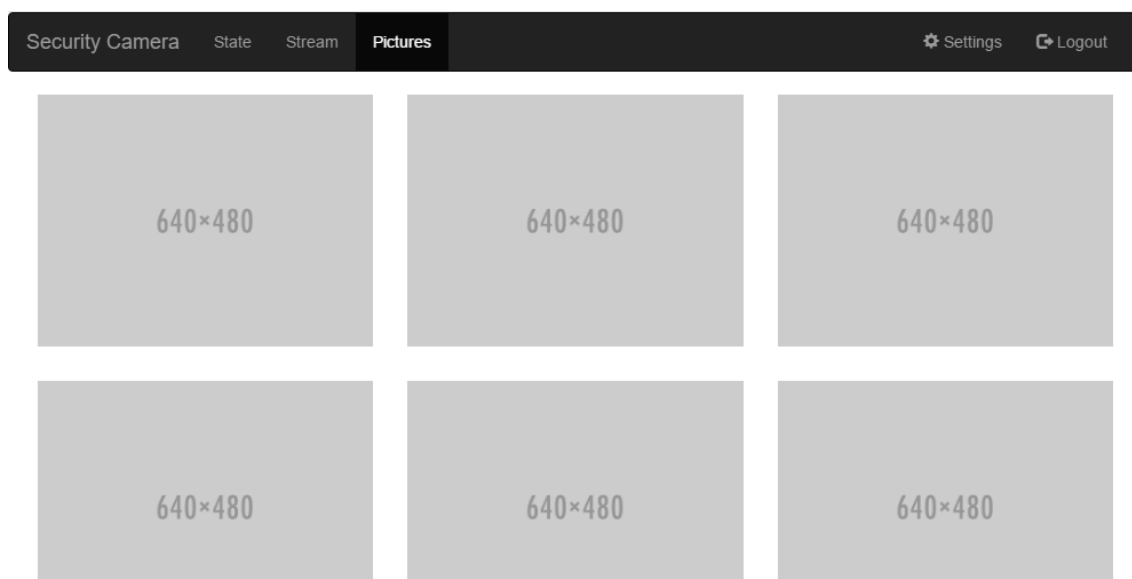
Ahogy az a fenti kódrészletből látszik, minden egyes képet tároló div megkapja a col-lg-3 col-md-4 col-xs-6 classokat melyek alapján a képek a következőképp fognak különböző felbontások esetén megjelenni:

- lg méret esetén: egy kép a képernyő 3/12-ét teszi ki
-> 4 kép fog egy sorban megjelenni



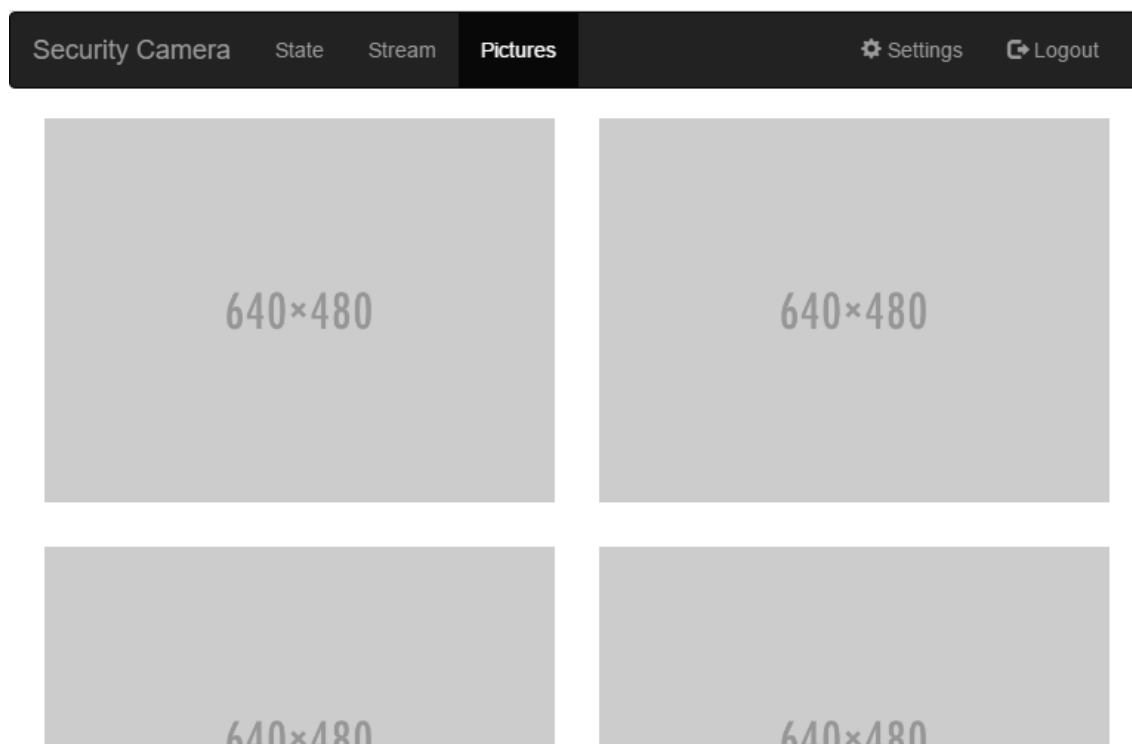
6.7. ábra – Korábban készült képek listája (soronként 4 kép)

- md méret esetén: egy kép a képernyő 4/12-ét teszi ki
-> 3 kép fog egy sorban megjelenni



6.8. ábra – Korábban készült képek listája (soronként 3 kép)

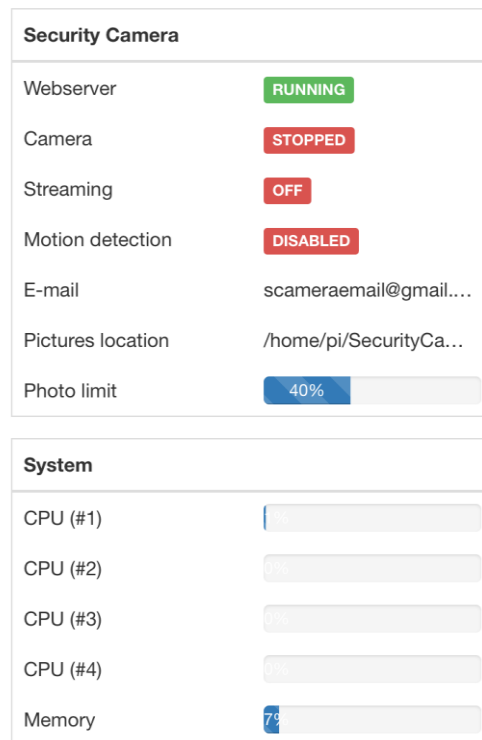
- xs méret esetén: egy kép a képernyő 6/12-ét teszi ki
-> 2 kép fog egy sorban megjelenni



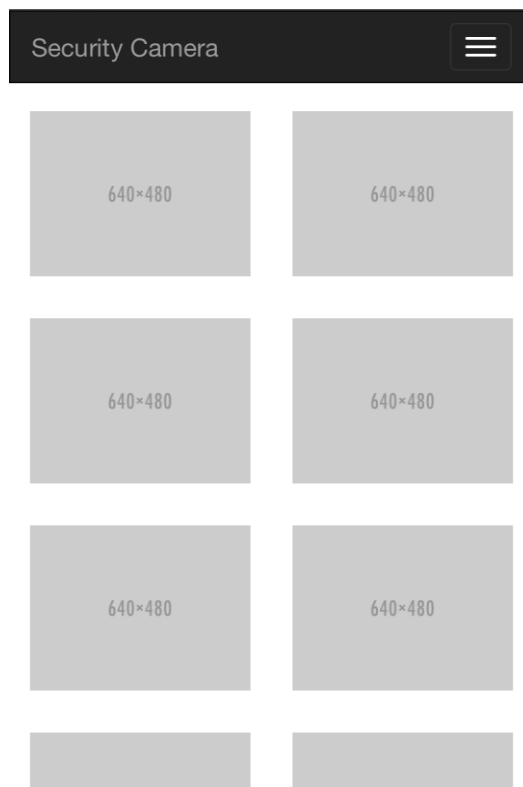
6.9. ábra – Korábban készült képek listája (soronként 2 kép)

Mindenképp érdemes még szót ejteni az általam is felhasznált img-responsive classról, ezzel a classal ugyanis egy kép is reszponzívvá tehető, mely azt fogja eredményezni, hogy amennyiben a képernyő szélessége már kisebb, mint maga a kép, a CSS úgy fogja alakítani a képet, hogy az még pont beleférjen az ablakba. Így amikor a Pictures oldalon tovább csökkentjük a felbontást, a képek szélteben továbbra is kitöltik a képernyőt.

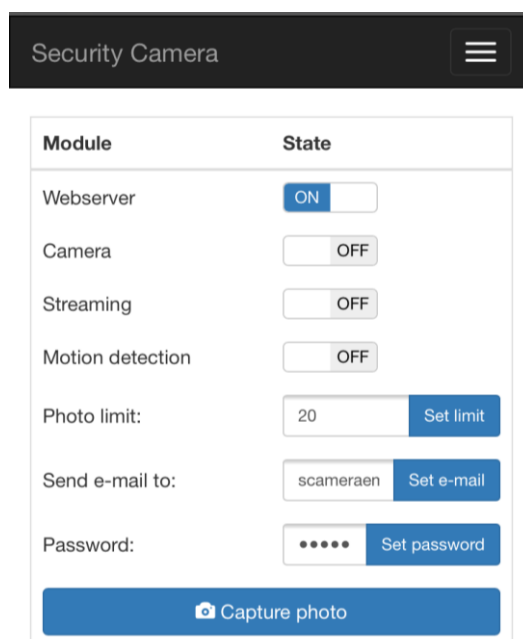
A Bootstrapet felhasználva, a következőképpen jelennek meg az oldalak egy okostelefonon:



6.10. ábra – State oldal (okostelefonról)



6.11. ábra – Pictures oldal (okostelefonról)



6.12. ábra – Settings oldal (okostelefonról)

6.3 Egyéb fájlok

Az elkészült Java alkalmazás relatíve sokféle funkciójának megvalósítását jónéhány külső fájl segíti, amik nem kerültek bele a lefordított és betömörített jar fájlba.

6.3.1 content mappa

A statikus HTML állományok könyvtára, a megfelelő URL-en közvetlenül elérhetőek, nem tartalmazznak olyan adatokat, amik az alkalmazás biztonságát veszélyeztetnék.

6.3.2 public mappa

A public könyvtár tartalmazza az összes kliensoldali JavaScript és CSS fájlt, melyek garantálják az webes felület helyes megjelenését és funkcionalitását. Ezen fájlok - melyek az internetről bármikor letölthető nyílt forráskódú fájlok - azért kerültek bele az alkalmazásba, hogy az elkészült megoldás használatához ne kelljen állandó internetes kapcsolat.

Ugyancsak ebbe a mappába kerülnek az eszköz használata során készült képek (hacsak át nem állítjuk a képek mentési helyét az alkalmazás elindítása során).

6.3.3 conf.ini

Az alkalmazás utolsó konzisztens állapotának beállításait tárolja egy JSON-ben. Erre azért van szükség, hogy amikor újraindítjuk az alkalmazást, olyan beállításokkal induljon, mint amilyenekkel megszakadt a futása. Ugyancsak ebben található az aktuális belépési jelszó, mely a fájlba írás előtt SHA-256 hashelésre kerül, hogy amennyiben valaki hozzájut a fájl tartalmához, akkor sem tudja visszafejteni az eredeti jelszót.

Amennyiben a fájl törlésre kerül, az alkalmazás az első elindítása során újragenerálja az eredeti beállításokkal.

6.3.4 .dll és .so

A különféle külső könyvtárak megkövetelik az alkalmazás indítása során dinamikusan betöltött fájlokat.

6.3.4.1 OpenCV

Az OpenCV fájljai az *opencv_java310.dll* és a *libopencv_java310.so*. Ezen két fájl segítségével a Camera modul képes bármilyen Windows-os és Linux-os környezetben elindulni és működni.

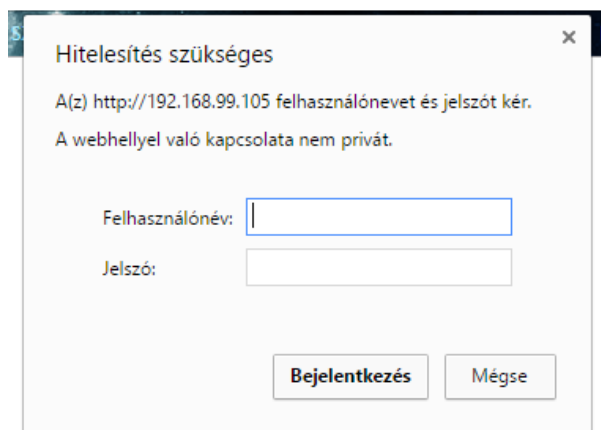
6.3.4.2 Sigar

A Sigar esetén valamivel komplikáltabb a helyzet, ugyanis a rendszeradatok elkérése nem csak operációs rendszer függő, hanem függ az adott eszköz processzorának architektúrájától is.

A *sigar-amd64-winnt.dll* és a *libsigar-arm-linux.so* fájlok garantálják a SystemInformationGatherer elindulását és működését bármilyen 64 bites AMD architektúrájú processzorral felszerelt Windows-os rendszeren, és bármilyen ARM architektúrájú processzorral felszerelt Linux-os rendszeren. Ez a két kategória már lefedte az általam használt eszközöket (Laptop, Raspberry Pi), viszont egy ettől ezektől eltérő konstrukció esetén nem fog elindulni az alkalmazás. A probléma ilyenkor könnyen orvosolható a megfelelő dll/so bemásolásával az alkalmazás könyvtárába.

7 Az alkalmazás használta

Az alkalmazás sikeres elindítása után (lsd. Üzembe helyezés), amennyiben még nem állítottunk be új jelszót, az alapértelmezett jelszóval tudunk belépni az admin fiókba (felhasználó név: admin, jelszó: admin).



Hitelesítés szükséges

A(z) http://192.168.99.105 felhasználónevet és jelszót kér.
A webhellyel való kapcsolata nem privát.

Felhasználónév:

Jelszó:

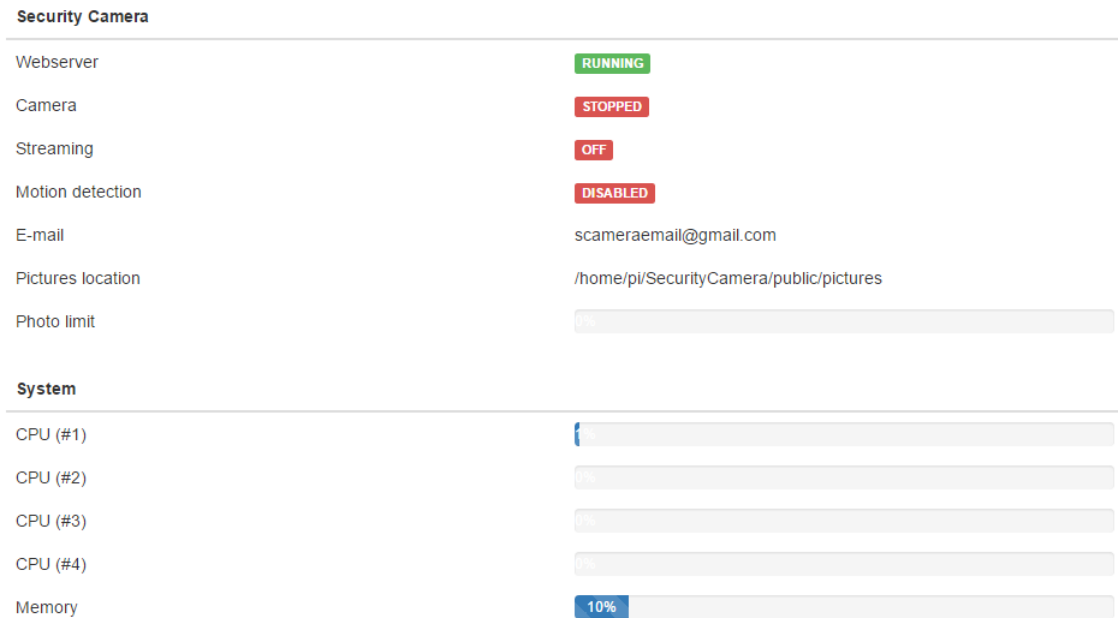
Bejelentkezés Mégse

7.1. ábra – Basic autentikáció: Hitelesítés szükséges popup

A sikerest bejelentkezést követően választhatunk a menüpontok közül.

7.1 State

Az alkalmazás kezdőoldala, itt kerülnek megjelenítésre az alkalmazás a felhasznált eszköz adatai (a különböző funkciók állapotai, processzor és memória használat... stb.).



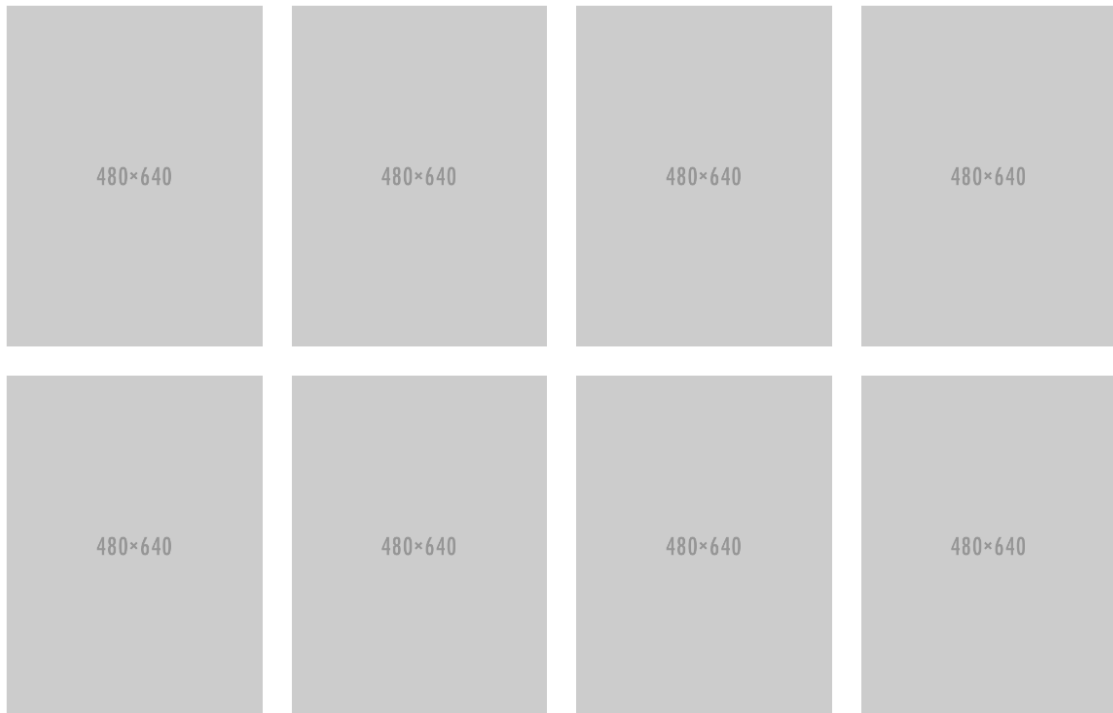
7.2. ábra – State oldal

7.2 Stream

Amennyiben be van kapcsolva a stream funkció, ezen az oldal tudjuk a stream élőképét megtekinteni.

7.3 Pictures

Az eddig rögzített képeket tekinthetjük meg itt. A képekre kattintva megjelenik eredeti méretben is.



7.3. ábra – Pictures oldal

7.4 Settings

Az alkalmazás összes beállítását innen tudjuk megtenni.

Module	State
Webserver	<input checked="" type="checkbox"/> ON
Camera	<input type="checkbox"/> OFF
Streaming	<input type="checkbox"/> OFF
Motion detection	<input type="checkbox"/> OFF
Photo limit:	<input type="text" value="20"/> <input type="button" value="Set limit"/>
Send e-mail to:	<input type="text" value="scameraemail@gmail.com"/> <input type="button" value="Set e-mail"/>
Password:	<input type="password" value="****"/> <input type="button" value="Set password"/>

7.4. ábra – Settings oldal

A *Webserver* leállításával letiltásra kerül a 80-as porton történő kommunikáció, de a kamera továbbra is működik az utolsó beállításoknak megfelelően.

A *Camera* modul kapcsolójának használatával ki és be tudjuk kapcsolni a kamera használatot.

A *Streaming* kapcsolóval a kamera real-time képének megjelenítését tudjuk ki-be kapcsolni.

A *Motion detection* bekapcsolása esetén a kamera e-mailt küld és képet készít mozgás esetén, viszont egyre ritkábban, azért hogy ne kerüljön elküldésre rengeteg üzenet, amikor valaki elhalad a kamera előtt. Az eredeti képkészítési időköz visszaállításra kerül a mozgásérzékelés vagy a kamera ki és bekapcsolásával.

A *Photo limit* mező segítségével beállíthatjuk, hogy mennyi képet tároljon maximálisan az eszköz. Amennyiben a képek száma eléri ezt az értéket, az új kép létrehozása során törlésre kerül a legrégebbi kép.

A *Send e-mail to* mező módosításával beállíthatjuk azt az e-mail címet, mely mozgás esetén értesítésre kerül.

A *Password* mező a bejelentkezett felhasználó jelszavának módosítására szolgál.

A *Capture photo* lenyomása esetén a kamera egy képet készít az aktuálisan megfigyelt területről.

7.5 Logout

A logout menüpont segítségével kijelentkeztetésre kerül a felhasználó.

8 Értékelés

Végeredményben úgy vélem az alkalmazás teljes mértékben megfelel a feladatkiírásnak, hiszen:

- hardveresen kizárólag egy Raspberry Pi-re és egy kamerára van szükség,
- szoftveresen kizárólag nyílt forráskódú alkalmazásokat használ fel,
- az egész projekt Java nyelven íródott,
- elkészült egy reszponzív webes felületet, melyen keresztül lehetőségünk van a kamera vezérlésére,
- lehetőségünk van kép készítésére és a kamera video streamjének megtekintésére.

8.1 Teljesítmény tesztek

Igyekeztem az alkalmazást a lehetőségeimhez mérten optimalizálni, ezzel is növelve az alkalmazás használatának élményét. A teljesítmény tesztek mind Raspberry Pi-ről (Raspberry Pi 2 Model B : 900MHz quad-core ARM Cortex-A7 CPU, 1GB RAM) futtatott webszerverrel készültek, hiszen a rendelkezésemre álló teszteszközök közül ez volt a leggyengébb hardver.

8.1.1 Erőforrás felhasználás

Átlagos felhasználás (streamelés és 1-2 felhasználó nézi is azt) esetén az alábbiak szerint néz ki az alkalmazás erőforrás felhasználása (Raspberry Pi 2 Model B esetén százalékos arányban):

- ~900 MHz processzor felhasználás (25%)
- ~60 MB memória fel (6,5%)

Az alkalmazás használta során (minimális egyéb alkalmazás futtatása közben) az erőforrás felhasználás az alábbiak szerint alakult:



8.1. ábra – htop: Az alkalmazás által felhasznált erőforrások

8.1.2 Terhelés teszt

Teljesítmény szempontjából mindenképp fontos volt meghatározni, hogy mennyi az maximális felhasználó szám, ami esetén még megfelelően működik az alkalmazás.

A tesztelés alapján még 6 fő stream nézése esetén is megfelelő volt az alkalmazás működése, és mivel nem céges felhasználásra készült, így úgy gondolom ez egy elfogadható eredmény.

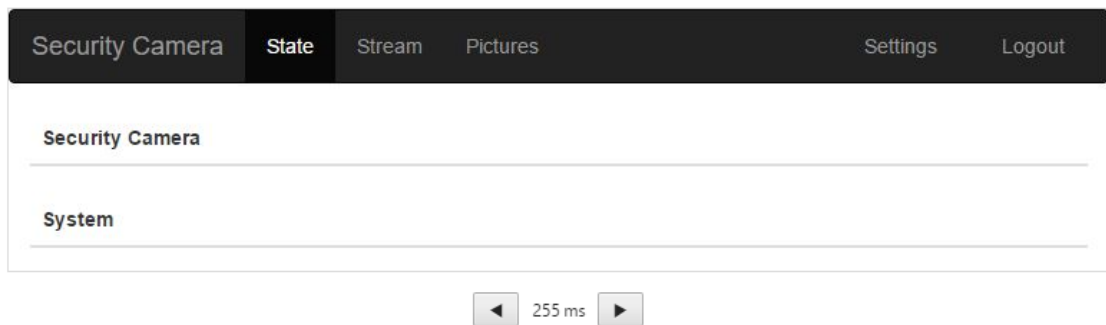
8.1.3 Webes felület

8.1.3.1 Oldal betöltés

Az oldalbetöltés idejének mérését a Chrome böngésző beépített fejlesztői eszközeivel végeztem. Az oldal betöltésének mérése során három fontos időpillanatot különböztettem meg:

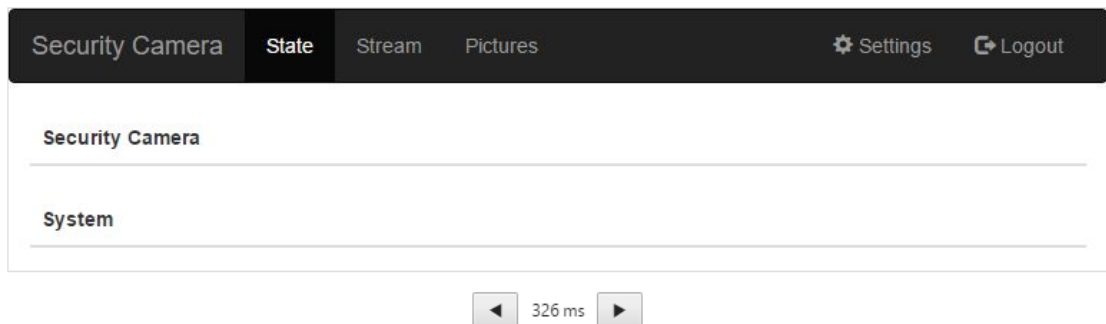
1. megjelenik a statikus oldal
2. betöltésre kerül a Bootstrap által használt betűkészlet
3. megjelennek a dinamikus adatok

A statikus oldal a betöltés megkezdése után 255 ms-al később jelent meg.



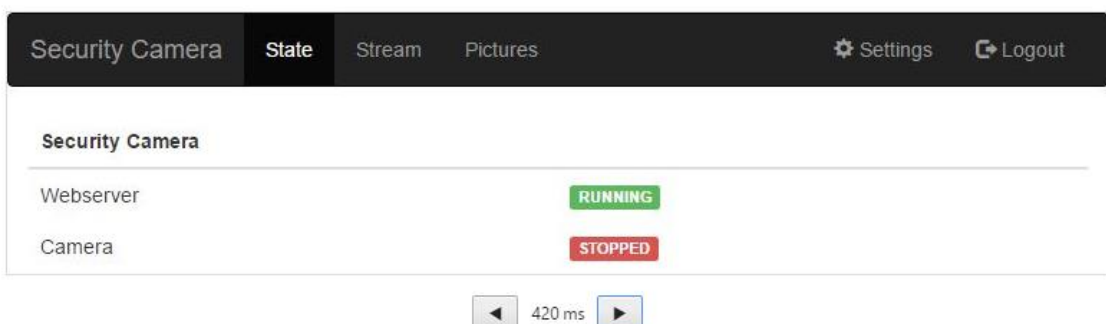
8.2. ábra – Megjelenik a statikus oldal (255 ms)

Az oldalhoz tartozó betűtípusnak megfelelő elemek, az oldal betöltésének megkezdése után 326 ms-al később jelentek meg.



8.3. ábra – Betöltésre kerül a font (326 ms)

Az oldal teljes (dinamikus adatokkal feltöltött) tartalma, az oldal betöltésének megkezdését követően 420 ms-al később jelent meg.



8.4. ábra – Megjelennek a dinamikusan betöltött adatok (420 ms)

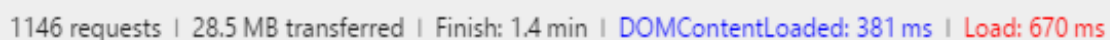
Ezeket a tesztek többszöri megismétlése után is hasonló eredményre jutottam, így az értékek átlagosnak tekinthetők.

Összességében elmondható, hogy 420 ms-os betöltési idővel még bőven az elfogadható kategóriába tartozik az alkalmazás.

8.1.3.2 Adatforgalom

Az adatforgalom tekintetében a streamelés funkció vizsgálatát választottam, hiszen a képek folyamatos letöltése során generálódik a legnagyobb forgalom.

Jól látszik, hogy ~1,4 percnyi stream nézés során több mint 1000 HTTP kérés ment a szerver felé, illetve 28,5 MB adatforgalom generálódott.



1146 requests | 28.5 MB transferred | Finish: 1.4 min | DOMContentLoaded: 381 ms | Load: 670 ms

8.5. ábra – Stream által generált forgalom

Ez az eredmény még nem a legtokéletesebb, de jó alapja lehet egy stabil alkalmazásnak. A későbbiekben csökkenteni lehetne a kérések számát, a csomagok méretének növelésével, viszont így valamilyen szinten megnőne a késleltetés.

Amennyiben tömörítenénk a képek méretét, vagy csak a változásokat küldenénk, az egyértelműen javítaná a teljesítményt. Ebben az esetben, egy olyan helyiségben, ahol nincs mozgás (és a kamera az esetek nagy részében ilyen területet figyel), minimálisra csökkenthetnénk az adatforgalmat. Természetesen az is nagyban növelné a sebességet, ha a TCP protokoll helyett egy „streamelésre alkalmasabb” protokollt használnánk (UDP/RTP).

8.2 Továbbfejlesztési lehetőségek

8.2.1 Jelenlegi funkciók optimalizálása

Az általam készített megoldás nem tökéletes, még bőven van mit finomítani rajta, többek között:

- biztonság: biztonsági kameráról lévén szó, ez egy olyan sarkalatos szempont, amit sosem lehet tökéletesre fejleszteni (HTTPS, fejlettebb autentikáció)
- stream: jelenleg képek egymás után történő megjelenítése, amit lehetne sokkal hatékonyabban is megoldani, például rövid videó részletek vagy a képkockák közti differenciák küldésével
- kinézet: a megjelenítés jelenleg elég minimalista, lehetne ennél sokkal szebb, összetettebb kinézetet is varázsolni az alkalmazásnak

8.2.2 További funkciók implementálása

Az OpenCV függvénykönyvtár már most is a része az alkalmazásnak, melynek a lehetőségei nem lettek túlzottan kihasználva. Rengeteg egyéb funkcióját lehetne a biztonsági kamera szolgálatába állítani:

- alakzatok, élőlények felismerése: nem szeretnénk, ha az udvarunk megfigyelése esetén, az utcán járó autókat, a lakásunk megfigyelése esetén a saját kutyánkat, mozgásnak érzékelné a kamera
- személyek felismerése: állandóan működő kamera esetén kikapcsolhatna a kamera, ha megérkezünk
- egyéb webes szolgáltatások biztosítása (pl. FTP server a képek eléréséhez)

De ahogy szoftveresen, úgy hardveresen is további funkciókkal bővíthetnénk az az eszköz funkcionalitását:

- szünetmentes táp: áramszünet esetén nem kapcsolna ki a kamera
- szervó motoros kamera állvány: a webes felületről szabályozhatnánk a kamera által megfigyelt területet (természetesen csak bizonyos határokon belül)
- infra kamera: sötétben is lenne lehetősége a kamerának mozgás detektálásra

8.3 Tapasztalatok

Mindent egybe vetve úgy érzem, hogy ez a szakdolgozat téma remek lehetőséget nyújtott új technológiák megismerésére, így mindenképpen rengeteg hasznos tudással gazdagodtam a feladat megoldása során.

9 Üzembe helyezés

A tesztelés során egy Raspberry Pi 2 Model B (Raspbian) állt rendelkezésemre, illetve egy ennél nagyságrendekkel nagyobb teljesítményű laptop (Windows 10 64-bit), így biztosan mondhatom, hogy ezeken az eszközökön biztosan megfelelően fut. Bizonyos követelményeket viszont csak a tesztek alapján becsültem.

9.1 Követelmények

9.1.1 Hardveres

- 1 GHz-es processzor (egymagos processzor esetén)
- 100 MB memória
- egy, az eszközhöz csatlakoztatott kamera

9.1.2 Szoftveres

- Java Runtime Environment 1.8-as verziója (a futtatáshoz)
- (Opcionális) Java Development Kit 1.8-as verziója (a fordításhoz)
- tetszőleges operációs rendszer, mely a fentebb említett Java környezetet képes futtatni

9.1.3 Raspberry Pi esetén

A már említett hardveres és szoftveres követelményeken túl Raspberry Pi esetén szükség van egy kernel modul betöltésére (amennyiben ez még nem történt meg) minden egyes újraindítás után.

Ennek a modulnak a betöltését az alábbi parancs kiadásával tudjuk megtenni:

```
modprobe bcm2835-v4l2
```

(Megjegyzés: a parancs futtatásához rendszergazdai jogosultságra lehet szükség)

9.1.4 Alkalmas eszközök

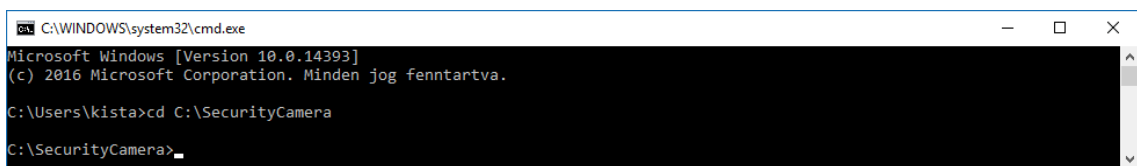
Raspberry Pi 2-es változatától kezdve felfelé bármelyik Raspberry Pi modell tökéletesen megfelel a fenti specifikációknak, úgy ahogy a manapság kapható személyi számítógépek bármelyike is.

9.2 Az alkalmazás elindítása

Az alkalmazás alapvetően Raspbiant futtató Raspberry Pi miniszámítógéphez készült, viszont fel lett készítve más operációs rendszereken és eszközökön történő futtatás, ahogy az az előző fejezetekből ki is derült.

A futtatás lépései minden támogatott rendszeren és eszközön hasonló. Célszerű az adott operációs rendszer alapértelmezett parancssorát használni (Command Line/Bash/Terminal), így látni fogjuk a program kimenetét (logját).

1. Az alkalmazás fordítása és szükséges fájlok letöltése.
VAGY
Előre összeállított csomag letöltése
(<https://github.com/kistamas00/SecurityCamera/raw/master/SecurityCamera/SecurityCamera.zip>).
2. Szükség esetén tetszőleges helyre kicsomagoljuk az alkalmazást.
3. Parancssorból elnavigálunk az alkalmazás mappájába (cd parancs).

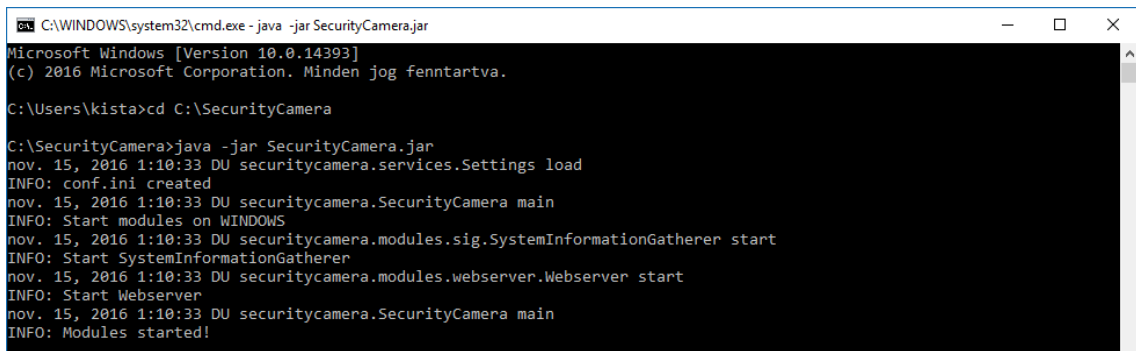


9.1. ábra – Mappák közti navigálás (cd <mappa>)

4. A feltelepített java alkalmazás segítségével futtatjuk a SecurityCamera.jar nevű fájlt, az alábbi parancs kiadásával (Megjegyzés: Az elindítás során további paraméterek is megadhatóak, ezek listája a következő fejezetben olvasható.):

```
java -jar SecurityCamera.jar
```

(Megjegyzés: a parancs futtatásához rendszergazdai jogosultságra lehet szükség)



```
C:\WINDOWS\system32\cmd.exe - java -jar SecurityCamera.jar
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. Minden jog fenntartva.

C:\Users\kista>cd C:\SecurityCamera

C:\SecurityCamera>java -jar SecurityCamera.jar
nov. 15, 2016 1:10:33 DU securitycamera.services.Settings load
INFO: conf.ini created
nov. 15, 2016 1:10:33 DU securitycamera.SecurityCamera main
INFO: Start modules on WINDOWS
nov. 15, 2016 1:10:33 DU securitycamera.modules.sig.SystemInformationGatherer start
INFO: Start SystemInformationGatherer
nov. 15, 2016 1:10:33 DU securitycamera.modules.webserver.Webserver start
INFO: Start Webserver
nov. 15, 2016 1:10:33 DU securitycamera.SecurityCamera main
INFO: Modules started!
```

9.2. ábra – Alkalmazás futtatása (java -jar SecurityCamera.jar)

Amennyiben nem kaptunk hibaüzenetet, az alkalmazás sikeresen elindult. A „Modules started!” felirat az összes modul elindulása után kerül kiírásra.

9.2.1 Paraméterek

Az elindítás során az alábbi paramétereket használhatjuk:

- --picture-path (vagy röviden -p): kiválaszthatjuk a képek tárolására használt mappát, így könnyedén menthetjük a fájljainkat külső tárolóra (Megjegyzés: relatív és abszolút útvonalat is elfogad)
- --log-file-name (vagy röviden -l): megadhatjuk, hogy milyen néven hozzon létre az alkalmazás egy log fájlt, melybe a konzolban megjelenő log elmentésre kerül (Megjegyzés: Ez csupán egy fájlnev és nem elérési út!)

Például az alábbi parancs kiadását követően az alkalmazás a készült képeket a C meghajtón lévő pictures mappába menti, illetve az alkalmazás mappájában létrehozásra kerül egy log.txt nevű fájl, melyben megtalálható lesz az alkalmazás által készített log:

```
java -jar SecurityCamera.jar -p C:\pictures -l log.txt
```

Ábrajegyzék

- 6.1. ábra – A megoldás során elkészített és felhasznált modulok
- 6.2. ábra – Camera modul
- 6.3. ábra – Webserver modul
- 6.4. ábra – SystemInformarionGatherer modul
- 6.5. ábra – A megoldás során elkészített és felhasznált szolgáltatások
- 6.6. ábra – Streamelés folyamatának állapotgépe
- 6.7. ábra – Korábban készült képek listája (soronként 4 kép)
- 6.8. ábra – Korábban készült képek listája (soronként 3 kép)
- 6.9. ábra – Korábban készült képek listája (soronként 2 kép)
- 6.10. ábra – State oldal (okostelefonról)
- 6.11. ábra – Pictures oldal (okostelefonról)
- 6.12. ábra – Settings oldal (okostelefonról)
- 7.1. ábra – Basic autentikáció: Hitelesítés szükséges popup
- 7.2. ábra – State oldal
- 7.3. ábra – Pictures oldal
- 7.4. ábra – Settings oldal
- 8.1. ábra – htop: Az alkalmazás által felhasznált erőforrások
- 8.2. ábra – Megjelenik a statikus oldal (255 ms)
- 8.3. ábra – Betöltésre kerül a font (326 ms)
- 8.4. ábra – Megjelennek a dinamikusan betöltött adatok (420 ms)
- 8.5. ábra – Stream által generált forgalom
- 9.1. ábra – Mappák közti navigálás (cd <mappa>)
- 9.2. ábra – Alkalmazás futtatása (java -jar SecuritiyCamera.jar)

Irodalomjegyzék

- [1] Raspberry Pi Security Camera : <https://pimylifeup.com/raspberry-pi-security-camera/>
- [2] Java Program – Security Camera : <https://www.youtube.com/watch?v=e-NbANRKZIk>
- [3] Webcam Capture : <http://webcam-capture.sarxos.pl/>
- [4] JRPiCam : <https://github.com/Hopding/JRPiCam>
- [5] OpenCV – About : <http://opencv.org/about.html>
- [6] Gson – GitHub : <https://github.com/google/gson>
- [7] SIGAR Wiki : <http://sigar.hyperic.com/>
- [8] JavaBeans Activation Framework – Wikipedia : https://en.wikipedia.org/wiki/JavaBeans_Activation_Framework
- [9] JavaMail – Wikipedia : <https://en.wikipedia.org/wiki/JavaMail>
- [10] jQuery : <https://jquery.com/>
- [11] Bootstrap – Wikipedia : [https://en.wikipedia.org/wiki/Bootstrap_\(front-end_framework\)](https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework))