



Universitatea Politehnica Timișoara  
Facultatea de Automatică și Calculatoare  
Departamentul Automatică și Informatică Aplicată



## MirrorView

Sistem de vizualizare a informațiilor personalizate  
pe un ecran integrat în oglindă

Conducător științific :

Ş. l. dr. ing. Sorin Nanu

Autor :

Mirea Andrei Sergiu Sorin

Lucrare de licență

Mirea Andrei Sergiu Sorin

## Cuprins

<b>TABEL AL FIGURILOR</b>	<b>4</b>
<b>1 INTRODUCERE</b>	<b>6</b>
1.1 DESCRIEREA TEMEI ŞI ÎNCADRAREA APLICAȚIEI ÎNTR-UN DOMENIU DE STUDIU	7
1.2 OBIECTIVELE LUCRĂRII ŞI SPECIFICAȚIILE DEZVOLTĂRII APLICAȚIEI	8
1.3 DESCRIERE SOLUȚII EXISTENTE PE PIAȚĂ	9
1.4 STRUCTURA PE CAPITOLE	11
<b>2 TEHNOLOGII UTILIZATE</b>	<b>12</b>
2.1 COMPONENTA HARDWARE	12
2.1.1 <i>Descrierea sistemului Raspberry Pi</i>	13
2.1.2 <i>Descrierea camerei Microsoft LifeCam HD-3000</i>	15
2.1.3 <i>Descrierea elementului de afișare</i>	16
2.2 COMPONENTA SOFTWARE	17
2.2.1 <i>Python</i>	17
2.2.2 <i>HyperText Markup Language, sau HTML, și Jinja</i>	20
2.2.3 <i>Limbajul de programare SQL</i>	20
2.2.4 <i>PyCharm Professional</i>	21
2.2.5 <i>SQLite</i>	22
2.2.6 <i>GitHub</i>	22
<b>3 IMPLEMENTAREA SOLUȚIEI</b>	<b>24</b>
3.1 IMPLEMENTAREA BAZEI DE DATE	25
3.2 IMPLEMENTAREA INTERFEȚEI GRAFICE ȘI A FUNCȚIONALITĂȚILOR ACESTEIA	26
3.2.1 <i>Colectarea de date</i>	29
3.2.2 <i>Managementul utilizatorilor</i>	33
3.2.3 <i>Conecțarea cu baza de date</i>	35
3.3 IMPLEMENTAREA SAITULUI DE CONFIGURARE	36
3.3.1 <i>Implementarea aplicației Python pentru crearea saitului de configurare</i>	37
3.3.2 <i>Structura codului sursă HTML în tandem cu Jinja</i>	40
3.3.3 <i>Imagini ale paginilor saitului</i>	43
<b>4 UTILIZAREA SISTEMULUI INFORMATIC DE AFIȘARE A INFORMAȚIILOR PERSONALIZATE PE UN ECRAN INTEGRAT ÎN OGINDĂ</b>	<b>46</b>
<b>5 CONCLUZII</b>	<b>48</b>
5.1 ÎMBUNĂTĂȚIRI VIITOARE	48
<b>BIBLIOGRAFIE</b>	<b>49</b>

## Tabel al figurilor

Nr crt.	Titlu	Sursă
1	Oglindă de obsidian din Anatolia	<a href="https://eaglesanddragonspublishing.com/wp-content/uploads/2015/05/Obsidian-Mirror-from-Anatolia.jpg">https://eaglesanddragonspublishing.com/wp-content/uploads/2015/05/Obsidian-Mirror-from-Anatolia.jpg</a>
2	Oglindă egipteană din bronz	<a href="http://biblicalisraeltours.com/wp-content/uploads/2016/12/Mirror-bronze.jpg">http://biblicalisraeltours.com/wp-content/uploads/2016/12/Mirror-bronze.jpg</a>
3	Oglindă renașcentistă	<a href="https://www.dorotheum.com/fileadmin/lot-images/40S90916/normal/repraesentativer_italienischer_renaissance_wandspiegel_5058254.jpg">https://www.dorotheum.com/fileadmin/lot-images/40S90916/normal/repraesentativer_italienischer_renaissance_wandspiegel_5058254.jpg</a>
4	Oglindă de la Evervue	<a href="https://www.evervue.co.uk/wp-content/uploads/MirrorVue-Standard-Single-Sink-300x300.jpg">https://www.evervue.co.uk/wp-content/uploads/MirrorVue-Standard-Single-Sink-300x300.jpg</a>
5	Oglindă de la Embrace	<a href="https://www.embracesmartmirror.com/wp-content/uploads/2017/12/embrace-touchscreen-smart-mirror">https://www.embracesmartmirror.com/wp-content/uploads/2017/12/embrace-touchscreen-smart-mirror</a>
6	Oglindă DIY	<a href="https://cdn.vox-cdn.com/thumbor/kLMrjzxchqCge3YL5hADU8nQrw=/0x0:1080x1080/920x0/filters:focal(0x0:1080x1080):no_upscale()/cdn.vox-cdn.com/uploads/chorus_asset/file/9063303/theverge_08162017_1923_001.jpg">https://cdn.vox-cdn.com/thumbor/kLMrjzxchqCge3YL5hADU8nQrw=/0x0:1080x1080/920x0/filters:focal(0x0:1080x1080):no_upscale()/cdn.vox-cdn.com/uploads/chorus_asset/file/9063303/theverge_08162017_1923_001.jpg</a>
7	Schemă bloc a Raspberry Pi 4B	<a href="https://www.raspberrypi.org/forums/viewtopic.php?t=271632">https://www.raspberrypi.org/forums/viewtopic.php?t=271632</a>
8	Raspberry Pi Zero	<a href="https://www.raspberrypi.org/homepage-9df4b/static/65b0d08aba609951b5a64529cc7f455/7fd5d/6b0defdbbf40792b64159ab8169d97162c380b2c_raspberry-pi-zero-1-1755x1080.jpg">https://www.raspberrypi.org/homepage-9df4b/static/65b0d08aba609951b5a64529cc7f455/7fd5d/6b0defdbbf40792b64159ab8169d97162c380b2c_raspberry-pi-zero-1-1755x1080.jpg</a>
9	Raspberry Pi 4B	<a href="https://www.raspberrypi.org/homepage-9df4b/static/f1682eef7da7e8d989662d147f48977c/7fd5d/f532739a-171e-4aa0-b9f3-d05e20710b69_raspberry-pi-4-model-b.jpg">https://www.raspberrypi.org/homepage-9df4b/static/f1682eef7da7e8d989662d147f48977c/7fd5d/f532739a-171e-4aa0-b9f3-d05e20710b69_raspberry-pi-4-model-b.jpg</a>
10	Raspberry Compute Module 3+	<a href="https://www.raspberrypi.org/homepage-9df4b/static/3cfce381bf7f590051e6c441bf4bea31c/7fd5d/0807fa6b937b11be2d3acc8eafeafda005b147a0c_cm304.jpg">https://www.raspberrypi.org/homepage-9df4b/static/3cfce381bf7f590051e6c441bf4bea31c/7fd5d/0807fa6b937b11be2d3acc8eafeafda005b147a0c_cm304.jpg</a>
11	Microsoft LifeCam HD-3000	<a href="https://m.media-amazon.com/images/S/aplus-media/mg/3e8dd051-e594-48bf-a78a-f4daf15b77b3.jpg">https://m.media-amazon.com/images/S/aplus-media/mg/3e8dd051-e594-48bf-a78a-f4daf15b77b3.jpg</a>
12	Samsung SyncMaster XL2370HD	<a href="https://images.samsung.com/is/image/samsung/de_LS23ELDKF-EN_001_Front?\$L2-Thumbnail\$">https://images.samsung.com/is/image/samsung/de_LS23ELDKF-EN_001_Front?\$L2-Thumbnail\$</a>
13	Apple MacBook Pro 13" 2017	<a href="https://mymarket.live/wp-content/uploads/2019/02/apple-macbook-pro-13-inch-2016-1892-045.jpg">https://mymarket.live/wp-content/uploads/2019/02/apple-macbook-pro-13-inch-2016-1892-045.jpg</a>
14	Interpretatorul Python afișând Zen of Python	Poză de ecran proprie
15	Exemplu de algoritm scris în Python	Poză de ecran proprie
16	Interfața pentru a crea un nou proiect din PyCharm	Poză de ecran proprie
17	Interfața web GitHub	Poză de ecran proprie
18	Structura de directoare și fișiere a proiectului	Poză de ecran proprie
19	Scheletul interfeței Tk.	Poză de ecran proprie
20	Interfața propriu-zisă	Poză de ecran proprie
21	Implementarea clasei Newsletter	Poză de ecran proprie
22	Constructorul clasei Window	Poză de ecran proprie
23	Funcția update_tk()	Poză de ecran proprie
24	Modul email	Poză de ecran proprie
25	Modul calendar	Poză de ecran proprie

26	Funcția de preluare a emailurilor	Poză de ecran proprie
27	Funcția de preluare a evenimentelor de afișat	Poză de ecran proprie
28	Funcția de preluare a stării atmosferice	Poză de ecran proprie
29	Funcția de preluare a știrilor	Poză de ecran proprie
30	Clasa News utilizată în modulul de știri	Poză de ecran proprie
31	Funcția encode_pictures()	Poză de ecran proprie
32	Funcția de recunoaștere a fețelor	Poză de ecran proprie
33	Clasa Database	Poză de ecran proprie
34	Clasele ce mînează tabelele din baza de date	Poză de ecran proprie
35	Utilizarea metodelor disponibile pentru aflarea sau setarea utilizatorului activ	Poză de ecran proprie
36	Harta saitului	Poză de ecran proprie
37	Pașii de configurare ai obiectului Flask	Poză de ecran proprie
38	Apelarea metodei run()	Poză de ecran proprie
39	Clasa utilizată pentru configurarea suplimentară	Poză de ecran proprie
40	Implementarea funcționalităților paginii de administrare	Poză de ecran proprie
41	Implementarea claselor utilizate pentru formulare	Poză de ecran proprie
42	Funcția căii "/google" denumită intuitiv google()	Poză de ecran proprie
43	Modelul de bază al saitului	Poză de ecran proprie
44	Modelul folosit pentru pagina de autentificare	Poză de ecran proprie
45	Pagina de autentificare	Poză de ecran proprie
46	Pagina de înregistrare	Poză de ecran proprie
47	Pagina de recuperare a parolei uitate	Poză de ecran proprie
48	Pagina de configurare, varianta la prima autentificare a unui utilizator nou	Poză de ecran proprie
49	Pagina de schimbare a parolei	Poză de ecran proprie
50	Interfața grafică al contului "Default"	Poză de ecran proprie

## 1 Introducere

Sticla se află în societatea umană de la începuturile acesteia, mai exact din Epoca de Piatră. În această perioadă, aceasta era găsită în natură sub formă de obsidian (sticlă opacă vulcanică cu un mare conținut de siliciu). Omul se folosea de aceasta pentru a își confecționa obiecte ascuțite.

Odată cu dezvoltarea tehnologiei de șlefuire, societatea umană a început să folosească obsidianul pentru a crea oglinzi într-o formă cunoscută societății moderne. Prima de acest fel a fost găsită de către arheologi în Anatolia (Turcia modernă) și datată în jurul anului 6000 î. Hr.. [1]



*Fig. 1 Oglindă de obsidian din Anatolia*



*Fig. 2 Oglindă egipteană din bronz*

De-a lungul timpului au existat și versiuni de metal ale unei oglinzi. Acestea erau confecționate dintr-o bucată de metal (cupru, bronz sau argint) care era șlefuită până se obținea gradul de reflecție dorit.

Oglinda "modernă", cea cu care este obișnuit omul contemporan, a apărut la începutul perioadei Renașterii odată cu dezvoltarea tehnologiei de "suflare" a sticlei. [2]



*Fig. 3 Oglindă renașcentistă*

## 1.1 Descrierea temei și încadrarea aplicației într-un domeniu de studiu

Tema propusă în această lucrare este menită să eficientizeze procesul de management al timpului personal. În perioada actuală, timpul a devenit o resursă destul de costisitoare prin urmare o digitalizare a oglinziei reduce din timpul irosit de accesarea diferitelor platforme utilizate cum ar fi calendar personal sau email.

Conceptul de oglindă inteligentă este unul recent apărut în urma lansării pe piață a unor tehnologii (sisteme cu microprocesor) mult mai puțin costisitoare. În cazul celei prezentate în această lucrare este folosit un sistem cu microprocesor, un ecran oarecare și o cameră web de calitate medie.

Lucrarea prezentată în detaliu de-a lungul următoarelor capitole presupune un sistem cu ecranul integrat într-o oglindă de vizualizare a informațiilor calendarului și emailului Google personal, starea vremii curente, știri dintr-un domeniu ales de utilizator și ceasul alături de dată. Informațiile afișate pe ecran se vor schimba dinamic în funcție de cine se află în față ei utilizând un senzor optic pentru recunoaștere facială. Configurarea acesteia se realizează cu ajutorul unui site găzduit pe un server local.

Conceptul de sistem cu microprocesor este unul destul de bine cunoscut în domeniul tehnologiei, acesta stând la baza multor echipamente folosite în viața de zi cu zi, precum mașina sau telefonul intelligent. Asemenea sisteme au fost dezvoltate încă din anii 1960, dar foarte limitate din punct de vedere al capacitatii de procesare. [3] Pentru a se ajunge la nivelul din anilor 2010 a fost nevoie de reducerea costurilor producției acestora și de micșorarea ca dimensiune fizică a componentelor din care acesta este alcătuit.

Tranzistorul este unul din componentele de bază care a influențat creșterea capacitatii de procesare. Modul în care acesta influențează performanța unui sistem este simplu, adică cu cât se integrează mai mulți într-un sistem cu atât acesta este mai performant. Chiar dacă în acest paragraf este prezentat simplist principiul, acesta de fapt este mai complex depinzând de frecvența tactului, tipul instrucțiunilor utilizate, modul de accesare al resurselor externe și.a.m.d.. [4]

Această tendință de digitalizare a obiectelor din viața de zi cu zi a prins amploare în ultimii 15 - 20 de ani. Internet of Things, sau IoT, este numele acestei tendințe. Cu această lucrare mi-am propus să dezvolt o aplicație din acest domeniu vast.

IoT este un domeniu emergent caracterizat prin complexitatea, nu neapărat a dispozitivelor individuale, ci a interconectării acestora cu serviciile deja existente și folosite, precum ecosistemele oferite de Apple, Google sau Microsoft. Momentan pe piață se află o multitudine de produse IoT precum frigidere, becuri, aparate de cafea, cuptoare cu microunde, periute de dinți, dispozitive de climatizare și.a.m.d.. [5]

Privind din exterior această dezvoltare rapidă a tendințelor tehnologice și conceptuale, motivația acestei lucrări este una simplă dar în dezvoltarea acesteia am întâmpinat o multitudine de provocări. Motivația provine din dorința de a-mi dezvolta capacitatele de dezvoltare software utilizând limbajul de programare Python în vederea obținerii unui produs IoT.

## 1.2 Obiectivele lucrării și specificațiile dezvoltării aplicației

În vederea realizării lucrării următoarele etape au fost necesare :

- documentarea privind mediului de programare cel mai bine aplicat lucrării dezvoltate;
- documentarea privind limbajul de programare Python cum ar fi reguli de sintaxă, metode de implementare a modulelor software, ușurința integrării a altor module terțe în proiect;
- stabilirea sistemului cu microprocesor folosit și a posibilităților de conectare a componentelor hardware externe cu acesta;
- realizarea modulelor software necesare și integrarea modulelor terțe folosite;

Pentru a putea realiza obiectivele prezentate mai sus au fost folosite următoarele :

- Raspberry Pi 4B;
- alimentator pentru Raspberry Pi 4B;
- cablu HDMI;
- adaptor microHDMI - HDMI;
- Microsoft LifeCam HD-3000;
- card MicroSD - 32GB;
- ecran oarecare cu conexiune HDMI;
- Python - limbaj de programare;

- Google API;
- SQLite;
- SQL - limbaj de programare;
- HTML - limbaj de programare;
- Jinja - limbaj de programare;
- PyCharm Professional 2020.1;
- GitHub;
- Thonny;

### 1.3 Descriere soluții existente pe piață

Având în vedere faptul că acest produs face parte dintr-un domeniu relativ nou, soluțiile ca produs finit existente pe piață sunt reduse la număr, cele mai multe fiind de tipul "Do-It-Yourself", mai precis sub formă de instrucțiuni de construcție. Există ca soluție produsul celor de la Evervue care oferă o varietate mare de posibilități în dimensiuni ale oglinzi sau a locului de folosire (baie sau alte camere cu umiditate redusă). Oglinzile oferite de această companie sunt pentru a fi utilizate în mediul afacerilor, turism sau uz casnic conform saitului companiei. [6]

Altă companie care oferă oglinzi inteligente este Embrace. Produsul acestora este mai apropiat de conceptul prezentat în această lucrare. Avantajul acestuia este că se folosește un dispozitiv cu Android pentru a afișa ceea ce dorește utilizatorul și faptul că are ecran tactil pentru control. Folosirea sistemului de operare Android oferă posibilitatea instalării oricărei aplicații disponibile pe Play Store<sup>1</sup>, unul din punctele forte conform specificațiilor producătorului. [7]

Din punct de vedere al prețului cele două produse existente pe piață sunt asemănătoare. Ambele au prețul aproximativ de 1000 EUR, însă cum cei de la Evervue au o gamă vastă de posibilități se poate ajunge și la produse de peste 5000 EUR.

Variantele cele mai multe la număr sunt de tipul "Do-It-Yourself"<sup>2</sup> (DIY) unde dezvoltatorii independenți au creat aplicațiile de rulat pe un dispozitiv anume (de obicei laptop

---

<sup>1</sup> Magazin oferit de Google pentru achiziția de aplicații pentru dispozitivele cu Android.

<sup>2</sup> Tradus mot a mot "Fă singur".

reutilizat, Raspberry Pi sau alte sisteme asemănătoare). Aceste proiecte DIY vin cu un set de instrucțiuni și cerințe hardware. Avantajul acestora este faptul că utilizatorul are posibilitatea de a folosi ce materiale dorește atât timp cât se încadrează în cerințele proiectului, prin urmare costul final poate fi mult mai mic decât al celor două companii prezentate anterior.

Proiectul prezentat în această lucrare se încadrează în varianta DIY, permitând oricărui om care are un display nefolosit, un Raspberry Pi și un senzor optic (de ex. o cameră web). Modul cum se poate integra într-o oglindă fiind lăsat la libera alegere a utilizatorului.



Fig. 4 Oglindă de la Evervue

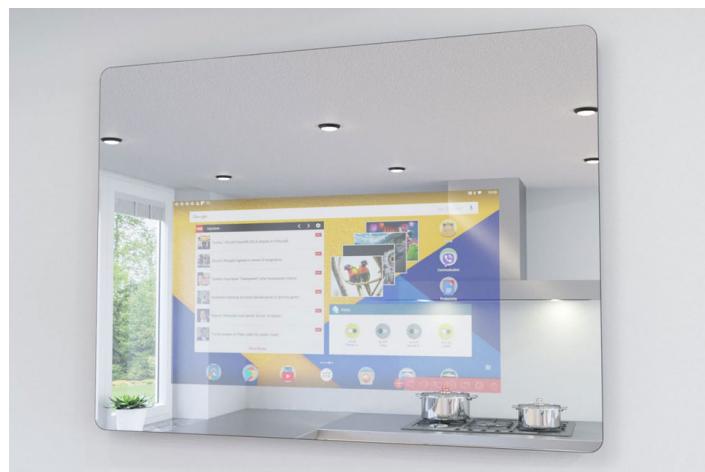


Fig. 5 Oglindă de la Embrace



Fig. 6 Oglindă DIY

## 1.4 Structura pe capitole

Structura lucrării este compusă din 4 capitole care acoperă funcționalitățile sistemului, implementarea acestuia și modul de folosire.

În capitolul 2 sunt prezentate cele două componente care formează sistemul și tehnologiile utilizate în dezvoltarea acestuia. Fiecare componentă reprezintă câte un subcapitol și anume Componenta Hardware și Componenta Software.

De-a lungul capitolului 3 este prezentat modul cum s-au implementat funcționalitățile sistemului și anume a celor trei componente software, baza de date, interfața grafică și saitul de configurare.

În capitolele 4 și 5 sunt prezentate modul cum se utilizează sistemul, o posibilă implementare fizică și respectiv concluziile trase la finalitatea elaborării acestei lucrări.

## 2 Tehnologii utilizate

În cadrul lucrării s-a realizat un sistem de vizualizare a informațiilor personale pe un ecran integrat într-o oglindă. Acesta se poate configura cu ajutorul unui site găzduit pe același sistem electronic. Pentru a oferi posibilitatea folosirii de către mai mulți utilizatori se pot configura mai multe conturi separate, autentificarea pe oglindă făcându-se automat pe baza senzorului optic.

Etapele predominante din dezvoltarea proiectului sunt următoarele :

- documentarea asupra modulelor software necesare de dezvoltat și a celor trei utilizate;
- realizarea scheletului software precum baza interfeței grafice și a saitului;
- integrarea modulelor în schelet și optimizarea funcționării acestora;

Sistemul a fost creat utilizând două categorii de componente. Componenta hardware a fost cea care a dictat capabilitățile sistemului și cea software a fost cea care a permis dezvoltarea funcționalităților sistemului. Acestea vor fi descrise separat în următoarele două subcapitole.

### 2.1 Componenta hardware

Componentele hardware utilizate pentru a dezvolta lucrarea prezentată este formată din:

- elemente de procesare - Raspberry Pi 4B;
- elemente de intrare - Microsoft LifeCam HD-3000;
- elemente de stocare a datelor - microSD de 32GB;
- elemente de conexiune - cablu HDMI, adaptor microHDMI - HDMI;
- elemente de alimentare - alimentator Raspberry Pi 4B;
- elemente de afișare - ecran oarecare de minim 13 inci și rezoluție minimă HD;

Elementele de conexiune și de stocare sunt produse generice alese în funcție de necesități. Dimensiunea cardului microSD de 32GB a fost aleasă deoarece este nevoie de minim 2GB pentru sistemul de operare al Raspberry-ului, mai exact Raspbian OS, și de aproximativ 1GB pentru librăria OpenCV, utilizată pentru procesarea imaginilor. Pe langă cele două necesități de stocare, dimensiunea a fost aleasă și din considerente de extindere a capabilităților proiectului. Asupra alegerii cablului HDMI nu au existat anumite constrângeri deoarece

proiectul prezentat nu se folosește de funcționalități apărute în ultimele versiuni ale protocolului. [8] Elementul de alimentare al plăcii Raspberry Pi a fost pus la dispoziție de către producătorii sistemului.

### 2.1.1 Descrierea sistemului Raspberry Pi

Raspberry Pi este o serie de sisteme cu microprocesor dezvoltată în Regatul Unit al Marii Britanii și al Irlandei de Nord de către Raspberry Pi Foundation pentru a promova bazele domeniului tehnologiei informației în școli. [9] Primul model s-a bucurat de o popularitate atât de mare încât s-a ajuns la iterația a 4-a, cea folosită în prezenta lucrare.

Acest sistem este foarte versatil putând fi folosit pentru o multitudine de proiecte, de la un sistem automat de control al unei case până la un sistem de control al unui robot pentru a realiza anumite obiective. Pentru a oferi această versatilitate sistemul este dotat cu un microprocesor relativ puternic față de competitori, o placă grafică integrată, o multitudine de interfețe și memorie RAM de dimensiuni medii din punct de vedere al capacitații de stocare.

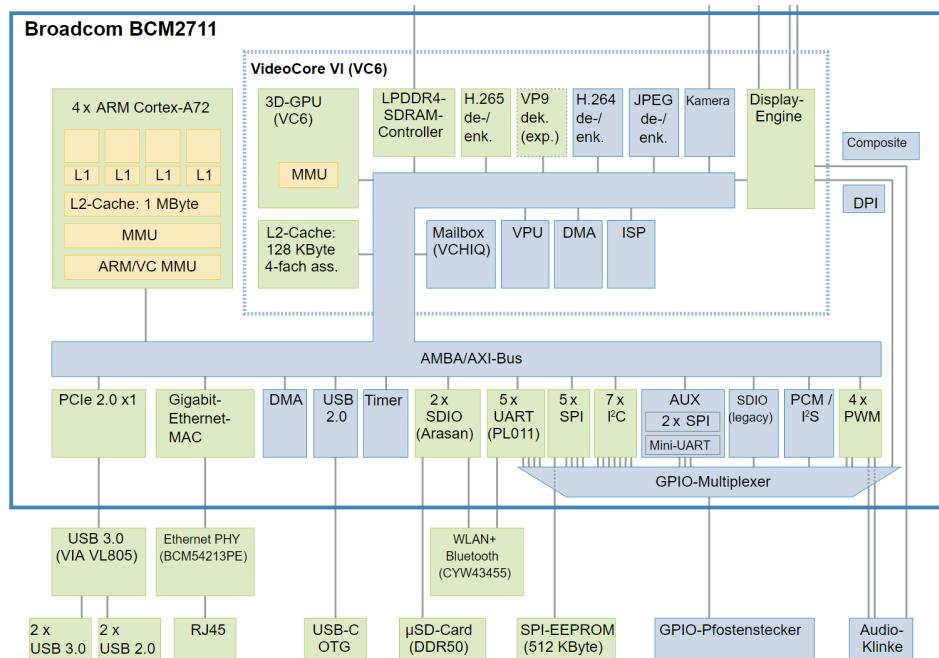


Fig. 7 Schemă bloc a Raspberry Pi 4B

Microprocesorul și placă grafică ale modelului 4B sunt integrate în SoC (system-on-chip), mai exact în modelul Broadcom BCM2711. Acesta are integrat un microprocesor de tip

ARM Cortex-A72 cu patru nuclei având o frecvență de tact de 1,5 GHz și o arhitectură pe 64 de biți. Placa grafică este de tipul VideoCore VI cu capabilitatea de a afișa pe două ecrane 4K 2160p simultan.

În mod general, placa de dezvoltare Raspberry Pi 4B are mai multe dimensiuni de memorie RAM (2, 4, 8 GB). Modelul ales de mine este de 4GB deoarece lucrarea presupune și existența unui server web.

Pe partea de conectivitate, sistemul dispune de un modul Ethernet cu viteze până la 1Gbit/s, modul Bluetooth 5.0, modul Wi-Fi care oferă capacitatea de conectare la rețele de 2.4 GHz și 5GHz, patru porturi USB (Universal Serial Bus), un port GPIO (General Purpose Input Output) cu 40 de pini, două porturi microHDMI, port 3.5mm pentru audio, UART, SPI, I<sup>2</sup>C și două porturi de tip MIPI pentru ecran și senzor optic. [10]

Pe lângă modelele care dispun de toate specificațiile asemănătoare celor prezentate mai sus, există și variante mai reduse ca dimensiuni fizice create special pentru proiecte unde puterea de procesare sau conectivitatea necesară nu este foarte solicitantă. Aceste modele sunt Raspberry Pi Zero (există și modelul Zero W care are și modul Wi-Fi) și Compute Module conform paginii care prezintă oferta echipamentelor disponibile. [11]



Fig. 8 Raspberry Pi Zero



Fig. 9 Raspberry Pi 4B



*Fig. 10 Raspberry Compute Module 3+<sup>3</sup>*

### 2.1.2 Descrierea camerei Microsoft LifeCam HD-3000

Acest modul de conectare video, produs al Microsoft, apărută în prima jumătate a anilor 2000, are o rezoluție de filmare HighDefinition (1270x720 pixeli) la 30 de cadre pe secundă. Camera are și capacitatea de a captura imagini la o rezoluție de 1280x800 de pixeli. Senzorul optic este color, având o profunzime a culorilor de 24 biți și ajustare automată a balansului de alb cu posibilitatea de suprascriere a acesteia.

Pe lângă posibilitățile foto-video, aceasta are integrat și un microfon omnidirecțional. Acest modul se poate conecta la majoritatea dispozitivelor electronice cu suport USB, fiind compatibilă cu aproape orice sistem de operare de pe piață actuală. [12]



*Fig. 11 Microsoft LifeCam HD-3000<sup>4</sup>*

---

<sup>3</sup> [ [https://www.raspberrypi.org/homepage-9df4b/static/3cf381bf7f590051e6c441bf4bca31c/7fd5d/0807fa6b937b11be2d3acc8efefafda005b147a0c\\_cm304.jpg](https://www.raspberrypi.org/homepage-9df4b/static/3cf381bf7f590051e6c441bf4bca31c/7fd5d/0807fa6b937b11be2d3acc8efefafda005b147a0c_cm304.jpg) ]

<sup>4</sup> [ <https://m.media-amazon.com/images/S/aplus-media/mg/3e8dd051-e594-48bf-a78a-f4daf15b77b3.jpg> ]

### 2.1.3 Descrierea elementului de afișare

Din considerente de cost, în timpul dezvoltării proiectului s-a utilizat televizorul propriu (Samsung SyncMaster XL2370HD) sau ecranul propriului laptop (écran al laptopului Macbook Pro 13 2017). Specificația comună ale celor două este raportul ecranului, mai exact un raport de 16:9.

Având în vedere că s-au utilizat două rezoluții diferite în timpul dezvoltării, mai precis o rezoluție de 1280x720 și de 2560x1600, s-a putut testa răspunsul sistemului la diferite valori. Acest proiect fiind unul de tip DIY nu se pune accent pe calitatea ecranului sau utilizarea unui model anume, mai degrabă se pune accent pe rezoluția minimă HD și raportul de 16:9.



Fig. 12 Samsung SyncMaster XL2370HD



Fig. 13 Apple MacBook Pro 13" 2017

## 2.2 Componența software

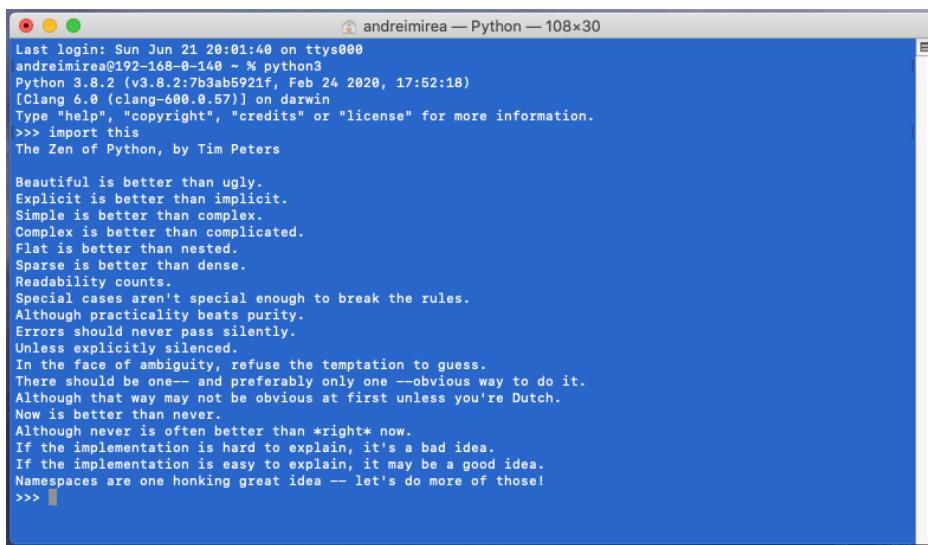
Pentru a dezvolta sistemul de vizualizare a informațiilor personalizate și de configurare a acestora au fost folosite mai multe tehnologii software.

### 2.2.1 Python

Python este un limbaj de programare interpretat, de nivel înalt, destinat programării în scop general. Creat de Guido Von Rossum și lansat în 1991, filosofia designului a limbajului pune accentul pe calitatea de a fi ușor de citit. Construcțiile de limbaj și caracterul orientării pe obiecte țină pentru ca programatorii să scrie cod atât pentru proiecte de scală largă cât și redusă. Acest limbaj de programare este perceput ca un succesor al limbajului de programare ABC.

Actuala versiune a acestui limbaj este 3.8.3. Compatibilitatea între versiunile recente și cele precedente este dată de capacitatea programatorilor de a actualiza modul de folosire a librăriilor standard pentru a fi funcționale cu variantele noi.

Filosofia de bază a limbajului de programare este cuprinsă în "Zen of Python", principii ce au influențat modul cum programatorii au structurat codurile sursă create. Această colecție de principii poate fi oricând accesată din interpretatorul Python rulând comanda `import this`.



```
andreimirea — Python — 108x30
Last login: Sun Jun 21 20:01:40 on ttys000
andreimirea@192-168-0-140 ~ % python3
Python 3.8.2 (v3.8.2:7b3ab5921f, Feb 24 2020, 17:52:18)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>> █
```

Fig. 14 Interpretatorul Python afișând Zen of Python

În loc de a avea toată funcționalitatea înăuntrul nucleului, Python a fost conceput pentru a fi deosebit de extensibil. Modularitatea compactă a făcut populară ideea de a adăuga interfețe programabile la aplicații deja existente. Acest lucru se poate observa în multitudinea de librării standard și terțe create de programatori individuali sau de întregi companii, precum Google.

Librăriile Python, mai exact multitudinea acestora, reprezintă punctul forte al acestui limbaj. Acestea sunt foarte variate, de la simpla librărie math cu care se pot realiza calcule matematice complexe până la librarii pentru crearea de interfețe grafice sau saituri. Numărul acestor librării depășește 200.000 la momentul actual și acoperă multe domenii :

- automatizare;
- analiză de date;
- baze de date;
- interfețe grafice;
- procesare de imagini;
- învățare automată (machine learning);
- aplicații de mobil;
- computații științifice;
- rețelistică;
- administrare de sisteme;
- cadre de testare automată (test frameworks);
- procesare de text;
- cadre de sisteme web (web frameworks);
- web scraping (preluare de informații brute de pe saituri);

Sintaxa și semantica acestui limbaj de programare au fost gândite tocmai pentru a oferi o lizibilitate foarte sporită. Una din diferențele față de alte limbi de programare este utilizarea indentării pentru a delimita zone de cod, de exemplu înăuntrul unei funcții sau unei clase, delimitare care în alte limbi de programare se face cu ajutorul accoladelor sau a unor cuvinte cheie. Multe din cuvintele de control sunt asemănătoare altor limbi de programare, precum `for`, `while`, `if`, `return`, dar există și elemente care par a fi la fel cu echivalentul din alte limbi de programare. Pentru a evidenția simplitatea utilizării acestui limbaj de programare se va utiliza următorul exemplu.

```

1  from os import system
2
3
4  def print_divisors(n):
5      i = 1
6      while i <= n:
7          if n % i == 0:
8              print(i)
9          i = i + 1
10
11
12 > if __name__ == '__main__':
13     limit = input("Pana la ce numar se doreste calcularea divizorilor acestuia: ")
14
15     for i in range(1,limit):
16         system('clear')
17         print(str(i) + ':')
18         print_divisors(i)
19         input()
20         system('clear')
21

```

Fig. 15 Exemplu de algoritm scris in Python

Exemplul de mai sus este al unui algoritm ineficient de calcul al tuturor divizorilor numerelor de la 2 până la limita introdusă de utilizator. Importarea librăriilor este facilă, în sensul că dacă se dorește importarea unei întregi librării se poate folosi `import (os)`, sau, cum este în exemplu, doar a unei componente (funcție, clasă), funcția `system` din librăria `os`. Există un anumit set de reguli de spațiere a construcțiilor în Python, cum ar fi între finalul unei componente și implementarea alteia trebuie să existe două rânduri libere.

O altă particularitate notabilă este faptul că acest limbaj de programare, în momentul executării, va rula fiecare linie de cod scrisă la nivelul 0 al indentării. Dacă se dorește structurarea unui algoritm după structura limbajelor ce folosesc funcția `main` se poate utiliza artificiul prezentat în figură.

Implementarea interpretatorului este realizată în mai multe limbi de programare cum ar fi C, Common Lisp, C++ (în tandem cu C), C#, Java sau chiar Python. [13] Totuși, din toate variantele de implementări, cea în C este de referință purtând numele de CPython.

Numele de Python este derivat din numele grupului de comedie britanic Monty Python, grup de care creatorul lui limbajului îi place. În documentația limbajului sunt diferite referințe către grupul de comedie, cum ar fi folosirea cuvintelor "spam" și "eggs" în loc de faimoasele "foo" și "bar".

Începând cu anul 2003, Python s-a clasat constant în primele zece cele mai populare limbi de programare conform TIOBE Programming Community Index. Aceasta este folosit în

diverse domenii, de unde și numărul larg de librării disponibile, de către o multitudine de organizații precum Google, Wikipedia, NASA, CERN, Facebook, etc și integrat în diverse aplicații precum Abaqus, Blender, Cinema 4D, GIMP, Maya, etc. [14]

### 2.2.2 HyperText Markup Language, sau HTML, și Jinja

HTML este limbajul de etichetare standard utilizat în documentele afișate de navigatoarele web. Acest limbaj poate a fi asistat de alte tehnologii, precum CSS<sup>5</sup> sau limbi de script (JavaScript de exemplu). Versiunea actuală este HTML5, întreținerea acestuia făcându-se de un consorțiu format din marii producători de navigatoare web (Apple, Google, Mozilla și Microsoft) numit WHATWG.<sup>6</sup>

Navigatoarele web preiau documentele HTML de la un server sau din stocarea locală și le traduce vizual în pagini multimedia. Acest limbaj de programare conține elemente care se numesc etichete. O etichetă constituie un construct care poate semnifica rubrica unui text, declararea unui paragraf, afișarea unor alte obiecte precum liste, imagini, linkuri. [15]

Jinja este o tehnologie de modelare folosită în tandem cu Python, creată de Armin Ronacher. Aceasta este asemănătoare instrumentului de modelare Django<sup>7</sup>, însă Jinja oferă expresii înrudite cu Python și posibilitatea de a apela funcții cu argumente pe anumite obiecte. Este o tehnologie de modelare bazată pe text, astfel având capacitatea de a genera orice tip de etichetă de tip HTML sau altele. [16]

### 2.2.3 Limbajul de programare SQL

SQL, sau Structured Query Language, este un limbaj de programare specific domeniului folosit pentru a întreține informațiile dintr-un sistem de management relațional al bazelor de date (de exemplu SQLite<sup>8</sup>).

Acest limbaj de programare a devenit un standard al Institutului American de Standarde (cunoscut sub acronimul ANSI) în anul 1986 și al Organizației Internaționale a

<sup>5</sup> Cascading Style Sheets, standard pentru formatarea elementelor unui document HTML.

<sup>6</sup> Web Hypertext Application Technology Working Group.

<sup>7</sup> Soft cadru pentru dezvoltarea aplicațiilor web (en. web application framework) gratuit și cu sursă deschisă, scris în Python. [34]

<sup>8</sup> Va fi detaliat într-un subcapitol următor.

Standardizării (sau ISO) în anul 1987. Din această perioadă a început un proces de includere a mai multor funcționalități.

O comandă de tip SQL are mai multe cuvinte cheie care folosite într-o anumită ordine oferă funcționalități diverse. Există mai multe categorii de cuvinte chei precum:

- clauze, adică componente ale instrucțiunilor și interogărilor;
- expresii, adică cuvinte cheie care produc valori scalare sau tabele;
- predicate, adică condiții care sunt evaluate în logică ternară sau booleană în scop de control al funcționării comenzi sau seriei de comenzi;
- interogări, adică cuvinte chei ce permite primirea de date care satisfac anumite criterii;
- declarații, adică elemente ce au efect persistent asupra datelor sau structurii;
- datelor, sau pot controla funcționarea tranzacțiilor; [17]

#### 2.2.4 PyCharm Professional

PyCharm este un mediu de programare integrat apărut în Iulie 2010 și folosit în special pentru Python. A fost dezvoltat de o companie cehă JetBrains. Asigură capacitateți de analiză a codului, un depanator (debugger) grafic, testare automată, facilități pentru baze de date, integrare cu sisteme de versionare și suportă dezvoltarea de saituri folosind Django sau Flask. Pe lângă caracteristicile de bază se mai pot instala și alte adăugiri create de terți.

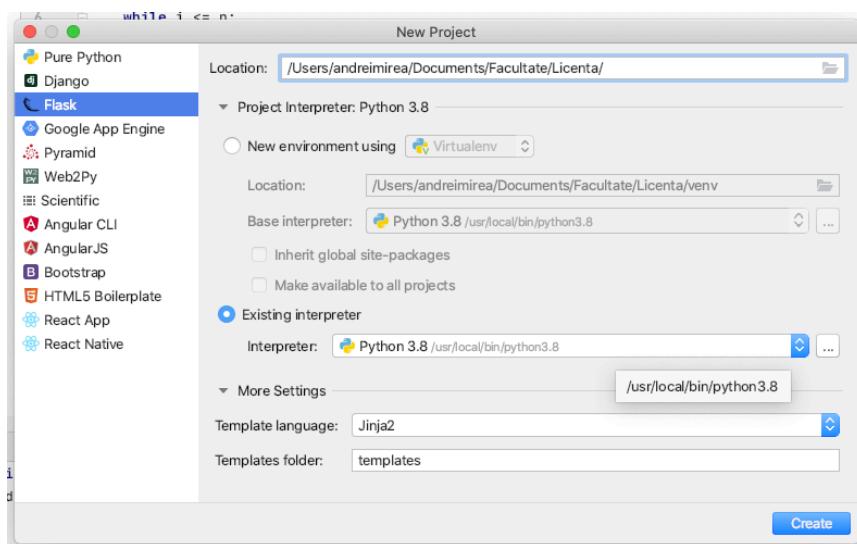


Fig. 16 Interfața pentru a crea un nou proiect din PyCharm

Aplicația este disponibilă pe multe platforme, precum Windows, macOS, sau Linux. Este disponibil atât în varianta gratis, Community Edition, cât și în varianta plătită, Professional Edition. [18]

Pentru dezvoltarea acestui proiect s-a utilizat varianta Professional 2020.1 având licență oferită de JetBrains pentru studenți pe baza emailului personal de la Universitatea Politehnica Timișoara.

### 2.2.5 SQLite

SQLite este un sistem relațional de management al bazelor de date conținut în librăria C. A fost dezvoltat în primăvara anului 2000 de către General Dynamics în colaborare cu Marina Statelor Unite ale Americii. În contrast cu celelalte sisteme de management acesta nu este un sistem de tip client-server. Aceasta este o alegere populară pentru a fi folosit în sisteme integrate. Este folosit în prezent de către navigatoare web și sisteme de operare.

Spre deosebire de celelalte sisteme de management al bazelor de date, SQLite nu este susținut de către un proces independent cu care aplicația terță este nevoită să comunice. Informațiile bazelor de date de tipul menționat sunt salvate într-un fișier. Operațiunile de citire a informațiilor conținute se poate face de către mai mulți clienți deodată, în schimb operațiunile de scriere se realizează pe rând.

Acest sistem implementează majoritatea funcționalităților descrise de SQL-92<sup>9</sup>. Tipul datelor este atribuit dinamic, mai precis nu este atribuit unei coloane ci valorilor individuale din coloane. Pe lângă aceasta este tipizat slab, adică pe o coloană de tip integer se poate insera o valoare de tip string, problema aceasta putând fi rezolvată prin setarea de constrângeri. [19]

### 2.2.6 GitHub

GitHub este o aplicație web folosită de către dezvoltatorii de software atât pentru versionarea proiectelor cât și pentru oferirea posibilității ca mai mulți dezvoltatori să lucreze pe aceleși module ale unui proiect. De către programatorii liberi profesioniști este folosit și pentru a găzdui, în mod public sau privat, proiectele software create de aceștia.

---

<sup>9</sup> A treia revizie a limbajului de interogare a bazelor de date SQL, realizat în noiembrie 1992. [35]

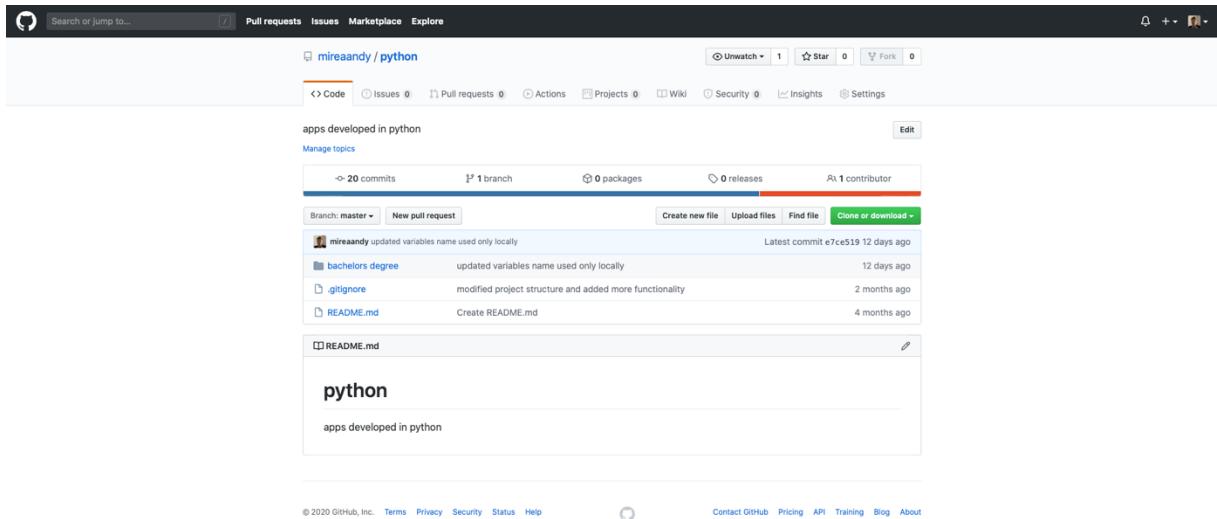


Fig. 17 Interfața web GitHub

În adiție față de capabilitățile de bază, aplicația oferă și :

- suport pentru a avea documentație;
- organizarea rezolvării anumitor obiective;
- suport pentru integrare continuă (CI);
- istoricul modificărilor aduse fiecărui modul în parte;
- diverse grafice;
- abilitatea de a se crea pagină web personală; [20]

### 3 Implementarea soluției

Proiectul prezentat în cadrul acestei lucrări a fost implementat utilizând echipamentele software și hardware prezентate în capitolul precedent. Soluția este formată din trei părți componente, mai exact partea conținând interfața grafică a oglinzii, partea cu care se face configurarea aplicației, mai precis un sait, și o bază de date care conține informații despre utilizatorii înregistrați.

Structura directoarelor conținătoare este prezentată în figura de mai jos.

```
andreimirea@macbook-pro bachelors degree % tree -v --charset utf-8
.
|-- config
|   |-- encodings.pickle
|   |-- haarcascade_frontalface_default.xml
|   |-- mirrorDatabase.db
|   |-- userCredentials
|       |-- adragan
|           |-- andrei
|               |-- dinuipati
|-- userPictures
    |-- Sabi3116.jpg
    |-- Sabi3117.jpg
    |-- andrei0.jpg
    |-- andrei1.jpg
    |-- andrei2.jpg
    |-- andrei3.jpg
    |-- andrei4.jpg
    |-- andrei5.jpg
    |-- andrei6.jpg
    |-- andrei7.jpg
    |-- andrei8.jpg
    |-- andrei9.jpg
    |-- andrei10.jpg
    |-- andrei11.jpg
    |-- andrei12.jpg
    |-- andrei13.jpg
    |-- andrei14.jpg
    '-- andrei15.jpg
    '-- weatherIcons
        |-- Cloud.png
        |-- Hail.png
        |-- Haze.png
        |-- Moon.png
        |-- Newspaper.png
        |-- PartlyMoon.png
        |-- PartlySunny.png
        |-- Rain.png
        |-- Snow.png
        |-- Storm.png
        |-- Sun.png
        |-- Sunrise.png
        |-- Tornado.png
        '-- Wind.png
|-- mirrorUI
    |-- __pycache__
        |-- news.cpython-38.pyc
        |-- userHandler.cpython-38.pyc
        '-- window.cpython-38.pyc
    '-- news.py
    '-- userHandler.py
    '-- window.py
|-- mirrorWeb
    |-- __pycache__
        |-- app.cpython-37.pyc
        |-- app.cpython-38.pyc
        |-- configuration.cpython-38.pyc
        |-- forms.cpython-38.pyc
        '-- models.cpython-38.pyc
    '-- app.py
    '-- configuration.py
    '-- credentials.json
    '-- forms.py
    '-- models.py
    '-- static
        '-- templates
            |-- base.html
            |-- confirmpass.html
            |-- forgotpass.html
            |-- index.html
            |-- main.html
            '-- signup.html
10 directories, 60 files
```

Fig. 18 Structura de directoare și fișiere a proiectului

Pentru deschiderea sistemului prezentat este nevoie de rularea a două fișiere python, mai exact window.py din directorul mirrorUI și app.py din directorul mirrorWeb.

Arhitectura hardware a echipamentelor utilizate este cuprinsă dintr-un Raspberry Pi 4B conectat cu ajutorul elementului de alimentare la tensiunea de 230V, cu ajutorul elementelor de conexiune a fost atașat ecranul și senzorul optic, cel folosit fiind Microsoft LifeCam HD-3000, a fost conectat prin USB. În cazul în care a fost folosit laptopul personal, au fost folosite echipamentele integrate de senzor optic și afișare.

### 3.1 Implementarea bazei de date

Pentru a implementa baza de date de SQLite au fost executate două comenzi de CREATE TABLE în mediul de programare PyCharm. Acestea au fost permis realizarea a două tabele, userData și userPictures. Comenzile, având câmpurile puse în evidență, sunt :

- CREATE TABLE "userData" (  
    "**id**" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,  
    "**userName**" varchar(100) NOT NULL DEFAULT ",  
    "**password**" varchar(128) NOT NULL,  
    "**newsTopic**" varchar(50) DEFAULT ",  
    "**isActive**" varchar(1) NOT NULL DEFAULT 0,  
    "**googleToken**" varchar(1000))
- CREATE TABLE "userPictures" (  
    "**userId**" INTEGER NOT NULL,  
    "**pictureId**" INTEGER NOT NULL,  
    "**picturePath**" varchar(100) NOT NULL,  
    "**encoded**" integer(1) NOT NULL,  
    CONSTRAINT "pictureId" PRIMARY KEY ("pictureId"),  
    CONSTRAINT "userId" FOREIGN KEY ("userId") REFERENCES  
    "userData" ("id") ON DELETE CASCADE ON UPDATE CASCADE)

Tabelul userData este folosit pentru a reține utilizatorii înregistrați. Se salvează despre acestia un nume de autentificare, parola acestora, subiectul știrilor ales, dacă sunt activi la un moment dat și calea către fișierul binar unde sunt salvate datele de conectare la serviciile Google.

Tabelul userPictures este folosit pentru a reține pozele utilizatorilor înregistrați. Pentru fiecare poză este reținută cheia de identificare a utilizatorului, o cheie unică de identificare a

acesteia, calea absolută către poză și dacă a fost prelucrată poza de către algoritmul de recunoaștere facială.

Cele două tabele sunt relaționate, mai exact câmpul userId din userData cu câmpul userId din userPictures, relație de tipul unul la mulți.

### 3.2 Implementarea interfeței grafice și a funcționalităților acestora

Interfața grafică a fost realizată în Python utilizând librăria Tkinter [21] (se mai folosește denumirea de Tk). Fiecare parte componentă a interfeței este de fapt un Frame (obiect din Tk) care are ca părinte fereastra principală, element de tip Tk. În figurile de mai jos sunt prezentate scheletul interfeței și interfața propriu-zisă.

```

33  class Clock(Frame):...
65
66
67  class Email(Frame):...
186
187
188  class EmailElement(Frame):...
116
117
118  class Calendar(Frame):...
156
157
158  class CalendarEvent(Frame):...
166
167
168  class Newsletter(Frame):...
187
188
189  class NewsElement(Frame):...
195
196
197  class Weather(Frame):...
281
282
283  class Window(Tk):...
326
327
328 >  if __name__ == '__main__':
329      window = Window()
330      window.mainloop()
331

```

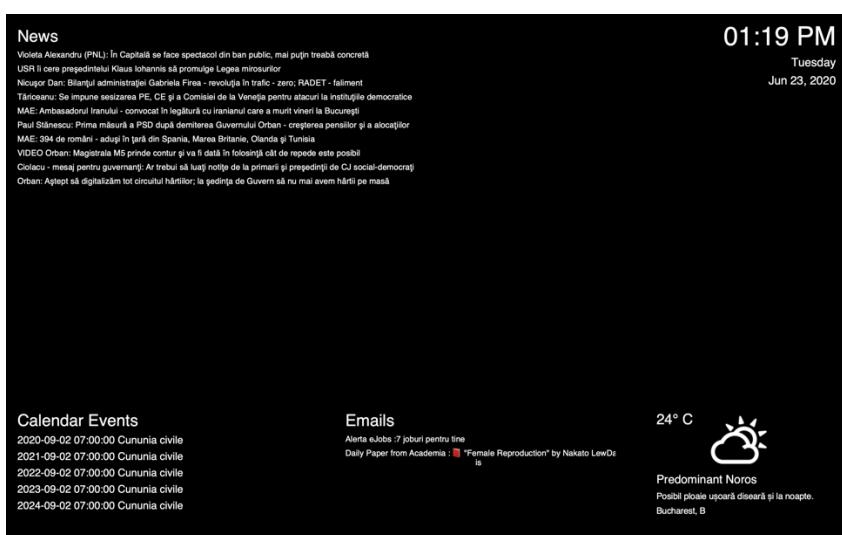


Fig. 19 Scheletul interfeței Tk.

Fig. 20 Interfața propriu-zisă

După cum se poate observa, numele claselor sunt sugestive fapt ce face identificarea acestora în figura alăturată simplă. Clasele Email, Calendar și Newsletter sunt conectate cu baza de date, locul unde sunt salvate căile către fișierele de configurare sau chiar date personalizate. Conectarea la baza de date și actualizarea informațiilor afișate, vorbind de cele trei clase menționate mai sus, sunt realizate în funcțiile `get_news()`, `get_emails()` și `get_events()`.

Obiectele care asigură afişarea textului și a iconițelor prezente în interfața grafică sunt de tipul Label sau împreună cu ImageTk unde este cazul. Acestea sunt parametrizate în funcție de necesitățile situație, precum setarea fontului sau a imaginii de afișat. Pentru exemplificarea utilizării acestor clase se va utiliza clasa Newsletter.

```

176 class Newsletter(Frame):
177     def __init__(self, parent, *args, **kwargs):
178         Frame.__init__(self, parent, bg='black')
179         self.title = 'News'
180         self.newsHandler = News(parent.currentActiveUser.newsTopic)
181         self.newsLabel = Label(self, text=self.title, font=("Helvetica", 28), fg="white", bg="black", justify="left")
182         self.newsElementContainer = Frame(self, bg="black")
183         self.parent = parent
184
185         self.newsLabel.pack(side=TOP, anchor=W)
186         self.newsElementContainer.pack(side=TOP, anchor=E)
187
188     def get_news(self):
189         for widget in self.newsElementContainer.winfo_children():
190             widget.destroy()
191
192         for newsTitle in self.newsHandler.get_news():
193             news_element = NewsElement(self.newsElementContainer, news_title=newsTitle)
194             news_element.pack(side=TOP, anchor=W)

```

*Fig. 21 Implementarea clasei Newsletter*

Obiectele descrise mai sus sunt declarate în funcția `__init__(self)`, echivalentul unui constructor conform metodologiei programării pe obiecte. În primă instanță se apelează constructorul clasei de bază (linia 178 din figură), mai apoi făcându-se o salvare a titlului de afișat. În cazul de față este necesar și folosirea obiectului de tip News pentru a putea prelua știrile de la sursă. Urmează folosirea obiectului de tip Label care ne asigură locul afișării titlului. Pentru a avea o altă secțiune în acest Frame se declară și o altă zonă Frame înăuntrul acesteia. Plasarea efectivă a obiectelor Label și Frame se face cu ajutor metodei aparținătoare `pack()`. Celelalte informații aparținătoare celorlalte clase utilizează aceleași principii descrise mai sus, diferența constând în informațiile afișate și modul de afișare a acestora.

Pentru asigurarea existenței unei ferestre se utilizează un obiect de tip Tk care în funcția `__init__(self)` sunt declarate diferite proprietăți utilizate de-a lungul rulării, cum ar fi clasa utilizată pentru relaționarea cu baza de date, sau frecvența de actualizare a anumitor componente și Frame-urile utilizare pentru afișarea informațiilor. Plasarea componentelor de afișare se realizează cu ajutorul funcției `place()`, altă metodă aparținătoare clasei Frame. Rularea efectivă a interfeței se face apelând funcția `mainloop()`, funcție componentă a clasei Window derivată din Tk.

```

283 class Window(Tk):
284     def __init__(self):
285         Tk.__init__(self)
286         self.title("mirror")
287         self.configure(background="black")
288         self.attributes("-fullscreen", True)
289         self.columnconfigure(0, weight=1)
290         self.rowconfigure(0, weight=1)
291         self.three_min_refresh_rate = 0
292         self.database = Database()
293         self.currentActiveUser = Database.get_active_user(None)
294         self.newsFrame = Newsletter(self)
295         self.clockFrame = Clock(self)
296         self.calendarFrame = Calendar(self)
297         self.weatherFrame = Weather(self)
298         self.emailFrame = Email(self)
299         self.noEmailsDisplayed = 5
300
301         self.clockFrame.place(x=self.winfo_screenwidth() * 0.85, y=self.winfo_screenheight() * 0.01)
302         self.newsFrame.place(x=self.winfo_screenwidth() * 0.01, y=self.winfo_screenheight() * 0.02)
303         self.calendarFrame.place(x=self.winfo_screenwidth() * 0.01, y=self.winfo_screenheight() * 0.75)
304         self.weatherFrame.place(x=self.winfo_screenwidth() * 0.77, y=self.winfo_screenheight() * 0.75)
305         self.emailFrame.place(x=self.winfo_screenwidth()*0.40, y=self.winfo_screenheight()*0.75)
306         self.update_tk()

```

Fig. 22 Constructorul clasei Window

La finalul constructorului prezentat în figură este apelată funcția `update_tk()` care se ocupă cu apelul funcțiilor de actualizare a informațiilor din fiecare Frame și a funcției de recunoaștere facială. Această funcție este auto-apelată o dată la o secundă cu ajutorul funcției `after()` din clasa Tk. Frecvența aceasta a fost aleasă pentru ca modulul de ceas să fie actualizat cât mai precis și din considerente de simplitate în a calcula frecvența de actualizare a celorlalte module.

```

308     def update_tk(self):
309         self.clockFrame.tick()
310
311         self.currentActiveUser = Database.get_active_user(None)
312         self.newsFrame.newsHandler.replace_keyword(self.currentActiveUser.newsTopic)
313
314         if self.three_min_refresh_rate == 5:
315             self.weatherFrame.get_weather()
316             encode_pictures()
317             recognize_face()
318             self.calendarFrame.get_events()
319             self.newsFrame.refresh_news()
320             self.emailFrame.get_emails()
321             self.three_min_refresh_rate = 0
322
323             self.three_min_refresh_rate += 1
324
325             self.after(1000, self.update_tk)

```

Fig. 23 Funcția `update_tk()`

În următoarele subcapitole sunt prezentate implementările funcționalităților, cum ar fi recunoașterea facială, conectarea la baza de date de tip SQLite și preluarea informațiilor din contul personal Google și a vremii.

### 3.2.1 Colectarea de date

Funcționalitatea de preluare a emailurilor și a calendarului a fost implementată cu ajutorul documentației de la Google API [22]. Pentru a putea face acest lucru a fost nevoie de activarea a unor elemente de dezvoltator din contul personal Google. Implementarea conectării în Python este facilitată de două librării puse la dispoziție de companie sub denumirea de `googleapiclient.discovery` și `google.auth.transport.requests`.

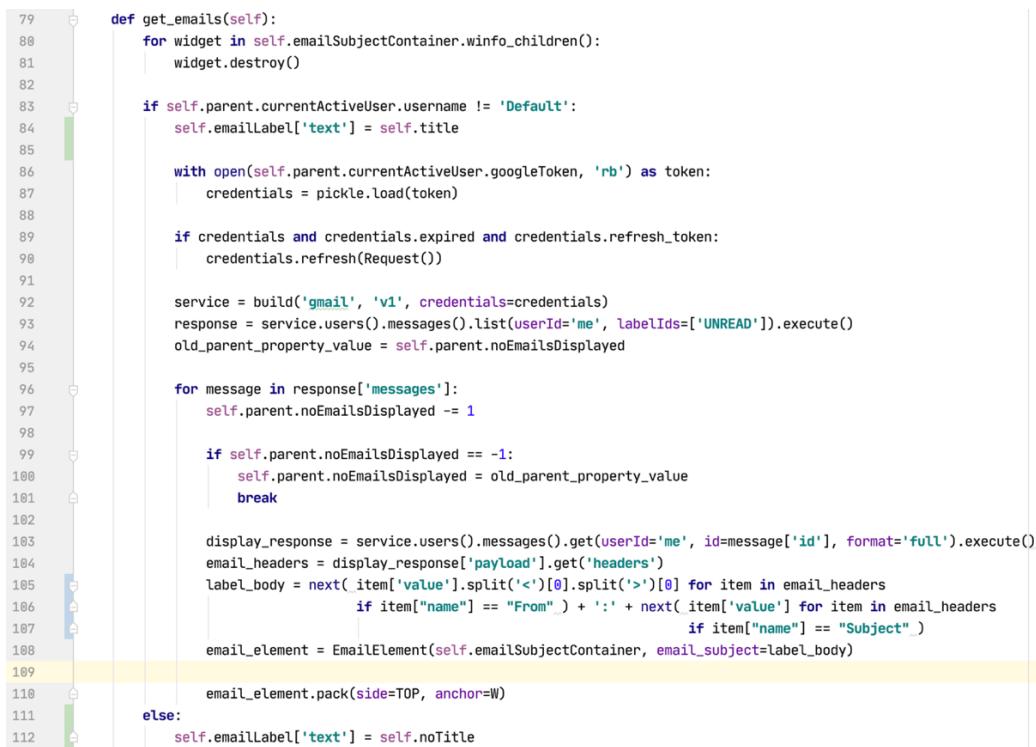


*Fig. 24 Modul email*



*Fig. 25 Modul calendar*

Informațiile de conectare la contul utilizatorului pentru a prelua informațiile minimale sunt salvate sub formă binară într-un fișier a cărui cale este salvată în baza de date. După preluarea obiectului de conectare, se verifică starea acestuia de validitate și în cazul expirării se face o actualizare a cheilor de conectare (linia 90 din figura 26). Pentru a se conecta la contul Google al utilizatorului este nevoie de crearea unui serviciu având în componență obiectul de conectare. Utilizând acest serviciu se execută comanda de a lista cheile de identificare a emailurilor necitite. Această listă este mai apoi parcursă pentru a prelua expeditorul și subiectul acestora folosind același serviciu. Informațiile preluate precedent sunt aduse într-o formă lizibilă, mai exact "expeditor : subiect", ulterior declarându-se un obiect de tip EmailElement având ca text forma lizibilă a unui email și se afișează cu ajutorul funcției `pack()`. În cazul în care modulul de recunoaștere facială nu identifică o persoană cunoscută se ascunde acest Frame.



```

79     def get_emails(self):
80         for widget in self.emailSubjectContainer.winfo_children():
81             widget.destroy()
82
83         if self.parent.currentActiveUser.username != 'Default':
84             self.emailLabel['text'] = self.title
85
86         with open(self.parent.currentActiveUser.googleToken, 'rb') as token:
87             credentials = pickle.load(token)
88
89         if credentials and credentials.expired and credentials.refresh_token:
90             credentials.refresh(Request())
91
92         service = build('gmail', 'v1', credentials=credentials)
93         response = service.users().messages().list(userId='me', labelIds=['UNREAD']).execute()
94         old_parent_property_value = self.parent.noEmailsDisplayed
95
96         for message in response['messages']:
97             self.parent.noEmailsDisplayed -= 1
98
99             if self.parent.noEmailsDisplayed == -1:
100                 self.parent.noEmailsDisplayed = old_parent_property_value
101                 break
102
103             display_response = service.users().messages().get(userId='me', id=message['id'], format='full').execute()
104             email_headers = display_response['payload'].get('headers')
105             label_body = next(item['value'].split('<')[0].split('>')[0] for item in email_headers
106                               if item['name'] == "From" ) + ':' + next(item['value'] for item in email_headers
107                               if item['name'] == "Subject" )
108             email_element = EmailElement(self.emailSubjectContainer, email_subject=label_body)
109             email_element.pack(side=TOP, anchor=W)
110
111         else:
112             self.emailLabel['text'] = self.noTitle

```

Fig. 26 Funcția de preluare a emailurilor

În mod similar se preiau și informațiile evenimentelor salvate în Google Calendar [23], mai precis se folosesc aceleași informații de conectare, aceeași funcție `build()`, diferența constând în parametrii trimiși și a metodelor utilizate.



```

136     def get_events(self):
137         for widget in self.calendarEventContainer.winfo_children():
138             widget.destroy()
139
140         if self.parent.currentActiveUser.username != 'Default':
141             self.calendarLabel['text'] = self.title
142
143         with open(self.parent.currentActiveUser.googleToken, 'rb') as token:
144             credentials = pickle.load(token)
145
146         if credentials and credentials.expired and credentials.refresh_token:
147             credentials.refresh(Request())
148
149         service = build('calendar', 'v3', credentials=credentials)
150         events_result = service.events().list(calendarId='primary', timeMin=datetime.utcnow().isoformat() + 'Z',
151                                               maxResults=5, singleEvents=True).execute()
152         events = events_result.get('items', [])
153
154         for widget in self.calendarEventContainer.winfo_children():
155             widget.destroy()
156
157         for event in events:
158             event_text = event['start']['dateTime'].split('T')[0] + ' ' + \
159                         event['start']['dateTime'].split('T')[1].split('+')[0] + ' ' + event['summary']
160             calendar_event = CalendarEvent(self.calendarEventContainer, event_name=event_text)
161             calendar_event.pack(side=TOP, anchor=E)
162
163         else:
164             self.calendarLabel['text'] = self.noTitle

```

Fig. 27 Funcția de preluare a evenimentelor de afișat

Modulul de vreme se folosește de un sait web terț pentru a oferi informațiile stării atmosferice curente. Acest sait este <https://api.darksky.net>. [24] Pentru a preluă locația fizică a utilizatorului se folosește saitul <http://api.ipstack.com>. [25]

```

233     def get_weather(self):
234         location_req_url = f"http://api.ipstack.com/{self.get_ip()}?access_key=11442fd93a2b6f35695a8bda9f2891f9"
235
236         response_ip = requests.get(location_req_url)
237         location_obj = json.loads(response_ip.text)
238
239         lat = location_obj['latitude']
240         lon = location_obj['longitude']
241
242         location2 = f'{location_obj["city"]}, {location_obj["region_code"]}'
243
244         weather_req_url = f"https://api.darksky.net/forecast/8da1b82e7ba6ae18bdf188c489a852d6/{lat},{lon}?lang=ro&units=si"
245
246         response_weather = requests.get(weather_req_url)
247         weather_obj = json.loads(response_weather.text)
248
249         degree_sign = u'\N{DEGREE SIGN}'
250         temperature2 = "%s%s %s" % (str(int(weather_obj['currently']['temperature'])), degree_sign)
251         currently2 = weather_obj['currently']['summary']
252         forecast2 = weather_obj["hourly"]["summary"]
253
254         icon_id = weather_obj['currently']['icon']

```

Fig. 28 Funcția de preluare a stării atmosferice

Principiul după care se face preluarea vremii și a locației este simplu. Se face o cerere spre un sait, aceasta având o anumită structură, pentru locație linia 234 și pentru vreme linia 244 din figura 28. Ca urmare a primirii cererii, saitul răspunde cu un text formatat sub formă de JSON în care se află informațiile necesare pentru afișat sau prelucrare ulterioară.

Modulul de știri se folosește de o clasă creată de mine ce preia informațiile brute de pe <http://www.agerpress.ro>. Modul de utilizare a clasei folosite este facil, mai exact se setează subiectul știrilor de preluat, proprietatea keyword, și se apelează funcția `get_news()` pentru primirea știrilor sub forma finală destinată afișării.

```

188     def refresh_news(self):
189         for widget in self.newsElementContainer.winfo_children():
190             widget.destroy()
191
192         for newsTitle in self.newsHandler.get_news():
193             news_element = NewsElement(self.newsElementContainer, news_title=newsTitle)
194             news_element.pack(side=TOP, anchor=W)

```

Fig. 29 Funcția de preluare a știrilor

```

42     class News:
43
44         def __init__(self, keyword):
45             self.keyword = keyword
46
47         def replace_keyword(self, keyword):
48             self.keyword = keyword
49
50         def get_news(self):
51             answer = []
52
53             try:
54                 content = requests.get(websites[self.keyword]).content
55             except:
56                 return answer
57
58             soup = BeautifulSoup(content, "html5lib")
59             news = soup.findAll("article", class_="unit_news shadow p_r")
60
61             for newsPiece in news:
62                 titles = newsPiece.findAll("div", class_="title_news")
63
64                 for title in titles:
65                     answer.append(title.get_text())
66
67             return answer

```

Fig. 30 Clasa News utilizată în modulul de știri

Funcția `get_news()` preia codul HTML al saitului în funcție de subiectul ales de utilizator, cu ajutorul librăriilor `requests` care conține clasa `requests` [26] (linia 54 din figura 30) și `bs4` [27] care conține clasa `BeautifulSoup` (linia 58 din aceeași figură). După preluarea codului HTML se caută în componente HTML și se returnează titlurile știrilor. Mai exact, se caută toate elementele de tip `article`<sup>10</sup> de clasă "unit\_news shadow p\_r" și în cadrul fiecărui element găsit se caută elementele `div`<sup>11</sup> de clasă "title\_news" unde se găsește de fapt textul titlurilor știrilor.

<sup>10</sup> Element de tip etichetă prezent în limbajul de programare HTML.<sup>11</sup> idem.

### 3.2.2 Managementul utilizatorilor

Pentru a oferi posibilitatea utilizării a aceleiași oglinzi de către mai mulți utilizatori, sistemul prezentat oferă funcționalitatea de recunoaștere facială utilizând librăria terță face\_recognition. [28] Această facilitate este implementată în două funcții, encode\_pictures() și recognize\_face().

```

54     def encode_pictures():
55         pictures = Database.session.query(UserPicture).filter_by(encoded=0).all()
56         knownEncodings = []
57         knownNames = []
58
59         for pic in pictures:
60             name = pic.picturePath.split('/')[-1].split('.')[0].translate(str.maketrans('', '', digits))
61             image = imread(pic.picturePath)
62             rgb = cvtColor(image, COLOR_BGR2RGB)
63             boxes = face_recognition.face_locations(rgb, model='hog')
64             encodings = face_recognition.face_encodings(rgb, boxes)
65
66             for encoding in encodings:
67                 knownEncodings.append(encoding)
68                 knownNames.append(name)
69
70             pic(encoded = 1
71
72             Database.session.commit()
73
74         if len(pictures) != 0:
75             data_all = {"encodings": knownEncodings, "names": knownNames}
76             dump_file = open(get_project_path() + '/config/encodings.pickle', mode="rb")
77             dump_file_size = os.path.getsize(dump_file.name)
78
79             if dump_file_size != 0:
80                 data_old = pickle.load(dump_file)
81                 data_all.update(data_old)
82
83             dump_file = open(get_project_path() + '/config/encodings.pickle', mode="wb")
84
85             pickle.dump(obj=data_all, file=dump_file)
86             dump_file.close()
```

Fig. 31 Funcția encode\_pictures()

Primul pas ce este realizat în funcția encode\_pictures() este de a prelua calea pozelor ale unui utilizator, acestea fiind introduse de către acesta din pagina web de configurare a contului și le prelucră. Pentru fiecare poză găsită neprelucrată, această funcție va căuta fețe cu ajutorul modelului HOG<sup>12</sup> (linia 63 din figura 31), le va prelucra (linia 64 din figura 31) și trăsăturile rezultante, variabila encodings, vor fi salvate într-un fișier sub formă binară.

---

<sup>12</sup> descriptor de trăsături folosit în procesarea de imagini, histogram of oriented gradients. [36]

Pozele deja prelucrate vor fi ignorate din motive de eficiență a timpului de rulare și de redundanță.

```

89     def recognize_face():
90         dump_file = open(get_project_path() + '/config/encodings.pickle', "rb")
91
92         if os.path.getsize(dump_file.name) != 0:
93             data = pickle.load(file=dump_file)
94         else:
95             data = None
96
97         dump_file.close()
98
99         if data is not None:
100             camera = VideoCapture(index=0)
101             time.sleep(0.1)
102             if camera.isOpened():
103                 ret_val, frame = camera.read()
104                 camera.release()
105                 rgb = cvtColor(frame, COLOR_BGR2RGB)
106                 boxes = face_recognition.face_locations(rgb, model='hog')
107                 encodings = face_recognition.face_encodings(rgb, boxes)
108                 names = []
109
110             for encoding in encodings:
111                 matches = face_recognition.compare_faces(data["encodings"], encoding)
112                 name = "Default"
113
114                 if True in matches:
115                     matchedIdxs = [i for (i, b) in enumerate(matches) if b]
116                     counts = {}
117
118                     for i in matchedIdxs:
119                         name = data["names"][i]
120                         counts[name] = counts.get(name, 0) + 1
121
122                         name = max(counts, key=counts.get)
123                         names.append(name)
124
125             if len(encodings) == 0:
126                 names.append("Default")
127
128             print(names[0])
129             Database.set_active_user(None, user_active=names[0])
Fig. 32 Funcția de recunoaștere a fețelor

```

Funcția de `recognize_faces()` începe prin a deschide fișierul în care sunt salvate trăsăturile fețelor și numele acestora. Se deschide senzorul optic de la adresa 0 (linia 100 din figura de mai sus), se verifică dacă senzorul este pornit și se realizează o fotografie (linia 103 din figura 32). Fotografia mai apoi este prelucrată pentru a putea fi folosită de funcția `face_locations()` cu scopul de a se găsi modele faciale. Se extrag trăsăturile modelelor și pentru fiecare în parte se realizează o comparație cu trăsăturile fețelor deja salvate. Numele

feței găsite este mai apoi reținut în baza de date ca fiind activ. În cazul în care nu se află cineva recunoscut sau vreo persoană în fața oglinzi, utilizatorul "Default" este activat.

### 3.2.3 Conectarea cu baza de date

Baza de date se află local în directorul "config" sub numele de "mirrorDatabase.db". Conectarea componentei cu interfață grafică la această bază de date este facilitată de către librăriile sqlalchemy, sqlalchemy.ext.declarative și sqlalchemy.orm. [29]

Pentru a se folosi librăriile de mai sus a fost mai întâi implementată clasa Database. În aceasta se realizează declararea unui motor de conectare la baza de date locală (linia 17 din figura 33), unui model de bază a unui tabel (linia următoare) și configurarea acestuia cu motorul de conectare, și a unei sesiuni pentru a se putea executa comenzi SQL.

```

16     class Database:
17         databaseEngine = create_engine('sqlite:///` + get_project_path() + '/config/mirrorDatabase.db')
18         Model = declarative_base()
19
20         Model.metadata.create_all(databaseEngine)
21
22         session = sessionmaker(bind=databaseEngine)()
```

Fig. 33 Clasa Database

Ulterior declarării clasei Database mai este nevoie de implementarea a două clase derivate din Database.Model care să mimice cele două tabele existente în baza de date, mai precis să mimeze tabelele userData și userPictures.

```

33     class User(Database.Model):
34
35         __tablename__ = 'userData'
36         id = Column(Integer, primary_key=True)
37         username = Column(String(100), index=False, unique=True, nullable=False)
38         password = Column(String(128), index=False, unique=False, nullable=False)
39         newsTopic = Column(String(50), index=False, unique=False, nullable=True)
40         isActive = Column(String(1), index=False, unique=False, nullable=False)
41         googleToken = Column(String(1000), index=False, unique=True, nullable=True)
42         pictures = relationship('UserPicture', backref='user', lazy='dynamic')
43
44
45     class UserPicture(Database.Model):
46
47         __tablename__ = 'userPictures'
48         userId = Column(Integer, ForeignKey('userData.id'))
49         pictureId = Column(Integer, primary_key=True)
50         picturePath = Column(String(100), index=False, unique=True, nullable=False)
51         encoded = Column(Integer, index=False, unique=False, nullable=False)
```

Fig. 34 Clasele ce mimează tabelele din baza de date

După cum se observă în figura 34 fiecarei clase i-a fost salvată în proprietatea `_tablename_` numele tabelului aferent. Fiecare cealaltă proprietate există doar pentru a copia coloane din tabele. Modul mimării acestora este sugestivă datorită figurii 34.

Având în vedere aceste clase declarate, utilizarea bazei de date, adică realizarea operațiunilor de scriere sau citire, se rezumă la apelarea anumitor metode ale obiectului `session` din clasa `Database` după cum urmează în figura de mai jos.

```

24
25     @staticmethod
26     def get_active_user(self):
27         return Database.session.query(User).filter_by(isActive='1').first()
28
29     @staticmethod
30     def set_active_user(self, user_active):
31         Database.get_active_user(None).isActive = 0
32         Database.session.query(User).filter_by(username=user_active).first().isActive = 1
33         Database.session.commit()

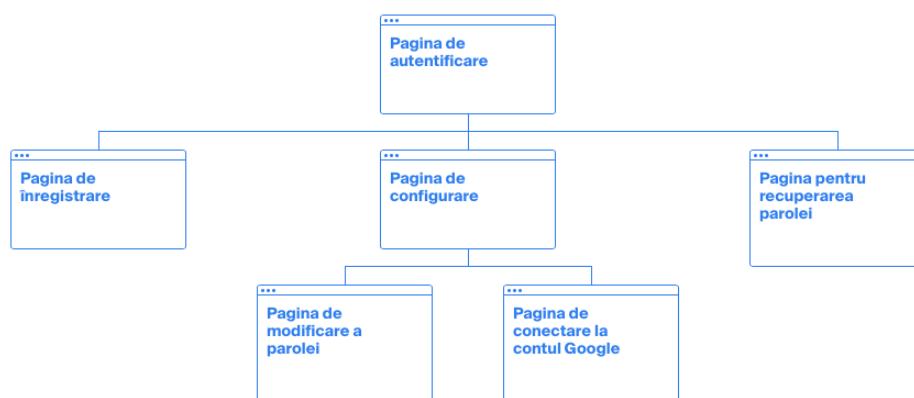
```

*Fig. 35 Utilizarea metodelor disponibile pentru aflarea sau setarea utilizatorului activ*

### 3.3 Implementarea saitului de configurare

Saitul web a fost implementat cu ajutorul librăriilor flask [30], folosită pentru crearea saitului și a utilizării acestuia, flask\_sqlalchemy [31] pentru conectarea cu baza de date, flask\_login [32] pentru a facilita funcționalitatea de autentificare, flask\_bootstrap [33] pentru a putea utiliza elemente Bootstrap și google\_auth\_oauthlib.flow pentru a realiza conectarea la contul Google al utilizatorului.

Pentru început se va prezenta harta saitului în figura de mai jos.



*Fig. 36 Harta saitului*

### 3.3.1 Implementarea aplicației Python pentru crearea saitului de configurare

Obiectul ce facilitează dezvoltarea unui sait utilizat în această lucrare este de tipul Flask. După declarare un prim pas este configurarea acestui obiect pentru a adăuga extensiile utilizate, precum conectarea cu o bază de date, utilizarea unui sistem de management al utilizatorilor autentificați și utilizarea librăriei online Bootstrap, conform codului sursă prezent în figura 37. Pentru punerea în funcțiune a saitului, cu alte cuvinte activarea serverului, este nevoie de apelarea metodei `run()` a obiectului Flask.

```

11 app = Flask(__name__)
12
13 app.config.from_object(configuration.Config)
14
15 database = SQLAlchemy(app)
16 migrate = Migrate(app, database)
17 login = LoginManager(app)
18 bootstrap = Bootstrap(app)
19 flow = Flow.from_client_secrets_file('credentials.json', app.config['GOOGLE_SCOPES'])
20 flow.redirect_uri = app.config['REDIRECT_URI']
21
22
23
24
25
26
27
28
29
30
31
Fig. 37 Pașii de configurare ai obiectului Flask

```

Fig. 38 Apelarea metodei `run()`

Pe lângă configurarea prin declararea altor obiecte compatibile cu Flask, în figură obiectele declarate având parametrul `app` în apel, a fost nevoie de implementarea și unei clase de configurare, mai exact clasa `Config`. Aceasta conține o cheie secretă pentru a securiza saitul, calea proiectului utilizată de multe ori de-a lungul rulării, calea absolută către baza de date, căi de redirectare utilizate în conectarea cu Google API, extensiile acceptate pentru pozele introduse de utilizator și subiectele știrilor ce pot fi afișate pe ecran.

```

18 class Config(object):
19     SECRET_KEY = '0123456789'
20     PROJECT_PATH = get_project_path()
21     SQLALCHEMY_DATABASE_URI = 'sqlite:///{} + PROJECT_PATH + '/config/mirrorDatabase.db'
22     SQLALCHEMY_TRACK_MODIFICATIONS = False
23     REDIRECT_URI = 'http://raspberry.tplinkdns.com:276/google'
24     GOOGLE_SCOPES = ['https://www.googleapis.com/auth/calendar.readonly',
25                      'https://www.googleapis.com/auth/gmail.readonly']
26     PICTURE_FOLDER = PROJECT_PATH + '/config/userPictures'
27     ALLOWED_EXTENSIONS = ['png', 'jpg', 'jpeg']
28     PICTURE_ID_FILENAME = get_number_of_pics() - 1 # .DS_Store reasons !!!!!
29     NEWS_WEBSITES = {
30         "Sport Intern": "https://www.agerpres.ro/sport-intern",
31         "Politica": "https://www.agerpres.ro/politica",

```

Fig. 39 Clasa utilizată pentru configurarea suplimentară

Pentru a se poate ajunge pe orice pagină din hartă este nevoie de funcția `învelitoare route()` cu parametrii acesteia fiind calea web și metodele acceptate (exemplu de folosire în figura de mai jos, linia 90), metodă tot a obiectului Flask, aplicată funcției care implementează funcționalitatea paginii descrise de calea web. În figura de mai jos este implementată calea "/main" a saitului local, pagină care se accesează după autentificarea unui utilizator, de aici

acesta își poate personaliza oglinda prin selecția subiectului știrilor (liniile 98 - 100 din fig. 40), furnizarea de poze pentru recunoașterea facială (liniile 102 - 118 din aceeași figură), schimba parola (liniile 120 - 121 din figura de mai jos) și butonul pentru integrarea contului personal Google.

```

90     @app.route('/main', methods=['GET', 'POST'])
91     @Login_required
92     def main():
93         form = forms.EditForm()
94
95         if form.is_submitted():
96             user = models.User.query.filter_by(username=current_user.username).first()
97
98             if form.newsTopic.data != user.newsTopic and form.newsTopic.data is not None:
99                 user.newsTopic = form.newsTopic.data
100                database.session.commit()
101
102             if 'files[]' in request.files:
103                 files = request.files.getlist('files[]')
104
105                 for file in files:
106                     if file and allowed_file(file.filename):
107                         filename = user.username + str(app.config['PICTURE_ID_FILENAME']) + '.' + \
108                             file.filename.rsplit('.', 1)[1].lower()
109                         filepath = os.path.join(app.config['PICTURE_FOLDER'], filename)
110
111                         file.save(filepath)
112
113                         pic = models.UserPicture(userId=user.id, picturePath=filepath, encoded=0)
114
115                         database.session.add(pic)
116                         database.session.commit()
117
118                         app.config['PICTURE_ID_FILENAME'] += 1
119
120             if form.changePassword.data:
121                 return redirect(url_for('confirmpass'))
122
123         return render_template('main.html', form=form, main=True, changed=True)
124
125     return render_template('main.html', form=form, main=True)

```

Fig. 40 Implementarea funcționalităților paginii de administrare

După cum se observă din figura 40, pe lângă funcția `route()` mai este folosită și o altă învelitoare, și anume `login_required`, pentru a ne asigura că utilizatorii neautentificați nu pot ajunge la această cale. Urmează implementarea funcționalităților în funcția `main()`, care, de obicei și de-a lungul implementării celorlalte căi, începe cu declararea formularului de afișat și se sfărșește prin a returna apelul funcției `render_template()` care afișează un model (concept descris în următorul subcapitol), în cazul prezentat se afișează modelul `main`, linia 125 din figura 40. O altă funcție folosită în a trimite utilizatorul pe o altă pagină a saitului este `redirect()` care are ca argument destinația pagina URL<sup>13</sup>. Pe lângă acești doi pași, se dă funcționalitate și în momentul în care utilizatorul trimite formularul spre aceeași cale pentru a fi prelucrat.

---

<sup>13</sup> Uniform Resource Locator, secvență de caractere standard, folosită pentru localizarea și identificarea unor resurse de pe Internet.

Formularele utilizate sunt de tipul FlaskForm, prezentat în figura 41, și acestea oferă metode precum `is_submitted()` pentru a verifica dacă se revine la aceeași cale în urma apăsării butonului de trimitere a cererii din formular, sau `validate_on_submit()` care pe lângă verificarea asemănătoare cu funcția precedentă se vor și valida datele introduse cu ajutorul unui validator declarat la nivelul clasei formularului. Principiile de folosire sunt aceleași la toate clasele de formulare, din acest motiv se va prezenta doar unul din toate implementate.

```

24 class RegistrationForm(FlaskForm):
25     username = StringField('Username', validators=[DataRequired()])
26     password = PasswordField('Password', validators=[DataRequired()])
27     password2 = PasswordField('Repeat Password', validators=[DataRequired(), EqualTo('password')])
28     submit = SubmitField('Register')
29
30
31 class EditForm(FlaskForm):
32     changePassword = BooleanField('Check this if you want to change your password')
33     newsTopic = SelectField('News Topic', choices=get_choices())
34     submitChanges = SubmitField("Submit changes")

```

Fig. 41 Implementarea claselor utilizate pentru formulare

Acstea clase ce moștenesc FlaskForm și care formează formularele propriu zise, au declarate câmpurile folosite pentru introducerea informațiilor, cum ar fi un nume de utilizator sau parola contului. Fiecare formular are și un câmp de tipul SubmitField care reprezintă butonul de trimitere a cererii.

Această aplicație se folosește de aceeași bază de date utilizată în implementarea precedentei secțiuni. Principiile de conectare și folosire sunt identice cu cele prezentate în subcapitolul 3.2.3. Diferența constă în moștenirea a încă unei clase, UserMixin, în declararea clasei ce mimică tabelul userData și a implementării a două funcții suplimentare de setare sau verificare a parolei. Această moștenire este folosită pentru managementul utilizatorului autentificat. Parolele utilizatorilor în momentul înregistrării sunt salvate în mod criptat, criptare asigurată de librăria werkzeug.security. [34]

Pentru a se asigura conectarea sistemului la contul Google al utilizatorului s-a utilizat librăria `google_auth_oauthlib.flow`. Pentru a începe conectarea este nevoie de apăsarea butonului "Click here to integrate with Google", fapt ce va duce la apelarea funcției atribuită căii "/google" prezentă în figura 42.

```

146     @app.route('/google', methods=['GET', 'POST'])
147     @login_required
148     def google():
149         if 'code' not in request.args:
150             auth_uri, _ = flow.authorization_url()
151             return redirect(auth_uri)
152
153         else:
154             user = models.User.query.filter_by(username=current_user.username).first()
155
156             if not user.googleToken:
157                 flow.fetch_token(code=request.values['code'])
158
159                 user.googleToken = app.config['PROJECT_PATH'] + '/config/userCredentials/' + current_user.username
160
161                 database.session.commit()
162                 pickle.dump(flow.credentials, open(user.googleToken, mode='wb'))
163
164             return redirect(url_for('main'))

```

Fig. 42 Funcția căii "/google" denumită intuitiv google()

În momentul apelării funcției datorită butonului, în argumentele cererii nu se vor afla date în câmpul code, ceea ce va conduce la utilizarea obiectului denumit flow de tip Flow (declararea acestuia se poate vedea în figura 37 la linia 19). Această utilizare constă în generarea linkului de conectare la Google a utilizatorului. Apoi utilizatorul este trimis la acest link. Utilizatorul este primit de o interfață cunoscută de la Google și este rugat să își introducă datele de conectare la contul personal. Apoi după ce a citit drepturile ce i le acordă lucrării de față, se apasă butonul de acceptare care trimită utilizatorul înapoi spre calea "/google".

În acest moment, în cererea ce a venit spre server, cea de la Google, are câmpul code fapt ce duce la rularea celeilalte părți a structurii if. Acest câmp este prelucrat de către obiectul de tip Flow și se generează calea fișierului în care vor fi salvate binar obiectul de conectare pentru utilizări ulterioare salvându-se în baza de date. La final se salvează în fișier obiectul și utilizatorul este trimis spre pagina de configurare a contului.

### 3.3.2 Structura codului sursă HTML în tandem cu Jinja

Datorită utilizării tehnologiei Jinja, realizarea documentelor HTML care formează saitul pentru configurarea sistemului de afișare a informațiilor personalizate a fost eficientizată prin conceptul de modelare. Astfel s-a putut crea o pagină web de bază care conține mai multe secțiuni care pot fi extinse ulterior. Această pagină de bază este o extindere a altrei baze din

librăria flask\_bootstrap pentru a fi folosite tehnologii terțe de stilizare. Prima secțiune, formată la rândul ei din alte două blocuri, conține un titlu și o bară de navigare, ambele fiind dinamice. Următoarea secțiune este folosită pentru a se putea insera alte funcționalități în documente ce moștenesc documentul de bază.

```

1  {% extends 'bootstrap/base.html' %}

2  {% block title %}
3      Mirror configuration
4  {% endblock %}

5  {% block navbar %}
6      <nav class="navbar bg-light">
7          <span class="navbar-brand text-dark" style="...">
8              {% if current_user.is_anonymous %}
9                  Welcome to your mirror configuration website
10             {% else %}
11                 {{ current_user.username.capitalize() }}'s mirror settings
12             {% endif %}
13         </span>
14         <ul class="navbar-nav">
15             <div class="btn-group btn-light">
16                 {% if not main %}
17                     {% if current_user.is_anonymous %}
18                         <button type="button" class="btn btn-secondary">
19                             <a class="nav-link text-light" href="{{ url_for('index') }}> Login </a>
20                         </button>
21                         <button type="button" class="btn btn-secondary">
22                             <a class="nav-link text-light" href="{{ url_for('forgotpass') }}> Forgot password? </a>
23                         </button>
24                     {% endif %}
25                 {% endif %}
26                 {% if not logout %}
27                     {% if not current_user.is_anonymous %}
28                         <button type="button" class="btn btn-secondary">
29                             <a class="nav-link text-light" href="{{ url_for('logout') }}> Logout </a>
30                         </button>
31                     {% endif %}
32                 {% endif %}
33             {% endif %}
34             {% if not signup %}
35                 {% if current_user.is_anonymous %}
36                     <button type="button" class="btn btn-secondary">
37                         <a class="nav-link text-light" href="{{ url_for('signup') }}> Register </a>
38                     </button>
39                 {% endif %}
40             {% endif %}
41         </div>
42     </ul>
43  </nav>
44  {% endblock %}

45  {% block content %}
46      <br style="...">
47      {% block app_content %}
48      {% endblock %}
49  <br style="...">
50  {% endblock %}
51  <br style="...">
52  {% endblock %}

```

Fig. 43 Modelul de bază al saitului

Blocurile ce formează secțiunile descrise mai sus sunt declarate cu ajutorul cuvântului cheie `block` și încheiate cu `endblock` având o sintaxă anume. Dinamicitatea titlului și a bării de navigare este dată de comenzi de tip `if`, interpretate de Jinja, unde se verifică starea utilizatorului curent (autentificat sau nu) sau existenței anumitor obiecte care semnalizează care moștenitor se folosește de bază, în figura 43 la liniile 10, 18, 19, 28, 29, 35 și 36.

Toate paginile prezente în harta saitului din figura 36 moștenesc acest model de bază, acestea fiind o extensie la blocul `content` din a doua secțiune a acestuia. Tehnologia folosită cel mai des în realizarea paginilor web este cea a formularelor, practic acțiunile de autentificare, deautentificare și schimbare a parolei sau a configurațiilor sistemului sunt implementate cu

ajutorul acestora. Din acest motiv se va prezenta un model drept exemplu (cel din figura de mai jos), celelalte folosind aceleasi principii.

```

1  {% extends "base.html" %} 
2
3  {% block app_content %} 
4      <div class="container center-block mx-auto" style="...>
5          <h1> Sign in </h1>
6
7          <form action="" method="post" novalidate>
8              {{ form.hidden_tag() }} 
9
10         <div class="form-group">
11             {{ form.username.label(class_="form-control-label") }} 
12
13             <div class="input-group mb-3">
14                 {{ form.username(size=32, class_="form-control") }} 
15
16                 {{ if form.username.errors %}}
17                     <div class="input-group-append">
18                         {{ for error in form.username.errors %}}
19                             <span class="input-group-text" style="...> {{error}} </span>
20                         {{ endfor %}} 
21
22                     </div>
23                 {{ endif }} 
24
25             </div>
26         </div>
27
28         <div class="form-group">
29             {{ form.password.label(class_="form-control-label") }} 
30
31             <div class="input-group mb-3">
32                 {{ form.password(size=32, class_="form-control") }} 
33
34                 {{ if form.password.errors %}}
35                     <div class="input-group-append">
36                         {{ for error in form.password.errors %}}
37                             <span class="input-group-text" style="...> {{error}} </span>
38                         {{ endfor %}} 
39
40                     </div>
41                 {{ endif }} 
42
43             </div>
44         </div>
45
46         <div class="form-group form-check">
47             {{ form.remember_me(class_="form-check-input") }} 
48             {{ form.remember_me.label(class_="form-check-label") }} 
49
50         </div>
51
52         <div class="form-group">
53             {{ form.submit(class_="btn btn-primary") }} 
54
55         </div>
56     </form>
57
58 {% endblock %}
```

Fig. 44 Modelul folosit pentru pagina de autentificare

Structurarea documentului HTML s-a realizat utilizând etichete de tip `div` având anumite stiluri din Bootstrap atașate. În cadrul acestor `div`-uri, formularele sunt afișate apelând la câmpurile din formularul trimis acestui model prin intermediul funcției `render_template()`. În cazul existenței erorilor în formulare, din cauza validatoarelor, acestea sunt parcurse pentru fiecare câmp și afișate (de exemplu liniile 35 - 42 din figura 44).

Ocazional, în funcție de necesitate, sunt folosite și alte etichete HTML cum ar fi cele de tip `input` pentru a se putea furniza poze de cu utilizatorul pentru recunoașterea facială, de tip

a pentru a afișa linkuri spre alte pagini sau label pentru a afișa un text sub o anumită formatare.

### 3.3.3 Imagini ale paginilor saitului

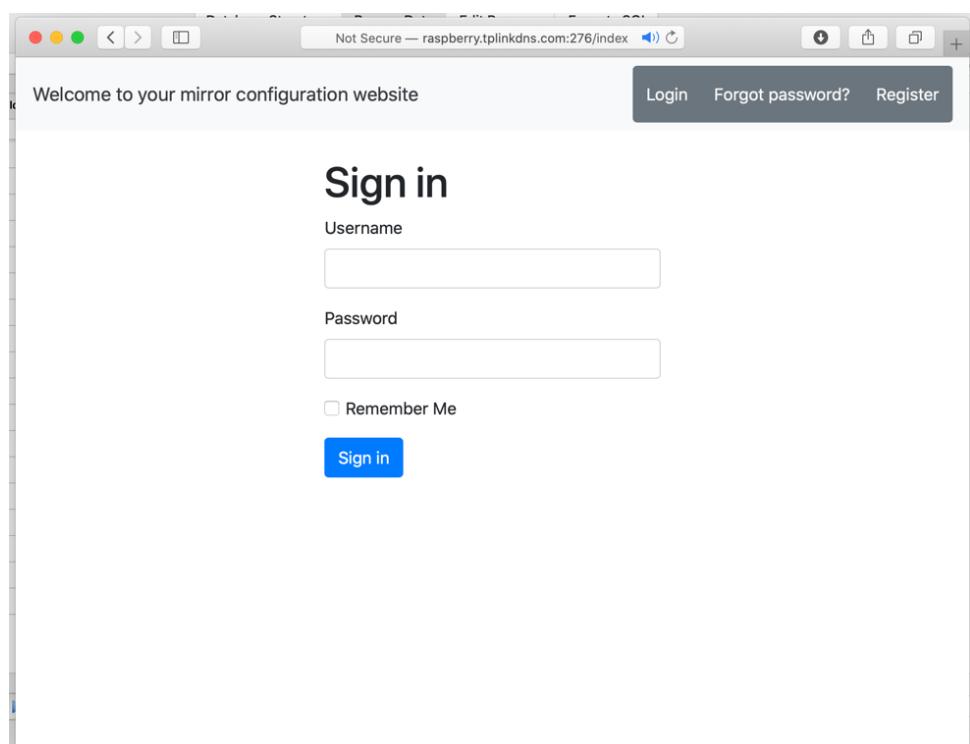


Fig. 45 Pagina de autentificare

The screenshot shows a web browser window with the URL `Not Secure — raspberry.tplinkdns.com:276/signups`. The page title is "Welcome to your mirror configuration website". At the top right are links for "Login" and "Forgot password?". The main heading is "Register". Below it are three input fields: "Username", "Password", and "Repeat Password", each with a corresponding text input box. A blue "Register" button is located at the bottom left of the form area.

Fig. 46 Pagina de înregistrare

The screenshot shows a web browser window with the URL `Not Secure — raspberry.tplinkdns.com:276/forgot`. The page title is "Welcome to your mirror configuration website". At the top right are links for "Login", "Forgot password?", and "Register". The main heading is "Forgotten Password". Below it are two input fields: "Username" and "New Password", each with a corresponding text input box. A blue "Submit password" button is located at the bottom left of the form area.

Fig. 47 Pagina de recuperare a parolei uitate

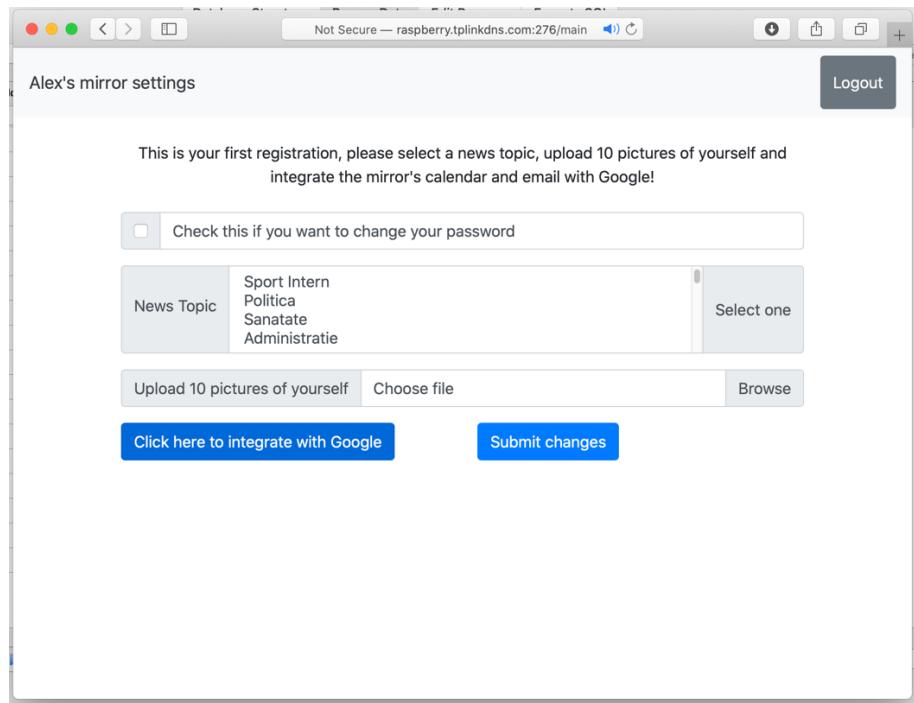


Fig. 48 Pagina de configurare, varianta la prima autentificare a unui utilizator nou

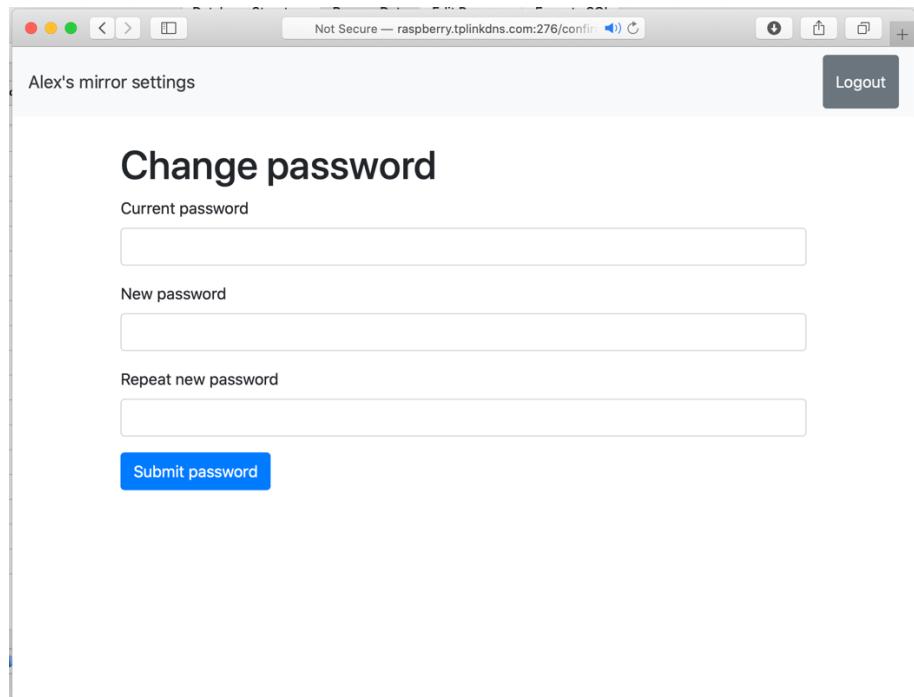


Fig. 49 Pagina de schimbare a parolei

## 4 Utilizarea sistemului informatic de afișare a informațiilor personalizate pe un ecran integrat în oglindă

Utilizarea presupune în primă fază crearea oglinzi (exemplu de creație fiind prezent în figura 6). Acest proces rămâne la atitudinea utilizatorului, importantă este folosirea unei oglinzi speciale ce permite trecerea prin spatele ei a unei părți a luminii ecranului. O multitudine de variante de implementări hardware sunt disponibile pe internet la o simplă căutare.

Din momentul pornirii sistemului software, aplicația cu interfață grafică va afișa modulul de ceas, vreme și știrile configurate în contul "Default", aceasta neavând alți utilizatori înregistrați.

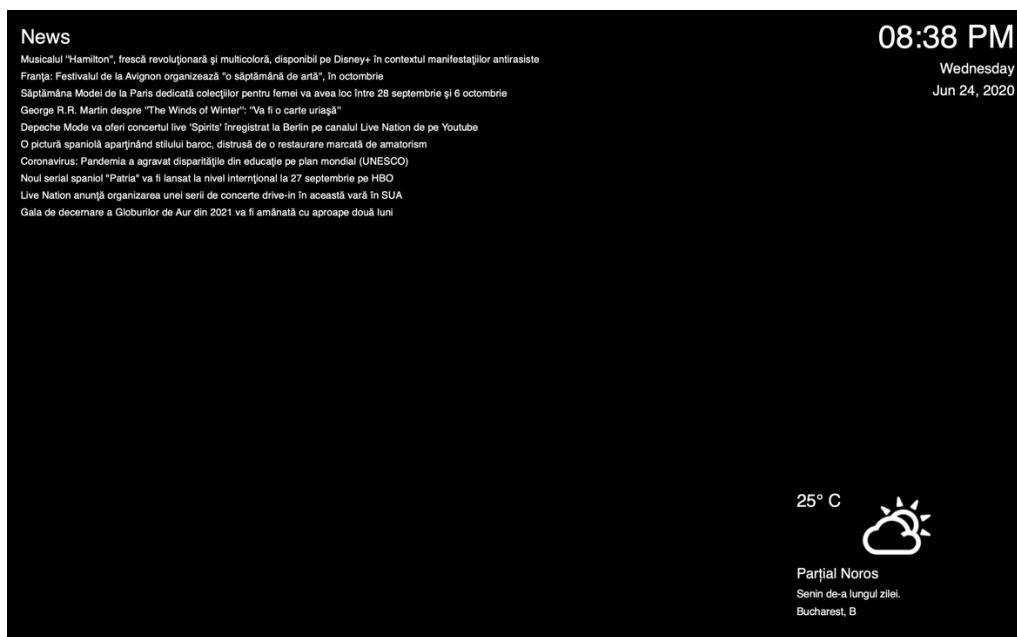


Fig. 50 Interfața grafică al contului "Default"

Este nevoie să se folosească un dispozitiv terț (telefon inteligent, laptop, calculator) pentru ca un utilizator nou să se înregistreze. Acest dispozitiv trebuie conectat la aceeași rețea de Wi-Fi ca oglinda. Se accesează site-ul, se creează un cont (figura 45) și se configurează după bunul plac (figura 48).

Oferta de configurare este compusă din :

- posibilitatea schimbării parolei;
- alegerea unui subiect de știri sau schimbarea celui deja setat;
- încărcarea de poze pentru recunoașterea facială;

- conectarea cu contul personal Google al utilizatorului;

După momentul configurării, sistemul va funcționa după parametrii descriși în capitolele precedente, utilizatorul oricând având ocazia unei modificări a configurațiilor. Oglinda poate fi pusă unde este nevoie și va funcționa după specificațiile precizate.

## 5 Concluzii

Așadar, în concluzie, lucrarea de față își propune a digitaliza un obiect folosit zi de zi și anume oglinda. Să afișeze informații ale multipli utilizatori folosind recunoașterea facială pentru a mai economisi timp, resursă care devine din ce în ce mai scumpă.

Această lucrare este un exemplu al tendințelor de conectare la Internet a cât mai multor obiecte utilizate frecvent, cum ar fi aragazul, frigiderul sau chiar mașina, tendință ce poartă numele de Internet of Things, sau Internetul Lucrurilor.

De-a lungul elaborării au fost întâmpinate o multitudine de dificultăți, de la familiarizarea unui nou limbaj de programare, anume Python, până la realizarea arhitecturii întregului sistem. O problemă ce a persistat mai mult a fost conectarea cu Google API. Documentația era foarte succintă și codul sursă exemplu era gândit doar pe o anumită situație, fapt ce a dus la multe sesiuni de încercări a diferite metode din obiectul de tip Flow.

### 5.1 Îmbunătățiri viitoare

Îmbunătățirile posibile se pot aduce foarte mult pe partea informațiilor afișate, anume să se ofere posibilitatea conectării și a conturilor de la Apple, pentru emailuri și calendar, sau Yahoo, pentru emailuri, o gamă mai largă de surse pentru știri, anume surse internaționale, și mai multe surse locale.

Îmbunătățiri ar putea veni și pe partea de sait, oferindu-i o mai mare flexibilitate utilizatorului, cum ar fi să vadă cine se află în fața oglinziei cu ajutorul senzorului optic, desigur cu acordul prealabil al celui ce folosește oglinda în momentul accesării camerei. O stilizare a saitului mai bună este o îmbunătățire, cum ar fi compatibilitatea interfeței web cu mai multe rezoluții de navigator web, la momentul prezentării acest aspect nefiind tratat.

Telefonul personal ar putea fi implicat în funcționarea oglinziei, anume prin accesarea notificărilor telefonului de către oglindă și să fie afișate, sau de pe laptopul personal.

## Bibliografie

- [1] „Glass,” Wikipedia, [Online]. Disponibil la: <https://en.wikipedia.org/wiki/Glass#History>. [Accesat 20 05 2020].
- [2] „Mirror,” Wikipedia, [Online]. Available: <https://en.wikipedia.org/wiki/Mirror>. [Accesat 20 05 2020].
- [3] „Microcontroller,” Wikipedia, [Online]. Disponibil la: <https://en.wikipedia.org/wiki/Microcontroller>. [Accesat 20 05 2020].
- [4] „Transistor count,” Wikipedia, [Online]. Disponibil la: [https://en.wikipedia.org/wiki/Transistor\\_count](https://en.wikipedia.org/wiki/Transistor_count). [Accesat 21 05 2020].
- [5] „Internet of things,” Wikipedia, [Online]. Disponibil la: [https://en.wikipedia.org/wiki/Internet\\_of\\_things](https://en.wikipedia.org/wiki/Internet_of_things). [Accesat 21 05 2020].
- [6] „MirrorVue Mirror,” EverVue, [Online]. Disponibil la: <https://www.evervuetv.com/mirrorvue/>. [Accesat 21 05 2020].
- [7] „The Smart Mirror Embrace™ is an Android™ Touchscreen Smart Mirror,” Embrace, [Online]. Disponibil la: <https://www.embracesmartmirror.com/shop/embrace-smart-mirror/>. [Accesat 21 05 2020].
- [8] „HDMI,” Wikipedia, [Online]. Available: <https://en.wikipedia.org/wiki/HDMI>. [Accesat 22 05 2020].
- [9] „Raspberry Pi,” Wikipedia, [Online]. Disponibil la: [https://en.wikipedia.org/wiki/Raspberry\\_Pi](https://en.wikipedia.org/wiki/Raspberry_Pi). [Accesat 22 05 2020].
- [10] „Raspberry Pi 4 Model B specifications,” Raspberry Pi Foundation, [Online]. Disponibil la: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/>.
- [11] „Buy a Raspberry Pi,” Raspberry Pi Foundation, [Online]. Disponibil la: <https://www.raspberrypi.org/products/>. [Accesat 22 05 2020].
- [12] „Microsoft Lifecam HD-3000 First Looks - Review 2011,” PCMag UK, [Online]. Disponibil la: <https://uk.pcmag.com/webcams/20973/microsoft-lifecam-hd-3000>. [Accesat 22 05 2020].
- [13] „List of Python software,” Wikipedia, [Online]. Disponibil la: [https://en.wikipedia.org/wiki/List\\_of\\_Python\\_software#Python\\_implementations](https://en.wikipedia.org/wiki/List_of_Python_software#Python_implementations). [Accesat 23 05 2020].
- [14] „Python,” Wikipedia, [Online]. Disponibil la: [https://en.wikipedia.org/wiki/Python\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)). [Accesat 23 05 2020].
- [15] „HTML,” Wikipedia, [Online]. Disponibil la: <https://en.wikipedia.org/wiki/HTML>. [Accesat 23 05 2020].
- [16] „Jinja,” Wikipedia, [Online]. Disponibil la: [https://en.wikipedia.org/wiki/Jinja\\_\(template\\_engine\)](https://en.wikipedia.org/wiki/Jinja_(template_engine)). [Accesat 23 05 2020].
- [17] „SQL,” Wikipedia, [Online]. Disponibil la: <https://ro.wikipedia.org/wiki/SQL>. [Accessed 23 05 2020].
- [18] „PyCharm,” Wikipedia, [Online]. Disponibil la: <https://en.wikipedia.org/wiki/PyCharm>. [Accesat 24 05 2020].
- [19] „SQLite,” Wikipedia, [Online]. Disponibil la: <https://en.wikipedia.org/wiki/SQLite>. [Accesat 24 05 2020].

- [20] „GitHub,” Wikipedia, [Online]. Disponibil la: <https://en.wikipedia.org/wiki/GitHub>. [Accesat 24 05 2020].
- [21] „tkinter — Python interface to Tcl/Tk — Python 3.8.3 documentation,” Python Software Foundation, [Online]. Disponibil la: <https://docs.python.org/3/library/tkinter.html>. [Accesat 13 02 2020].
- [22] „API Reference - Gmail API - Google Developers,” Google, [Online]. Disponibil la: <https://developers.google.com/gmail/api/v1/reference>. [Accesat 24 03 2020].
- [23] „API Reference - Calendar API - Google Developers,” Google, [Online]. Disponibil la: <https://developers.google.com/calendar/v3/reference>. [Accesat 10 03 2020].
- [24] „Dark Sky API: Documentation Overview,” Dark Sky, [Online]. Disponibil la: <https://darksky.net/dev/docs>. [Accesat 28 02 2020].
- [25] „API Documentation,” ipstack, [Online]. Disponibil la: <https://ipstack.com/documentation>. [Accesat 24 02 2020].
- [26] „Requests: HTTP for Humans™ — Requests 2.24.0 documentation,” Requests, [Online]. Disponibil la: <https://requests.readthedocs.io/en/master/>. [Accesat 30 11 2019].
- [27] L. Richardson, „Beautiful Soup Documentation — Beautiful Soup 4.9.0 documentation,” [Online]. Disponibil la: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/> . [Accesat 30 11 2019].
- [28] A. Geitgey, „GitHub - ageitgey/face\_recognition: The world's simplest facial recognition api for Python and the command line,” GitHub, [Online]. Disponibil la: [https://github.com/ageitgey/face\\_recognition](https://github.com/ageitgey/face_recognition). [Accesat 23 04 2020].
- [29] „ SQLAlchemy 1.3 Documentation,” SQLAlchemy, [Online]. Disponibil la: <https://docs.sqlalchemy.org/en/13/>. [Accesat 29 02 2020].
- [30] „Flask Documentation (1.1.x),” Pallets, [Online]. Disponibil la: <https://flask.palletsprojects.com/en/1.1.x/>. [Accesat 24 03 2020].
- [31] „Flask-SQLAlchemy Documentation (2.x),” Pallets, [Online]. Disponibil la: <https://flask-sqlalchemy.palletsprojects.com/en/2.x/>. [Accesat 26 03 2020].
- [32] „Flask-Login 0.4.1 documentation,” Pallets, [Online]. Disponibil la: <https://flask-login.readthedocs.io/en/latest/>. [Accesat 10 04 2020].
- [33] M. Nrinkmann, „Flask-Bootstrap 3.3.7.1 documentation,” [Online]. Disponibil la: <https://pythonhosted.org/Flask-Bootstrap/>. [Accesat 03 04 2020].
- [34] „Django (web framework),” Wikipedia, [Online]. Disponibil la: [https://ro.wikipedia.org/wiki/Django\\_\(web\\_framework\)](https://ro.wikipedia.org/wiki/Django_(web_framework)). [Accesat 23 05 2020].
- [35] „SQL-92,” Wikipedia, [Online]. Disponibil la: <https://en.wikipedia.org/wiki/SQL-92>. [Accesat 24 05 2020].
- [36] „Histogram of oriented gradients,” Wikipedia, [Online]. Disponibil la: [https://en.wikipedia.org/wiki/Histogram\\_of\\_oriented\\_gradients](https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients). [Accesat 24 05 2020].

**DECLARAȚIE DE AUTENTICITATE A LUCRĂRII  
DE FINALIZARE A STUDIILOR<sup>1</sup>**

Subsemnatul (a) MIREA ANDREI SERGIU SOFRIN, legitimat (ă)  
cu CI / BI seria TZ nr. 372071, CNP 1570627350041  
autorul lucrării MIRRORVIEW SISTEM DE VIZUALIZARE  
A INFORMAȚIILOR PERSONALIZATE PE UN ECRAN  
INTEGRAT ÎN OGLENDĂ  
elaborată în vederea susținerii examenului de finalizare a studiilor de LICENȚĂ  
organizat de către Facultatea  
DE AUTOMATICA și CALCULATOARE din cadrul  
Universității Politehnica Timișoara, sesiunea IUNIE a anului  
universitar 2019 - 2020, luând în considerare conținutul art. 39 din  
RODPI – UPT, declar pe proprie răspundere, că această lucrare este rezultatul propriei  
activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost  
folosite cu respectarea legislației române și a convențiilor internaționale privind  
drepturile de autor.

Timișoara,

Data

25.06.2020

Semnătura

<sup>1</sup> Declarația se completează „de mână” și se inserează în lucrarea de finalizare a studiilor, la sfârșitul acesteia, ca parte integrantă.