

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI  
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

**Be Present**  
**- Aplicație de strângere de fonduri -**

propusă de

***Ioana Turic***

**Sesiunea:** *februarie, 2020*

Coordonator științific

**Colab. Florin Olariu**

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI  
FACULTATEA DE INFORMATICĂ

# **Be Present**

## **- Aplicație de strângere de fonduri -**

***Ioana Turic***

**Sesiunea:** *februarie, 2020*

Coordonator științific

***Colab. Florin Olariu***

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele \_\_\_\_\_

Data \_\_\_\_\_

Semnătura \_\_\_\_\_

### **DECLARAȚIE privind originalitatea conținutului lucrării de licență**

Subsemnatata Turic Ioana cu domiciliul în Jud. BT Mun. Dorohoi Str. Ștefan cel Mare nr.50 născută la data de 28.07.1997, identificat prin CNP 2970728071368, absolventă a Universității „Alexandru Ioan Cuza” din Iași, Facultatea de Informatică specializarea Informatică, promoția 2020, declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art.143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul: Be Present - Aplicație de strângere de fonduri - elaborată sub îndrumarea dl. colab. Florin Olariu, pe care urmează să o susțină în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data azi, .....

Semnătură student .....

## DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Be Present - Aplicație de strângere de fonduri -*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, *data*

Absolvent *Ioana Turic*

---

(semnătura în original)

## Cuprins

1. Arhitectura și elemente de structură	8
1.1. Arhitectura aplicației	10
2. Sistemul de autentificare și autorizare	14
2.1. Autentificarea	15
2.2. Autorizarea	17
2.3. Utilizarea JWT în contextul aplicației	19
3. Principalele flow-uri în funcție de rolul utilizatorului	21
4. Procesarea imaginilor în contextul anonimizării persoanelor	25
4.1. Procesarea imaginii la nivel de client și trimiterea către API	25
4.2. Recunoașterea facială în imagini	28
4.3. Aplicarea GaussianBlur	29
4.4. Concluzie	30
5. Concluzii și direcții ulterioare	31
6. Bibliografie	32
7. Anexe	33

## **Introducere**

### **Motivația**

În ultima perioadă societatea noastră s-a dezvoltat destul de mult în direcția întraajutorării, oamenii nu rămân indiferenți la nevoile altor oameni, ale mediului înconjurător și la situațiile neprevăzute care provoacă dezastre naturale, dorindu-și în permanență să ajute la rezolvarea problemelor. Contextul tehnologic în care ne aflăm pune la dispoziție o serie de modalități eficiente, accesibile tuturor pentru desfășurarea actelor de bine-facere. Ideea vine din necesitatea existenței unei astfel de platforme care să faciliteze procesul de donație pentru diferite cauze caritabile existente în cadrul aplicației. Aceasta se adresează unei categorii largi de utilizatori, de la persoane care își doresc să doneze până la persoane care pot crea o cauză caritabilă pentru care stabilesc un scop monetar.

Pentru a încuraja donațiile către anumite cauze, sistemul pune la dispoziție înregistrarea unei categorii speciale de utilizatori, și anume companiile, care oferă vouchere în schimbul unei sume minime donate către cauzele pe care acestea le susțin. În acest mod, aplicația de strângere de fonduri reușește să reunească sub aceeași platformă persoane venite din arii de dezvoltare diferite, care însă au un scop comun, acela de a ajuta.

### **Gradul de noutate al temei**

Din punctul de vedere al implementărilor anterioare, această tematică este abordată de câteva asociații umanitare, însă cauzele caritabile expuse aparțin de cele mai multe ori asociației, sau sunt asociate cu o zi importantă din viața noastră, cum ar fi ziua de naștere. Dar de ce să nu promovăm astfel de activități în fiecare zi? Scopul aplicației de strângere de fonduri este acela de a oferi acces tuturor asociațiilor de acest tip, pentru a-și face vizibile cauzele pentru care luptă. Mai mult de atât participarea diverselor companii în tot acest proces aduce un avantaj important în atingerea țelurilor celor care inițiază strângerile de fonduri. Voucherele reprezintă și ele un element de noutate în rândul aplicațiilor de acest gen, deoarece prezența lor dovedește implicarea activă a companiilor în viața celor defavorizați și stimulează donațiile prin oferirea de discounturi în schimbul unei sume minime donate.

Aplicația se aliniază cu normele impuse de GDPR și va proteja datele cu caracter personal, cum sunt pozele ce pot fi afișate pentru un caz caritabil prin blur-area fețelor persoanelor din poze. Mai mult de atât, introducerea datelor cu caracter personal nu este obligatorie, iar odată introduse, persoana își oferă consimțământul cu privire la prelucrarea acestora.

### **Obiective generale și metodologiile folosite**

Folosind tehnologii de actualitate, aplicația de strângere de fonduri are principalul obiectiv de a facilita procesul de donare aducându-l într-un mediu virtual accesibil tuturor. Platforma se organizează sub forma unei aplicații web, care poate fi accesată cu ușurință prin intermediu URL-ului acesteia. Datorită utilizării în implementare a framework-ului de

JavaScript, Angular, aplicația este una modularizată atât pe partea de implementare, cât și pe partea vizuală oferind o experiență plăcută utilizatorilor ei. API-ul din spatele interfeței este implementat în framework-ul de Java, Spring Boot, și pune la dispoziție pe langa serviciile ce alcătuiesc core-ul aplicației și un sistem de autentificare și autorizare pe baza unor roluri prestabilite.

Ca și metodologie de organizare a timpului și funcționalităților implementate am abordat metodologia Agile, astfel am început cu funcționalitățile de bază ale aplicației, continuând cu adăugarea pe parcurs a altor funcționalități.

### **Descrierea sumară a soluției**

Deoarece o astfel de aplicație necesită un bun management al userilor, permisiunilor acestora și al acțiunilor pe care le pot iniția în cadrul platformei, structura acesteia conține 2 părți: serverul expus printr-un API și clientul care îl consumă. De asemenea, pentru un bun control al datelor care sunt înregistrate sau expuse de către aplicație a fost necesară utilizarea unei baze de date relaționale, fapt pentru care am ales să utilizez PostgreSQL. Autentificarea și autorizarea le-am realizat cu Spring Security folosind JWT (JSON Web Token) și rolurile definite la nivel de aplicație. Pentru structurarea și implementarea aplicației am utilizat framework-ul de Java, Spring Boot alături de Maven, deoarece managementul dependențelor este unul foarte eficient, și este promovat principiul IoC (Inversion of Control). Pentru procesarea imaginilor astfel încât să respecte normele impuse de GDPR am utilizat biblioteca OpenCV.

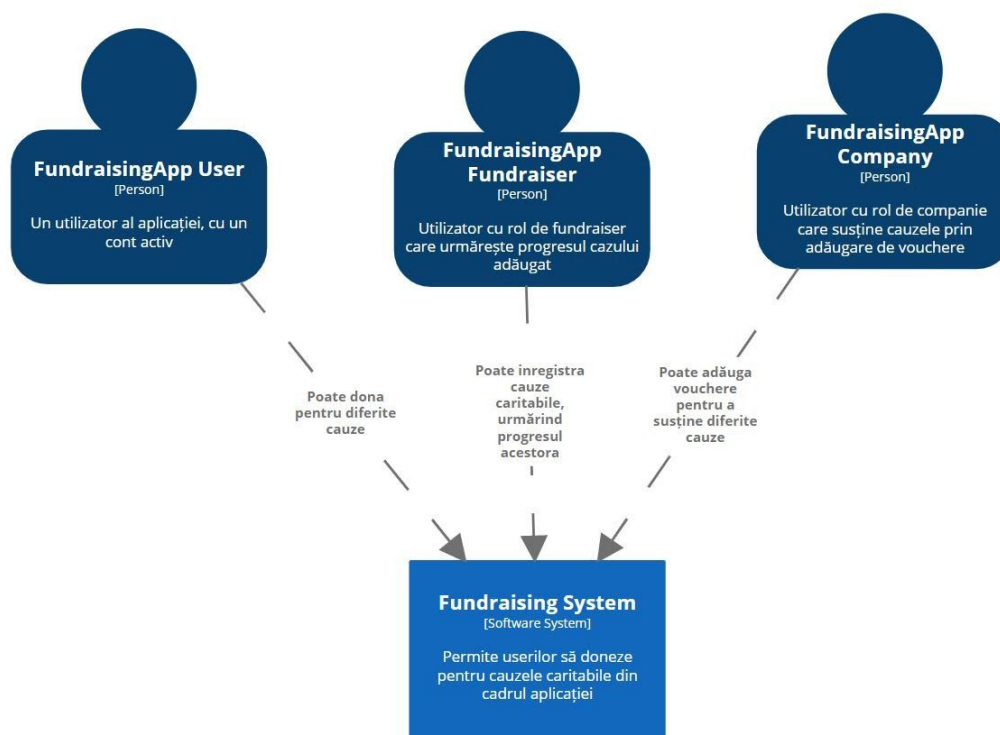
### **Structura lucrării**

Lucrarea este structurată pe 4 capitole care descriu demersul întreprins pentru a atinge scopul propus, și anume acela de a implementa o platformă eficientă de donații care să respecte prevederile cu referire la GDPR. Primul capitol descrie din punct de vedere arhitectural atât API-ul, clientul ce îl consumă cât și relațiile de la nivelul bazei de date. Al doilea capitol cuprinde o analiză a modulului de autorizare și autentificare, a abordărilor folosite în implementarea acestuia alături de motivația utilizării acestor abordări. În al treilea capitol accentul cade pe logica aplicației, astfel vor fi prezentate principalele flow-uri pe care utilizatorii le pot realiza. Capitolul al patrulea vine cu soluții în ceea ce privește problema asigurării anonimității persoanelor, descriind întregul proces pe care îl parcurge o imagine cu fața unei persoane de la client, la API și înapoi la client pentru a fi afișată cazului asociat.

## 1. Arhitectura și elemente de structură

Privind către scopul aplicației și dorința ca aceasta să fie una ușor utilizabilă de către toți stakeholderii ei, o abstractizare a întregului context este necesară pentru a avea o privire de ansamblu asupra sistem software.

Am considerat că cea mai eficientă modalitate de a ilustra abstractizarea sistemului software se realizează prin utilizarea diagramelor structurale C4 models. În C4 model ierarhia de abstractizări poate fi urmărită prin crearea unei colecții care să conțină Context, Container, Componenta și Cod (Mai multe informații precum și definițiile termenilor utilizați pot fi regăsite în Anexe). În cele ce urmează voi exemplifica aplicarea diagramelor C4 model de nivelul 1 și nivelul 2 asupra sistemului software al aplicației de strângere de fonduri.



System Context diagram for Fundraising System

Workspace last modified: Mon Jan 27 2020 11:41:43 GMT+0200 (Eastern European Standard Time)

Fig. 1.1 Diagrama C4 model nivelul 1

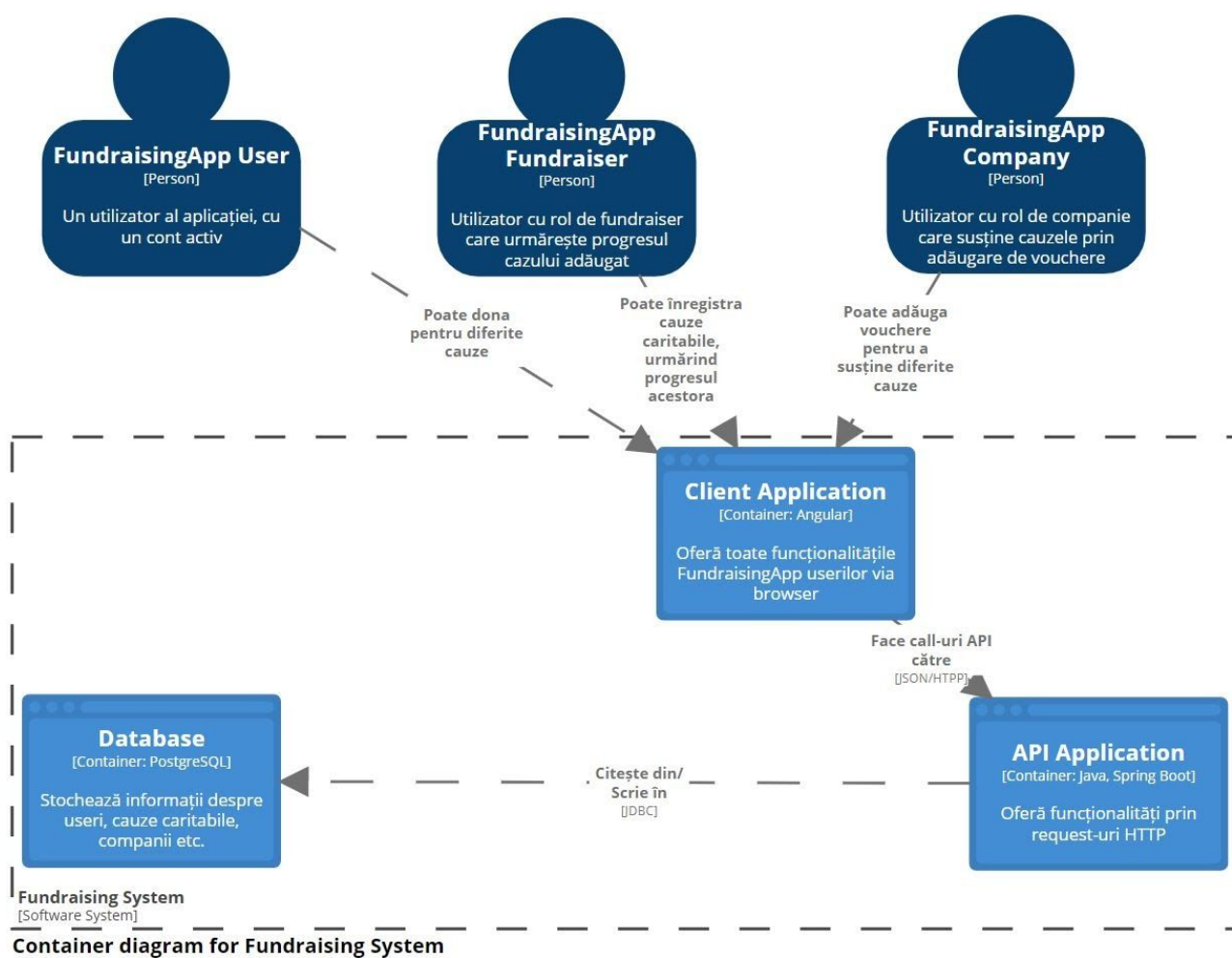
Source: <https://structurizr.com/workspace/50453/workspace-editor#views>

În Fig. 1.1 am reprezentat diagrama de nivelul 1 sau diagrama de context în care, după cum se poate observa și în imagine, focusul este asupra întregului sistem ca un tot unitar și



asupra utilizatorilor cu care acesta interacționează. Astfel că în centru se află întregul sistem, iar în jurul lui se află principalele categorii de utilizatori care interacționează cu sistemul prin diferite tipuri de relații. Scopul acestei diagrame este de a fi o reprezentare cât mai succintă a întregii aplicații fără a intra în detalii de implementare, și așa cum îi spune și numele, creează context întregului proces de dezvoltare.

Odată creat contextul de diagrama de nivelul 1, este necesară descrierea sistemului printr-o diagramă container sau diagrama de nivelul 2. În cadrul acesteia se abstractizează unitățile rulabile ale sistemului sub forma unor containere, care în acest caz coincid cu layerele aplicației, și tot în acest pas se definesc legăturile dintre acestea.



Workspace last modified: Mon Jan 27 2020 13:50:36 GMT+0200 (Eastern European Standard Time)

Fig. 1.2 Diagrama C4 model nivelul 2

Source: <https://structurizr.com/workspace/50453/workspace-editor#views>

Figura 1.2 surprinde diagrama de nivelul 2 pentru aplicația de strângere de fonduri, o reprezentare la un nivel mai înalt decât diagrama precedentă, care abstractizează arhitectura sistemului și distribuirea responsabilităților. În figura 1.2 se pot observa principalele containere ale aplicației: aplicația client, API-ul și baza de date unde sunt stocate toate entitățile, alături de relațiile dintre acestea.

Flow-ul descris este unul simplu, abstract, astfel încât orice user poate comunica cu aplicație client pentru a accesa anumite funcționalități din cadrul acesteia; aceasta la randul ei va face apeluri către API-ul expus de aplicația de backend prin request-uri HTTP care pot conține sau nu JSON-uri. Următorul layer, cu care comunică în mod direct API-ul, este baza de date în care se pot insera sau din care se pot aduce date.

Având o privire de ansamblu asupra platformei prin analiza celor 2 diagrame c4 model, în care am văzut la un nivel abstract cum se definește sistemul software și cum informația cerută/transmisă de stakeholderi circulă prin toate containerele, putem continua cu o analiză amănunțită a arhitecturii aplicației.

## 1.1. Arhitectura aplicației

Alegerea arhitecturală în ceea ce privește realizarea aplicației de strângere de fonduri a fost influențată de anvergura acesteia, de aceea, modelul arhitectural care se pliază pe necesități este unul monolitic. Așa cum îi spune și numele, arhitectura monolitică se bazează pe existența unui singur nivel care combină diferite componente sub același API.

Contextul actual dezvăluie existența a 4 componente: componenta de autorizare și autentificare, componenta responsabilă cu prezentarea, componenta care cuprinde logica aplicației, și componenta responsabilă cu layerul datelor. În continuare voi realiza o prezentare a fiecărei componente motivând alegerea tehnologiei cu care a fost implementată.

- a) **Componenta de autorizare și autentificare.** Autentificarea și autorizarea reprezintă un pas important în ceea ce privește securitatea datelor expuse de un API, întrucât doar persoanele a căror identitate și drepturi sunt cunoscute ar trebui să le manipuleze. Autentificarea se referă la validarea credențialelor (în acest caz e-mail și parolă) pentru a verifica identitatea userului și a-i oferi acces la anumite resurse. Tipul de autentificare utilizat este autentificarea cu un singur factor, astfel că este suficient ca un utilizator să introducă un email și o parolă corectă pentru a fi autentificat. Următorul pas în procesul de autentificare îl reprezintă generarea tokenului, deoarece se utilizează autentificarea bazată pe token. Pe baza tokenului generat, care conține și rolul utilizatorului în aplicație se realizează autorizarea, astfel că în funcție de rol, un utilizator are acces la anumite informații și poate altera doar anumite date. Acest subiect va fi reluat în Capitolul al 2-lea.
- b) **Componenta de prezentare** sau clientul care consumă API-ul este organizat sub forma unei aplicații web care folosește framework-ul de JavaScript, Angular versiunea 8.3.

Alegerea Angular ca și modalitate de a dezvolta aplicația pe partea de client a fost motivată de avantajele cu care vine acest framework. Printre acestea se numără ușurința navigării utilizatorilor, faptul că are o structură modularizată, securitate prin intermediul primitivelor și interfețelor din TypeScript, reutilizarea și lizibilitatea codului.

- c) **Componenta responsabilă de logica aplicației** sau API-ul este un proiect Maven scris în Java, Spring Boot, și cuprinde întreaga logică a aplicației, de la procesul de autentificare și autorizare până la adăugare de donații și vouchere. Am ales Maven datorită modului foarte eficient de a face management dependențelor prin existența POM-ului și a Maven Repository-ului local, datorită multitudinii de librării și plugin-uri puse la dispoziție de Central Repository și datorită integrării cu Git.

În alegerea framework-ului de Java am ținut cont de faptul că Spring Boot promovează eliminarea configurărilor XML, proprietățile sunt simplificate, iar bean-urile sunt inițializate, configurate și conectate automat.

- d) **Componenta responsabilă cu layerul datelor** este reprezentată de utilizarea bazei de date PostgreSQL, la care accesul este asigurat de Spring Data JPA. Integrarea acestor 2 componente este una intuitivă și ușor de realizat prin simpla specificare a proprietăților necesare pentru configurarea PostgreSQL în fișierul `application.properties`. Mai mult de atât un grad ridicat de eficiență este adus de modul în care JPA, prin utilizarea adnotărilor specifice, face maparea de la entitățile prezente în cod, la tabelele din baza de date și la relațiile dintre acestea. De asemenea, JPA oferă suport pentru un bun management al ciclului de viață al unei entități, iar wrapperul Spring Data JPA adaugă opțiunea metodelor fără implementare care creează query-uri la baza de date doar din numele metodei.

PostgreSQL reprezintă o bună opțiune în rândul bazelor de date deoarece se află în deplină concordanță cu proprietățile ACID, pune la dispoziție o gamă largă de tipuri de date care pot fi stocate, o interfață grafică, și este ușor integrabil cu Spring Data JPA.

Mergând mai departe către designul bazei de date asignat acestui proiect, este necesară o prezentare detaliată a schemei bazei de date. Pentru o mai bună înțelegere a entităților existente și relațiile dintre acestea, se va porni de la cel mai de jos nivel și anume nivelul de date.

În Fig. 1.3 este reprezentată schema bazei de date unde se pot observa principalele entități, tabelele de legătură alături de relațiile dintre acestea. Vom începe cu entitatea utilizată în principal în procesul de logare, și anume Users, care prin cele 2 proprietăți: Email și Password asigură identificarea unică a unui utilizator. În strânsă legătură de aceasta entitate este cea de Roles, ce definește rolul pe care-l poate avea un utilizator în cadrul aplicației.



O altă entitate care se află în strânsă legătură cu cea de Users, este entitatea Companies, relația dintre acestea fiind una de OneToMany. Entitatea Vouchers depinde de entitatea de Companies cu ajutorul unei relații de ManyToOne, deoarece o companie poate adăuga mai multe vouchere la diferite cauze caritabile, ceea ce implică și existența unei relații de ManyToOne cu entitatea Fundraisers.

## 2. Sistemul de autentificare și autorizare

Privind în mod general către ceea ce înseamnă autentificarea și autorizarea în aplicațiile web ne putem da seama de importanța acestora în contextul securității. Deși procesul pare intuitiv (autentificarea confirmă identitatea unui user pe baza unor credențiale, iar autorizarea oferă accesul la anumite resurse), acesta este, de fapt, unul complex ce necesită luarea în considerare a mai multor factori.

În ceea ce privește partea de autentificare, cele mai întâlnite abordări atunci când vine vorba de implementarea acestui sistem sunt următoarele<sup>[1]</sup>:

- HTTP Basic authentication;
- folosind cookie-uri;
- cu ajutorul token-urilor;
- semnăturile pe fiecare request;
- parola de unică folosință.

HTTP Basic authentication este cea mai simplă modalitate de a asigura autentificarea, fără a utiliza cookie-uri sau sesiuni. Pentru aceasta clientul trebuie să trimită în fiecare request headerul Authorization cu valoarea formată din username și parolă codificate în Base64 și prefixate de cuvântul cheie "Basic". Există câteva dezavantaje în folosirea acestei metode, cum ar fi: username-ul și parola sunt trimise cu fiecare request, userul nu se poate deloga, expirarea credențialelor se realizează numai la schimbarea parolei, iar credențialele pot fi aflate prin intermediul unui atac de securitate.

Metoda de autentificare prin cookie-uri trimite către server un header Set-Cookie, în urma unui request HTTP. Browserul stochează acest cookie, urmând ca acesta să fie trimis cu fiecare request făcut la aceeași origine ca în cookie HTTP header. Ce este de menționat în privința acestei metode este că la setarea cookie-urilor tot timpul trebuie folosit flag-ul HttpOnly (pentru a evita atacurile de tip XSS); în același timp cookie-urile trebuiesc semnate pentru ca eventualele modificări făcute de client să fie identificate de către server.

Cea mai întâlnită metodă de autentificare bazată pe token este utilizarea JWT (JSON Web Token). Acest token conține 3 părți: headerul care conține informații despre tipul tokenului și algoritmul de hash utilizat, payload-ul care conține credențialele și semnătura. Pentru a putea fi folosite, aceste token-uri trebuiesc stocate în memoria locală a browserului, fapt pentru care sunt predispuse către atacuri XSS (Cross Site scripting).

O altă modalitate de a asigura autentificarea este prin intermediul semnăturilor pentru fiecare request. Astfel, atunci când un consumator al unui API face un request, acesta trebuie să fie semnat, ceea ce înseamnă că întregul request va fi hash-uit folosind o cheie privată (folosind un algoritm de hash pe bază de cheie, precum HMAC - SHA256). Pentru calcularea hash-ului sunt necesare: o metoda HTTP, path-ul requestului, headerele HTTP, checksum-ul payload-ului

HTTP și o cheie privată pentru a crea hash-ul. Pentru ca întreg sistemul să funcționeze, atât consumatorul cât și cel care pune la dispoziție API-ul trebuie să aibă aceeași cheie privată. Dezavantajul acestei modalități este acela că nu poate fi folosită de către browser/ client ci doar între API-uri.

O ultimă metodă de asigurarea a autentificării este prin folosirea unei parole de unică folosință. Aceasta este de obicei utilizată în aplicațiile care pun la dispoziție autentificarea în 2 pași, astfel ca userul după ce își introduce username-ul și parola, atât serverul cât și clientul vor genera o astfel de parolă unică ce va fi folosită o singură dată.

## 2.1. Autentificarea

În cadrul aplicației de strângere de fonduri, autentificarea cuprinde un singur factor de autentificare care se realizează prin simpla introducere a credențialelor (email și parolă). În urma validării credențialelor un token va fi returnat de către API și stocat de către client în Local storage-ul browserului. Pe baza acestui token userul ce îl deține va avea acces la anumite resurse, iar către altele accesul îi va fi restricționat.

La nivel de API sunt expuse 2 endpoint-uri care nu necesită autentificare: /register și /login. Procesul de autentificare începe cu un POST request la endpoint-ul de /login cu credențialele userului în format JSON.

The screenshot displays an API client configuration window. At the top, the 'Description' field is set to 'authenticationUserDto'. Below this, there are two tabs: 'Example Value' (selected) and 'Model'. The 'Example Value' tab shows a JSON object: 

```
{  "email": "test@test.com",  "password": "password"}
```

. At the bottom, there is a 'Parameter content type' dropdown menu with 'application/json' selected. A 'Cancel' button is located above the dropdown.

Fig. 2.1 JSON body-ul și tipul conținutul acceptate de requestul de login

Source: <http://localhost:8080/swagger-ui.html#/authentication-controller/>

În Fig. 2.1 este reprezentată modalitatea de trimitere a credențialelor către API, astfel se poate observa că este acceptat un DTO (Data Transfer Object), construit pe baza entității de la nivelul bazei de date User, care conține e-mailul și parola utilizatorului. De asemenea, este specificat și formatul în care serverul poate procesa requestul și anume application/json.

Odată ce aceste informații au fost trimise de către client, urmează procesarea lor la nivelul serverului. Autentificarea se realizează prin apelarea metodei `authenticate()` din `AuthenticationManager`, pus la dispoziție de către Spring Security, care primește ca parametru un obiect de tipul `UsernamePasswordAuthenticationToken` care conține email-ul și parola introduse de către utilizator. Pentru a verifica autenticitatea userului am utilizat `UserDetailsService`, interfață pentru care am realizat o implementare care să se plieze pe tipul de autentificare bazată pe token, și anume am suprascris metoda `loadUserByUsername()`. În cadrul acestei metode se realizează căutarea în baza de date a userului cu email-ul din body-ul requestului POST. Pe baza utilizatorului returnat se creează o instanță a clasei `JWTUserDetails`, implementare a interfeței `UserDetails` ce stochează informațiile despre utilizator.

Următorul pas în procesul de autentificare este generarea JWT-ului.

```
public String generateJWTToken(JWTUserDetails user)
{
    Claims claims = Jwts.claims().setSubject(user.getUsername());
    claims.put("roles", user.getAuthorities());
    claims.put("username", user.getName());
    return doGenerateToken(claims);
}
```

Tabela 2.1 Adăugarea claim-urilor în JWT

Așa cum se poate observa și în Tabela 2.1, în metoda `generateJWTToken()` pe baza instanței de `JWTUserDetails` returnate de metoda `loadUserByUsername()`, se vor seta claim-urile, afișate în partea de payload a JWT-ului, cu care va fi generat tokenul ca atare.

```
private String doGenerateToken(Map<String, Object> claims) {
    return Jwts.builder().setClaims(claims).setIssuedAt(new
        Date(System.currentTimeMillis()))
        .setExpiration(new Date(System.currentTimeMillis() +
            JWTToken_VALIDITY * 1000))
        .signWith(SignatureAlgorithm.HS256, secret).compact();
}
```

Tabela 2.2 Generarea JWT-ului

Pe baza claim-urilor, în metoda `doGenerateTokens` (surprinsă în Tabela 2.2) se va genera tokenul care va conține: claim-urile, data în milisecunde la care a fost generat și la care va expira, algoritmul cu care a fost semnat și secretul. Token-ul generat este stocat la nivelul unei



instanțe de `AuthenticationResponse` și mai apoi trimis ca și body în răspunsul requestului de `/login`.

Autentificarea la nivel de client este reprezentată de componenta de authentication care cuprinde componentele de register și login. Aceste 2 componente sunt definite în `app-routing.module.ts` ca fiind copii ai rutei de `"/auth"` ce nu necesită trecerea request-urilor prin `Auth Guard Service` (serviciu care filtrează requesturile și decide care sunt cele ce au nevoie de autorizare și care nu). Urmărind procesul de logare la nivelul clientului, după ce userul și-a introdus credențialele, un prim pas al acestuia îl reprezintă preluarea datelor din formular și trimiterea acestora în format JSON prin intermediul metodei de `login` (Tabela 2.3) din serviciul `UserService`, așa cum se poate observa în codul de mai jos.

```
login(user: LoginUser) {  
  return this.http.post<User>(`${environment.apiUrl}/login`,  
    JSON.stringify(user),  
    this.options);  
}
```

Tabela 2.3 Metoda de login de la nivelul serviciului din client

După ce requestul POST a fost finalizat cu succes, acesta va returna un JSON care conține la cheia `"token"` valoarea tokenului ce urmează a fi stocat la nivel de Local storage.

```
login(email, password) {  
  let loginUser = {  
    email: email,  
    password: password  
  }  
  return this.userService.login(loginUser)  
    .pipe(map(user => {  
      localStorage.setItem('token', JSON.stringify(user.token));  
      return user;  
    })));  
}
```

Tabela 2.4 Stocarea tokenului la nivel de Local storage

Așa cum se poate observa în Tabela 2.4 răspunsul requestului POST la endpoint-ul `"/login"` este procesat astfel încât doar valoarea tokenului să fie preluată și asignată cheii `"token"` din Local storage-ul browserului. De acolo valoarea tokenului este preluată și trimisă ca și header de Authorization sub forma `"Bearer {{valoarea tokenului}}"` în requesturile care necesită autorizare.

## 2.2. Autorizarea

Autorizarea în cadrul aplicației de strângere de fonduri este una bazată pe roluri, astfel ca am definit o entitate proprie `Roles` care cuprinde toate rolurile existente în aplicație. Fiind în strânsă legătură cu entitatea `Users`, definind rolul pe care un user îl poate avea în cadrul

aplicației, am definit între acestea o relație de tipul OneToMany, întrucât un rol poate fi asignat mai multor useri, însă un user poate avea un singur rol. În aplicație există 4 roluri: ADMIN, USER rol care este asignat oricărui utilizator atunci când își creează un cont, FUNDRAISER rol pe care îl primește un user care a adăugat un caz caritabil, după ce acesta a fost aprobat de către administrator și COMPANY rol pe care îl obține un user care a adăugat o companie.

La nivelul API-ului apartenența unui user la un rol, alături de acțiunile pe care le poate face un user în funcție de rolul său, sunt manageriate de configurarea făcută la nivel de Spring Security.

```
@Override
protected void configure(HttpSecurity httpSecurity) throws Exception{
    httpSecurity.csrf().disable()
        .authorizeRequests().antMatchers(
            ...
        ).permitAll()

    .antMatchers(HttpMethod.GET, "/categories").hasAnyAuthority("FUNDRAISER", "USER",
    "COMPANY", "ADMIN")

    .antMatchers(HttpMethod.GET, "/fundraisers/**").hasAnyAuthority("FUNDRAISER",
    "USER", "COMPANY", "ADMIN")

    .antMatchers(HttpMethod.POST, "/fundraisers").hasAnyAuthority("FUNDRAISER",
    "USER", "ADMIN")

    .antMatchers(HttpMethod.POST, "/companies").hasAnyAuthority("COMPANY",
    "USER", "ADMIN")

        .anyRequest().authenticated().and()

    .exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint).and()
    ().sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
        httpSecurity.addFilterBefore(jwtRequestFilter,
    UsernamePasswordAuthenticationFilter.class);
    httpSecurity.cors();
}
```

Tabela 2.5 Restricționarea acțiunilor în funcție de rolul utilizatorului

Conform Tabelei 2.5 fiecare tip de request la un anumit endpoint implică existența unui rol specific, verificat de metoda `hasAnyAuthority()`. În cazul în care un utilizator dorește să facă o anumită acțiune și nu are rolul necesar pentru a obține informația dorită va primi răspunsul 401 Unauthorized.

În cadrul aplicației client, la fel ca și pe partea de API, există un bun management al rolurilor pe care le au userii, afișându-se sau nu informațiile la care aceștia au acces. La nivel de `app-routing.module.ts` fiecare rută care necesită autentificarea are specificată cheia “canActivate”

pentru care valoarea este AuthGuardService, serviciu responsabil cu verificarea existenței unui token valid și a rolului necesar pentru a efectua un request.

```
canActivate(route: ActivatedRouteSnapshot) {
    this.currentUser = JSON.parse(localStorage.getItem('token'));
    if (this.currentUser) {
        var decode = jwt_decode(this.currentUser);
        if (route.data.roles &&
route.data.roles.indexOf(decode['roles'][0]['authority']) == -1) {
            this.router.navigate(['/auth']);
            return false;
        }
        return true;
    }
    return this.router.parseUrl('/auth');
}
```

Tabela 2.6 Managementul rolurilor la nivelul clientului

Metoda din Tabela 2.6 este apelată de fiecare dată când o rută cu cheia “canActivate” este apelată. În cadrul acesteia se verifica existența tokenului în Local storage, apoi tokenul este procesat și este verificată existența unui rol specific care permite accesul la ruta cerută. În cazul neîndeplinirii uneia din aceste 2 condiții utilizatorul va fi redirecționat către ruta de autentificare.

### 2.3. Utilizarea JWT în contextul aplicației

În cadrul aplicației de strângere de fonduri JWT-urile joacă un rol important în procesul de autentificare, astfel că în urma unui request de “/login” cu răspuns de succes este generat un astfel de token. În același timp este esențial în realizarea managementului rolurilor la nivel de client prin faptul că informații legate de rolul utilizatorului sunt stocate în acesta.

O privire de ansamblu asupra componenței unui JWT dezvăluie existența a 3 parti esențiale: headerul, payloadul și semnătura. Aceste componente sunt reprezentate sub forma xxxxx.yyyyy.zzzzz.<sup>[2]</sup> Pentru o mai bună înțelegere vom analiza cazul unui JWT generat de către requestul la endpointul “/login”.



### 3. Principalele flow-uri în funcție de rolul utilizatorului

Aplicația de strângere de fonduri se adresează unei game variate de utilizatori, de aceea în cadrul aplicației se regăsesc 4 roluri importante pe care aceștia le pot avea. Așa cum orice aplicație necesită, există un administrator care la nivelul aplicației are rol de ADMIN, pe lângă acesta orice utilizator care deține un cont are rolul de USER. Un utilizator care dorește să adauge un caz caritabil va obține rolul de FUNDRAISER după ce requestul său de a adăuga un caz caritabil a fost acceptat de către un administrator. În aceeași postură se află și utilizatorul care dorește să adauge o companie, având rolul de COMPANY abia după ce requestul a fost acceptat de către un administrator.

ADMIN-ul, așa cum se poate observa în Fig. 3.1, trece ca un user normal prin procesul de autentificare. Odată ce, pe baza credențialelor introduse, i s-a identificat rolul de administrator, acesta va putea realiza anumite acțiuni.

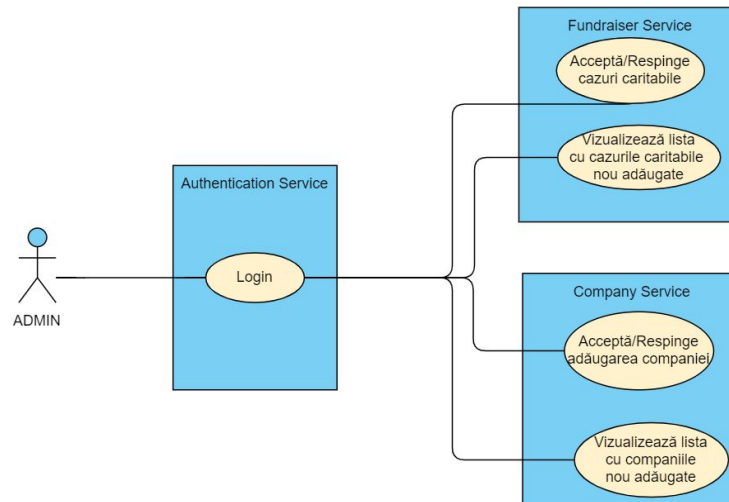


Fig. 3.1 Acțiunile ce pot fi întreprinse de către un utilizator cu rol de ADMIN

Source: <https://online.visual-paradigm.com/app/diagrams/#proj=0&type=UseCaseDiagram>

În cadrul sistemului de cazuri caritabile, va putea vizualiza o listă cu toate cazurile noi adăugate, având posibilitatea de a accepta sau respinge acele cazuri. Odată acceptate sau respinse acestea nu vor mai fi afișate în lista administratorului, ci vor fi afișate celorlalți utilizatori care au posibilitatea să doneze. În sistemul companiilor, administratorul poate realiza aceleași acțiuni, cu mențiunea că o companie poate adăuga vouchere cazurilor caritabile, după ce a fost acceptată.

Un utilizator normal având rolul de USER poate realiza următoarele acțiuni (reprezentate și în Fig 3.2):

- poate adăuga un nou caz caritabil în sistemul de cazuri caritabile, rolul acestuia devenind unul de FUNDRAISER, poate vizualiza lista cu toate cazurile caritabile existente și active, le poate accesa pagina de detalii și poate dona pentru acestea.

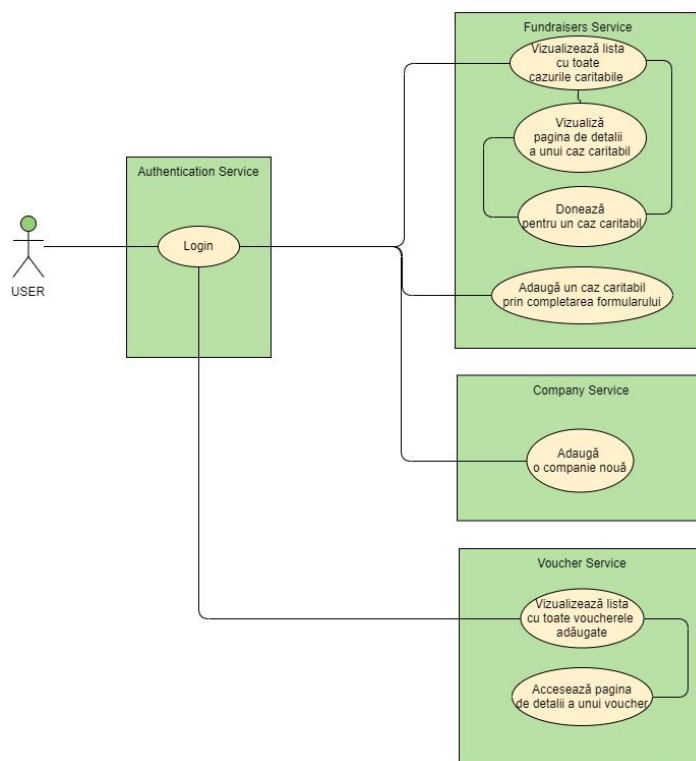


Fig. 3.2 Acțiunile ce pot fi întreprinse de către un utilizator cu rol de USER

Source: <https://online.visual-paradigm.com/app/diagrams/#proj=0&type=UseCaseDiagram>

- poate adăuga o companie în sistemul de companii, rolul acestuia devenind COMPANY
- poate vizualiza o listă cu toate voucherele adăugate de către companii în sistemul de vouchere, are posibilitatea de a accesa pagina de detalii a unui astfel de voucher unde va găsi informații legate de cazul caritabil pentru care au fost adăugate și compania de care au fost create.

Odată ce a adăugat o companie și requestul a fost acceptat de către un administrator, un utilizator cu rolul de COMPANY poate adăuga vouchere la cazurile caritabile pe care dorește să le susțină. După cum se poate observa și în Fig 3.3, acest tip de utilizator poate vizualiza lista cu cauze caritabile, accesa detaliile unui caz caritabil, de unde are opțiunea de a adăuga vouchere. Adăugarea voucherelor se realizează prin completarea unui formular. O companie poate, de asemenea, dona pentru o cauză caritabilă.

În sistemul de vouchere o companie poate întreprinde aceleași acțiuni ca un utilizator normal, și anume, poate vizualiza lista cu toate voucherele și poate accesa detaliile unuia pentru a vedea cazul caritabil asociat și compania ce îl susține.

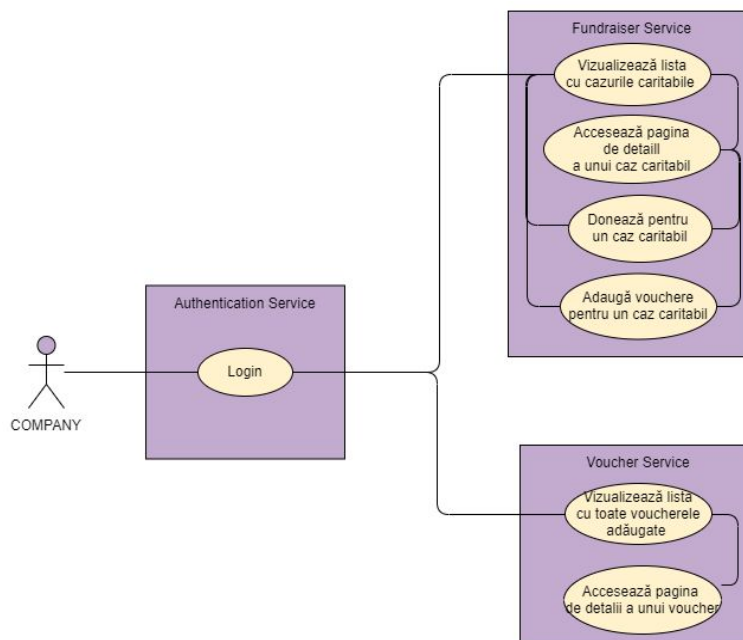


Fig. 3.3 Acțiunile ce pot fi întreprinse de către un utilizator cu rol de COMPANY

Source: <https://online.visual-paradigm.com/app/diagrams/#proj=0&type=UseCaseDiagram>

Cazul utilizatorilor care au rolul de FUNDRAISER este similar cu cel al companiilor când vine vorba de obținerea acestui rol, fiind necesară și în acest caz aprobarea administratorului. O dată rolul de FUNDRAISER obținut, acesta poate vizualiza toate cazurile caritabile, inclusiv cele adăugate de el, poate vizualiza pagina de detalii a fiecărui caz și poate adăuga unele noi. În sistemul de vouchere poate vizualiza toate voucherele și poate accesa detaliile acestora, inclusiv ale celor adăugate pentru cazurile sale caritabile (Fig 3.4).

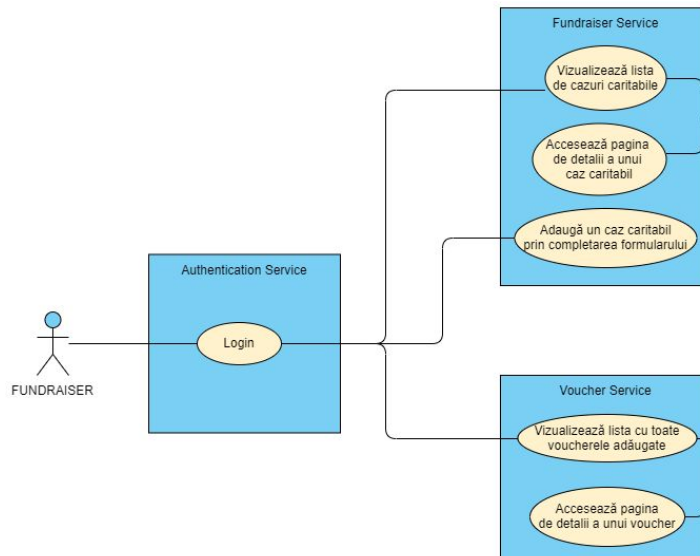


Fig 3.4 Acțiunile ce pot fi întreprinse de către un utilizator cu rol de COMPANY

Source: <https://online.visual-paradigm.com/w/zzunxxsq/diagrams/#proj=0&type=UseCaseDiagram>

Așadar, prin parcurgerea acestor flow-uri, precum și prin ilustrarea lor cu ajutorul diagramelor Use Case în care actorii sunt reprezentați de principalele roluri din cadrul aplicației de strângere de fonduri, business-ul din spatele aplicației devine mult mai ușor de înțeles.



## 4. Procesarea imaginilor în contextul anonimizării persoanelor

Un aspect foarte important al aplicației de strângere de fonduri este acela de a asigura păstrarea anonimă a informațiilor personale, aici fiind vorba de fotografiile pe care un utilizator le poate adăuga atunci când înregistrează un caz caritabil. În contextul alinierii cu ceea ce înseamnă normele GDPR, formularele nu vor avea câmpuri obligatorii atunci când vine vorba de date sensitive, iar prin completarea câmpurilor respective utilizatorul își oferă acordul cu privire la prelucrarea acestora. Cum imaginile încărcate de către utilizator pot conține alte persoane în afară de cel ce completează formularul, este necesară păstrarea secretă a identității persoanelor respective prin aplicarea unui layer deasupra fiecărei fețe identificate, astfel încât trăsăturile să nu poată fi identificate.

În continuare, este evidențiată o analiză a întregului proces de încărcare a imaginilor la nivel de client, procesarea acestora de către API, salvarea în baza de date și procesul invers până la afișarea lor în interfață. Într-un subcapitol separat se va realiza o analiză amănunțită a modului de procesare al imaginii în vederea identificării fețelor și aplicarea unui layer deasupra acestora folosind CascadeClassifier-ul pus la dispoziție de biblioteca OpenCV.

### 4.1. Procesarea imaginii la nivel de client și trimiterea către API

Aplicația pune la dispoziție utilizatorilor posibilitatea de a adăuga imagini atunci când completează formularele de înregistrare a unui caz caritabil, respectiv a unei companii. La nivel de client această funcționalitate este implementată printr-un input care acceptă fișiere multiple în format .JPG. Odată încărcate fișierele, numele acestora alături de un buton care permite ștergerea fiecărui fișier vor fi afișate într-un tabel (vezi Fig. 4.1).

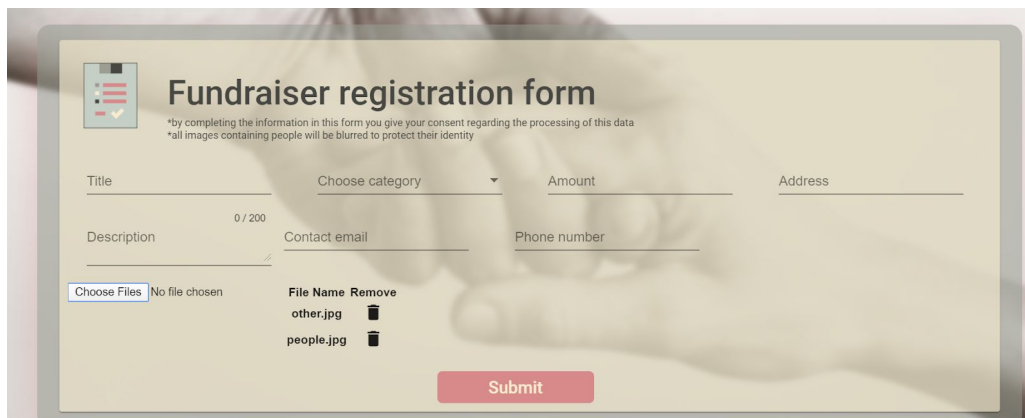


Fig. 4.1 Formularul de adăugare a unui caz caritabil

Datorită procesului asincron de încărcare a fișierelor și a faptului că fișierele trebuie trimise integral către API, a fost necesară utilizarea obiectului Promise.

```
getImage(file: File) {
```

```

return new Promise<string>((resolve) => {
    var myReader: FileReader = new FileReader();
    myReader.onloadend = (e) => {
        resolve(myReader.result as string)
    };
    myReader.readAsDataURL(file);
});
}

```

Tabela 4.1 Transformarea unei imagini în reprezentarea ei Base64 encoded

Deoarece imaginile încărcate au diferite dimensiuni și transformarea acestora în Base64 este un proces ce necesită timp pentru ca întreg conținutul imaginii să fie procesat, am utilizat o instanță a obiectului Promise în care se află stringul obținut în urma transformării. Astfel, după cum se poate observa și în Tabela 4.1, metoda `getImage()` va returna un Promise cu un fișier în format Base64, obținut prin apelarea metodei `readAsDataURL(file)`. La momentul creării instanței de Promise aceasta se afla în starea de pending. În continuare dacă task-ul asincron este completat cu succes este apelată metoda `resolve()` care randează rezultatul taskului asignat, în acest moment este apelată metoda `OnFulfilled()` care generează schimbarea stării instanței de Promise în Fulfilled. Cazul prezentat rezolvă problema existenței unei singure imagini, în continuare se va trata cazul existenței mai multor imagini.

```

getImage() {
    var images: Promise<string>[] = [];
    for (let j = 0; j < this.uploader.queue.length; j++) {
        var file: File = this.uploader.queue[j]._file;
        if(file.size > 10000000){
            alert("Each file should be less than 10 MB of size.");
            return;
        }
        images.push(this.getImage(file))
    }
    return images
}

```

Tabela 4.1 Procesarea listei de imagini

Deoarece metoda care transformă imaginea în reprezentarea ei Base64 returnează o instanță de Promise, atunci când vom procesa lista de imagini, va fi necesar ca fiecare imagine să treacă prin funcția mai sus menționată. De aceea, am creat o listă de obiecte Promise care va conține toate imaginile încărcate în format Base64, după ce acestea au fost preluate din input și trecute prin filtrul de dimensiune.

Următorul pas în procesul de trimitere al imaginilor, în JSON body-ul requestului POST sub forma unei liste de stringuri la cheia “images”, este tot unul asincron întrucât toate imaginile încărcate trebuie să se regăsească în lista de Promise-uri Base64.

```

...
Promise.all(this.getImages()).then((images_b64: string[]) => {
    this.fundraiserForm.value['images'] = images_b64;
    this.fundraiserService.addFundraiser(this.fundraiserForm.value)
        .subscribe(
            (data) => {
                alert('Fundraiser successfully added!');
                this.router.navigateByUrl('/home/fundraisers');
                this.fundraiserForm.reset();
            }
        );
});
...

```

Tabela 4.2 Secțiune din metoda OnFormSubmit() responsabilă cu formarea listei de imagini și trimiterea ei ca parte a formularului

Metoda `getImages()` returnează o listă de instanțe de `Promise`, ceea ce reprezintă de fapt un grup de promisiuni ce trebuie agregate, de aceea în Tabela 4.2 am utilizat metoda `Promise.all()`, care așteaptă terminarea tuturor operațiilor asincrone. Acest `Promise.all()` este la rândul lui un `Promise` care va apela metoda `resolve()` odată ce toate `Promise`-urile primite ca argument au fost finalizate. La finalizarea `Promise.all()` rezultatul este asignat unei variabile a cărei valoare va fi introdusă în formular sub cheia “images”. Formularul fiind acum complet va fi trimis ca body în requestul de adăugare al entității.

```

private String imageProcessing(String image){
    String prefix = image.split(",")[0];
    String extension = prefix.split("/") [1].split(";") [0];
    String processedImage = faceDetection.encodeToString(
        faceDetection.toBufferedImage(faceDetection.detectFaces(faceDetection.decodeImage(image))),
            extension);
    return prefix + "," + processedImage;
}

```

Tabela 4.3 Procesarea imaginii la nivelul serviciului de cazuri caritabile

Deoarece imaginea este primită codificată Base64, iar la nivel de API este necesară convertirea acesteia într-un `BufferedImage`, am implementat la nivelul serviciului de procesare a imaginii o metodă care face această transformare. După procesarea imaginii (subiect discutat pe larg în subcapitolul următor) este returnat un obiect de tipul `Mat` care va fi trecut prin metoda `toBufferedImage(Mat mat)` pentru a reveni la reprezentarea imaginii ca un `BufferedImage`. Din acest punct începe procesul invers în care acel `BufferedImage` este transformat din nou în string pentru a fi stocat la nivelul bazei de date. Intrega înlănțuire de metode alături de preprocesările

stringului din care sunt extrase prefixul și extensia imaginii încărcate, poate fi observată în Tabela 4.3.

Imaginea care ajunge la API sub forma de string are următoarea formă: `"data:image/jpeg;base64,/9j/4AAQSkZJRgABAgAAQABAAD..."`, iar partea care reprezintă imaginea propriu-zisă se regăsește după `"data:image/jpeg;base64,"`, de aceea este necesară extragerea acestui prefix. În același format este returnată și în contextul unui request GET, fiind apoi decodată și afișată în pagina de detalii a unui caz caritabil.

## 4.2. Recunoașterea facială în imagini

Ca parte a procesării imaginii, recunoașterea facială reprezintă un pas important deoarece în această etapă se identifică în cadrul imaginii cele 4 coordonate care determină un dreptunghi care să încadreze fața unei persoane. În implementarea acestei funcționalități am utilizat biblioteca OpenCV, mai exact obiectul de tip `CascadeClassifier()` care are la bază conceptul de clasificare în cascadă.

În primul rand, un clasificator este antrenat cu câteva sute de imagini reprezentând același obiect (în cazul aplicației de strângere de fonduri cu fețele persoanelor), numite exemple pozitive, care sunt scalate la aceeași dimensiune, și cu exemple negative (imagini arbitrat alese de aceeași dimensiune). După ce un astfel de clasificator este antrenat, poate fi aplicat pe regiuni de interes dintr-o imagine dată ca input. Clasificatorul returnează `"1"` dacă regiunea arată obiectul căutat, `"0"` altfel. Pentru a căuta obiectul în întreaga imagine, se poate muta fereastra de căutare peste imagine și verifica fiecare locație folosind clasificatorul. Clasificatorul este proiectat astfel încât să poată fi „redimensionat” cu ușurință pentru a putea găsi obiectele de interes la diferite dimensiuni, ceea ce este mai eficient decât redimensionarea imaginii în sine. Deci, pentru a găsi un obiect de dimensiune necunoscută în imagine, procedura de scanare ar trebui să fie făcută de mai multe ori la diferite dimensiuni. Cuvântul „cascadă” din numele clasificatorului înseamnă că clasificatorul rezultat constă din mai multe clasificatoare (etape) mai simple care sunt aplicate ulterior unei regiuni de interes până când la un moment dat candidatul este respins sau toate etapele sunt trecute. <sup>[3]</sup>

În cadrul aplicației de strângere de fonduri am utilizat un clasificator antrenat să detecteze fețele persoanelor în poziție frontală, cu ajutorul regulilor din fișierul: `haarcascade_frontalface_alt.xml`<sup>[4]</sup>. Așa cum se poate observa și în Tabela 4.4, am creat o instanță de `CascadeClassifier` care cu ajutorul metodei `load()` încarcă clasificatorul ce urmează a fi folosit în identificarea fețelor.

Parametrul acestei metode care procesează imaginea este un `BufferedImage`, însă pentru detectarea fețelor este necesar ca imaginea să fie un obiect de tip `Mat`, care primește ca parametri înălțimea și lungimea în pixeli a bufferului, și tipul elementelor din matrice. Această instanțiere practic doar pregătește dimensiunile pe care trebuie să le aibă matricea, urmând ca conținutul efectiv al imaginii să fie încărcat cu ajutorul metodei `put()`, ca `byte[]`.

```

...
CascadeClassifier faceDetector = new CascadeClassifier();
faceDetector.load("haarcascade_frontalface_alt.xml");

Mat image = new Mat(imageBuffer.getHeight(), imageBuffer.getWidth(),
CvType.CV_8UC3);
byte[] data = ((DataBufferByte)
imageBuffer.getRaster().getDataBuffer()).getData();
image.put(0, 0, data);

MatOfRect faceDetection = new MatOfRect();
faceDetector.detectMultiScale(image, faceDetection);

for(Rect rect : faceDetection.toArray()){
    //adăugarea layerului blurred pe fețele identificate
}
...

```

Tabela 4.4 Identificarea fețelor folosind clasificatorul în cascadă

Pentru stocarea fețelor identificate este necesară crearea unei instanțe de `MatOfRect` care pune la dispoziție metoda `detectMultiScale()` cu parametrii imaginea și clasificatorul, și va returna un array de fețe de diferite dimensiuni sub forma unor dreptunghiuri. Următorul pas în procesarea imaginilor îl reprezintă aplicarea unui layer de blur deasupra fețelor identificate.

### 4.3. Aplicarea GaussianBlur

Pentru procesarea imaginilor în acest sens am folosit un modul al librăriei OpenCV, `ImgProc` care pune la dispoziție o serie de metode menite să modifice imaginea originală în diferite moduri. Una din cele mai folosite metode este metoda `GaussianBlur` utilizată în general pentru a reduce zgomotele dintr-o imagine. Aceasta metoda blurează o imagine folosind un filtru Gaussian și are următoarea semnătură: **public static void GaussianBlur(Mat src, Mat dst, Size ksize, double sigmaX).**

```

for(Rect rect : faceDetection.toArray()){
    Mat rectMat = new Mat(image, rect);
    Imgproc.GaussianBlur(rectMat, rectMat, new Size(45, 45), 100, 100);
}

```

Tabela 4.5 Aplicarea GaussianBlur pe fiecare dreptunghi identificat de `faceDetection()`

După cum se poate observa și în Tabela 4.5 pentru a putea aplica metoda `GaussianBlur` este necesar să oferim o imagine sursă și o imagine destinație, în cazul de față ambii parametri au aceeași valoare deoarece se dorește suprascrierea imaginii cu varianta sa blurată. Pentru aceasta vom crea o nouă instanță de `Mat` care primește ca parametri matricea corespunzătoare imaginii neprocesate, și zonele de interes din aceasta (în acest caz dreptunghiurile unde au fost identificate fețele).

Al doilea parametru al metodei este dimensiunea kernelului Gaussian. Kernelul Gaussian este o matrice de pixeli, unde valorile pixelilor corespund valorilor curbei Gaussiene. Următorii parametri ai metodei reprezintă valorile pentru  $\sigma_X$  și  $\sigma_Y$ , și anume deviația standard a kernelului Gaussian pe direcția X respectiv Y.

Procesul de Gaussian blur constă în convoluția dintre imagine și un kernel Gaussian. Acest proces poate fi realizat bidimensional (imaginea și kernelul Gaussian sunt reprezentate ca matrici) într-un singur pas, dar și unidimensional pe orizontală apoi pe verticală, în doi pași, abordare mult mai eficientă din punct de vedere al complexității. În prima fază un kernel unidimensional este folosit pentru a blura imaginea doar vertical. În al doilea pas, același kernel unidimensional este utilizat pentru a blura direcția rămasă.<sup>[5]</sup>



Înainte



După

Fig. 4.2 Rezultatul procesării imaginii

#### 4.4. Concluzie

În Fig. 4.2 se poate observa rezultatul obținut în urma procesării imaginii și totodată outputul afișat utilizatorului în interfața aplicației. Acest demers este necesar în cadrul aplicațiilor de acest tip, deoarece identitatea persoanelor din fotografii trebuie să fie păstrată, în conformitate cu noțiunile impuse de GDPR.

## 5. Concluzii și direcții ulterioare

În concluzie, consider că ideea aplicației de strângere de fonduri, cu scopul de a reuni, prin donații, oamenii proveniți din medii diferite, este una inovativă prin funcționalitățile pe care le pune la dispoziție. Printre acestea se numără posibilitatea de a adăuga un caz caritabil, alegând informația ce va fi oferită, posibilitatea reprezentanților companiilor de a înregistra compania în cadrul aplicației și adăugarea de vouchere cauzelor caritabile pentru a motiva utilizatorii să doneze. Mai mult de atât această platformă se află în conformitate cu regulile privind datele personale ale utilizatorilor impuse de GDPR, și protejează identitatea persoanelor prin aplicarea unei tehnici de anonimizare asupra feței acestora.

Dincolo de funcționalitățile pe care le pune la dispoziție, aplicația susține ideea unui bun management al resurselor între cele 2 părți principale ale proiectului, API-ul și clientul. Astfel pornind de la cel mai jos nivel, cel al datelor, mergând mai departe către logica aplicației și expunerea principalelor endpoint-uri prin controlere, până la aplicația client care împachetează toată informația într-o interfață ușor accesibilă, se poate afirma că platforma este bine structurată și oferă o comunicare naturală între aceste layere.

Ce urmează? Fiind vorba de o aplicație de donații, o următoare funcționalitate este integrarea unui sistem real de plată în cadrul acesteia, capabil să managerieze conturile bancare ale utilizatorilor și să se ocupe de tranzacțiile între acestea. În continuare va fi implementat și un sistem de validare a tichetelor pe care utilizatorii le obțin în urma donațiilor, astfel încât să poată fi verificată autenticitatea lor de către companiile ce le-au pus la dispoziție. O modalitate eficientă o reprezintă generarea de coduri QR la fiecare tichet asociat unui utilizator, pentru companii fiind necesar un sistem de citire a acestora. Un alt plan de viitor este deploy-ul aplicației pe un server public, pentru a putea fi folosită de utilizatori reali, servind astfel nevoilor comunității.

## 6. Bibliografie

1. <https://blog.risingstack.com/web-authentication-methods-explained/>
2. <https://jwt.io/introduction/>
3. [https://docs.opencv.org/2.4/modules/objdetect/doc/cascade\\_classification.html#haar-feature-based-cascade-classifier-for-object-detection](https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html#haar-feature-based-cascade-classifier-for-object-detection)
4. [https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade\\_frontalface\\_alt.xml](https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_alt.xml)
5. [https://en.wikipedia.org/wiki/Gaussian\\_blur](https://en.wikipedia.org/wiki/Gaussian_blur)
6. [https://en.wikipedia.org/wiki/Java\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))
7. [https://www.tutorialspoint.com/spring\\_boot/spring\\_boot\\_introduction.htm](https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm)
8. <https://maven.apache.org/>
9. <https://angular.io/guide/architecture>
10. <https://docs.opencv.org/3.4.9/d1/dfb/intro.html>
11. <https://en.wikipedia.org/wiki/Convolution>
12. [https://docs.opencv.org/3.4/d3/d63/classcv\\_1\\_1Mat.html](https://docs.opencv.org/3.4/d3/d63/classcv_1_1Mat.html)
13. <https://c4model.com/>
14. <https://structurizr.com/>
15. <https://app.quickdatabasediagrams.com/#/d/fxszn>
16. <https://online.visual-paradigm.com/app/diagrams/#proj=0&type=UseCaseDiagram>
17. <https://codecraft.tv/courses/angular/es6-typescript/promises/>
18. <https://docs.spring.io/spring-security/site/docs/3.0.x/apidocs/org.springframework.security.core.userdetails.UserDetails.html>



## **7. Anexe**

### **7.1 Tehnologiile utilizate**

#### **7.1.1 Java**

Java este un limbaj de programare cu scop general, bazat pe clase, orientat pe obiecte și proiectat să aibă cât mai puține dependențe de implementare. Este destinat să permită dezvoltatorilor de aplicații să scrie o singură dată, să ruleze oriunde (WORA), ceea ce înseamnă că codul Java compilat poate rula pe toate platformele care acceptă Java fără a fi nevoie de recompilare. Aplicațiile Java sunt de obicei compilate ca bytecode care pot rula pe orice mașină virtuală Java (JVM), indiferent de arhitectura computerului de bază.<sup>[6]</sup>

#### **7.1.2 Spring Boot**

Spring Boot oferă o bună platformă pentru dezvoltatorii Java pentru a dezvolta o aplicație Spring independentă și de calitate, la care e necesară doar rularea. Proiectul se poate începe cu configurații minime fără a fi necesară o configurație completă a Spring. Spring Boot este proiectat cu următoarele obiective: pentru a evita configurația XML, pentru a dezvolta aplicații Spring pentru producție într-un mod mai ușor, pentru a reduce timpul de dezvoltare și pentru a rula aplicația în mod independent.<sup>[7]</sup>

#### **7.1.3 Maven**

Apache Maven este un instrument de gestionare și înțelegere a proiectelor software. Pe baza conceptului de proiect object model (POM), Maven poate gestiona construirea, generarea rapoartelor și documentarea unui proiect preluând informația de la o autoritate centrală.<sup>[8]</sup>

#### **7.1.4 Angular**

Angular este o platformă și framework pentru construirea aplicațiilor client în HTML și TypeScript. Angular este scris cu TypeScript. Implementează funcționalități de bază și opționale ca un set de biblioteci TypeScript care pot fi importate în aplicații.

Blocurile de bază ale unei aplicații Angular sunt NgModules, care oferă un context de compilare pentru componente. NgModules colectează codul aferent în seturi funcționale; o aplicație Angular este definită de un set de NgModule. O aplicație are întotdeauna cel puțin un modul rădăcină care permite bootstrapping-ul și are de obicei multe alte funcții.<sup>[9]</sup>

#### **7.1.5 OpenCV**

OpenCV este o librărie open source, licențiată BSD care include câteva sute de algoritmi de computer vision. Are o structură modulară, ceea ce înseamnă că pachetul include câteva librării shared sau statice. Conține modulele: Core functionality (core), Image Processing (imgproc), Video Analysis (video), Camera Calibration and 3D Reconstruction (calib3d), 2D

Features Framework (features2d), Object Detection (objdetect), High-level GUI (highgui), Video I/O (videoio) și alte module.<sup>[10]</sup>

#### **7.1.6 JWT (JSON Web Token)**

JSON Web Token (JWT) este un standard (RFC 7519) care definește o modalitate compactă și de sine stătătoare pentru a transmite în siguranță informații între părți ca obiect JSON. Aceste informații sunt de încredere și pot fi verificate, deoarece sunt semnate digital. JWT-urile pot fi semnate folosind un secret (cu algoritmul HMAC) sau o pereche de chei publică/ privată folosind RSA sau ECDSA.<sup>[2]</sup>

#### **7.1.7 Convoluția**

În analiza funcțională, convoluția este o operație matematică între două funcții ( $f$  și  $g$ ) care produc o a treia funcție care exprimă modificarea formei uneia dintre ele în funcție de cealaltă. Termenul de convoluție se referă atât la funcția rezultat cât și la procesul de calculare a acesteia. Este definită ca integrala produsului a două funcții după ce una a fost inversată și shiftată.<sup>[11]</sup>