

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

**FACULTATEA DE INFORMATICĂ**



LUCRARE DE LICENȚĂ

**Implementarea unei platforme de gestionare și  
editare a imaginilor folosind tehnologii web**

propusă de

**Cristian-Ștefan Rusu**

**Sesiunea: iulie, 2019**

Coordonator științific

**Conf. Dr. Vitcu Anca**

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

**FACULTATEA DE INFORMATICĂ**

**Implementarea unei platforme de  
gestionare și editare a imaginilor  
folosind tehnologii web**

**Cristian-Ștefan Rusu**

**Sesiunea: iulie, 2019**

Coordonator științific

**Conf. Dr. Vitcu Anca**

Avizat,  
Îndrumător lucrare de licență,  
Conf. Dr. Vitcu Anca.

Data: ..... Semnătura: .....

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

Subsemnatul **Rusu Cristian-Stefan** domiciliat în **România, jud. Bacau, mun. Bacau, strada Theodor Neculuță, nr. 13**, născut la data de **01 August 1996**, identificat prin CNP **1960801046211**, absolvent al Facultății de informatică, specializarea **informatică**, promoția 2018, 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 **Implementarea unei platforme de gestionare și editare a imaginilor folosind tehnologii web** elaborată sub îndrumarea doamnei **Conf. Dr. Vitcu Anca**, pe care urmează să o susțin 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 ei 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: .....

Semnătura: .....

### **Declarație de consimțământ**

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Implementarea unei platforme de gestionare și editare a imaginilor folosind tehnologii web**, 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.

Absolvent **Cristian-Ștefan Rusu**

Data: .....

Semnătura: .....

# Cuprins

<b>Introducere</b>	<b>2</b>
<b>Contribuții</b>	<b>3</b>
<b>1 Arhitectura</b>	<b>4</b>
1.1 Scenarii și cazuri de utilizare . . . . .	5
1.2 Detalii adiționale relevante arhitecturii . . . . .	6
<b>2 Implementare</b>	<b>8</b>
2.1 Detalii de implementare ale modulul de Front-end . . . . .	8
2.1.1 Tehnologii utilizate . . . . .	8
2.1.2 Prezentarea interfeței disponibile utilizatorului . . . . .	10
2.1.3 Editor-ul . . . . .	11
2.1.4 Prezentarea uneltelor disponibile în editor . . . . .	14
2.1.5 Galeria . . . . .	19
2.1.6 Secțiunea Explore . . . . .	19
2.2 Detalii de implementare ale modulului de Back-end . . . . .	19
2.2.1 Tehnologii utilizate . . . . .	19
2.2.2 Structura . . . . .	21
2.3 Detalii de implementare ale server-ului de fișiere . . . . .	22
2.3.1 Tehnologii utilizate . . . . .	22
2.3.2 Rolul . . . . .	22
<b>3 Rularea aplicației</b>	<b>26</b>
3.1 Server-ul Back-end . . . . .	26
3.2 Sever-ul de fișiere . . . . .	27
3.3 Server-ul Front-end . . . . .	28
<b>Concluzii</b>	<b>30</b>
<b>Bibliografie</b>	<b>31</b>

# Introducere

Încă de la inventarea aparatului foto, fiecare individ a avut nevoia de a-și stoca într-un loc sigur și ușor accesibil cele mai frumoase amintiri obținute dealungul anilor. Acest lucru a fost făcut posibil, până în trecutul apropiat, de albumele foto. Acum în era digitală, standardul este de a avea în orice moment, în orice loc, și prin orice mediu acces la aceste lucruri. Deasemenea, datorită adoptării mediului de stocare digital în defavoarea celui fizic, este îmbunătățită integritatea imaginilor în decursul timpului și confidențialitatea este sporită.

Mai mult decât atât, trecerea la mediul de stocare digital permite și modificarea cu ușurință a acestor imagini, lucru ce poate fi făcut cu scopul de a transmite noi idei și emoții.

# Contribuții

Această lucrare are ca scop realizarea unei platforme ușor de utilizat ce permite utilizatorilor să își găsească toate fotografiile într-un singur loc, dar și modificarea acestora după bunul plac.

Platforma pune la dispoziție posibilitatea de a încărca fotografii prin intermediul unei interfețe web. Orice fotografie încărcată pe platformă are posibilitatea de a avea o colecție de informații atașate acesteia, cum ar fi: nume, descriere, persoane.

Aceste fotografii sunt organizate într-o galerie, și pot fi găsite ușor cu ajutorul unei funcții de căutare.

O altă funcționalitate ce stă la baza acestei platforme este editor-ul de fotografii. Odată încărcată o fotografie în sistem, aceasta poate fi deschisă în editor. Acesta lucrează cu conceptul de "nivel" (engl. layer). Orice fotografie ne-editată poate fi considerată un layer. Sunt puse la dispoziție operații de baza asupra unui layer, precum: translația (repoziționarea), scalarea, rotația, distorsionarea. Asupra unui layer, de asemenea, se pot aplica diverse filtre de culoare. O altă opțiune de a obține un layer, este prin manipularea unor obiecte 3D încărcate de către utilizator.

Toate fotografiile încărcate sau realizate în editor, pot fi făcute publice, unde toți utilizatorii platformei le pot vizualiza, și pot oferi comentarii și evaluări.

# Capitolul 1

## Arhitectura

În următoarele rânduri voi descrie cerințele pe care aplicația aceasta trebuie să le îndeplinească pentru a furniza către utilizator funcționalitățile necesare pentru o platformă de gestionare și editare a imaginilor.

În primul rând, voi enumera criteriile de bază ce trebuie îndeplinite de orice platformă web, indiferent de conținut, ce își propune să ofere utilizatorilor săi securitate și control deplin asupra datelor partajate către platforma respectivă:

- Prevenirea persoanelor neautorizate de a accesa datele utilizatorului
- Stocarea persistentă a datelor
- Operații fundamentale asupra datelor (adăugare, ștergere, modificare)

Criterii adiționale pe care trebuie să le îndeplinească o platformă web, pentru a oferi utilizatorilor o bună experiență în utilizarea acesteia:

- Interfață grafică ușor de navigat
- Timp de răspuns rapid la comenzile utilizatorului

Pentru a îndeplini toate punctele expuse mai sus, am decis ca arhitectura sistemului să fie realizată cu sprijinul a trei servere: front-end, back-end și server-ul de fișiere.

Server-ul de front-end are responsabilitatea de a prezenta către utilizator, cu ajutorul unui client web, o interfață grafică prin care sunt afișate toate datele de interes, preluate de la back-end și file-server, și să permită navigarea prin diferite secțiuni ale platformei.

Server-ul back-end, asigură accesul asupra datelor doar persoanelor autorizate, și permite manipularea lor de către aceștia.



Iar în final, server-ul de fișiere, preia responsabilitatea de a gestiona imaginile încărcate de către utilizatori, lucru ce permite un timp de încărcare mai mic a interfeței web.

În figura 1.1 este ilustrat fluxul de acțiuni între fiecare dintre cele trei servere și utilizator.

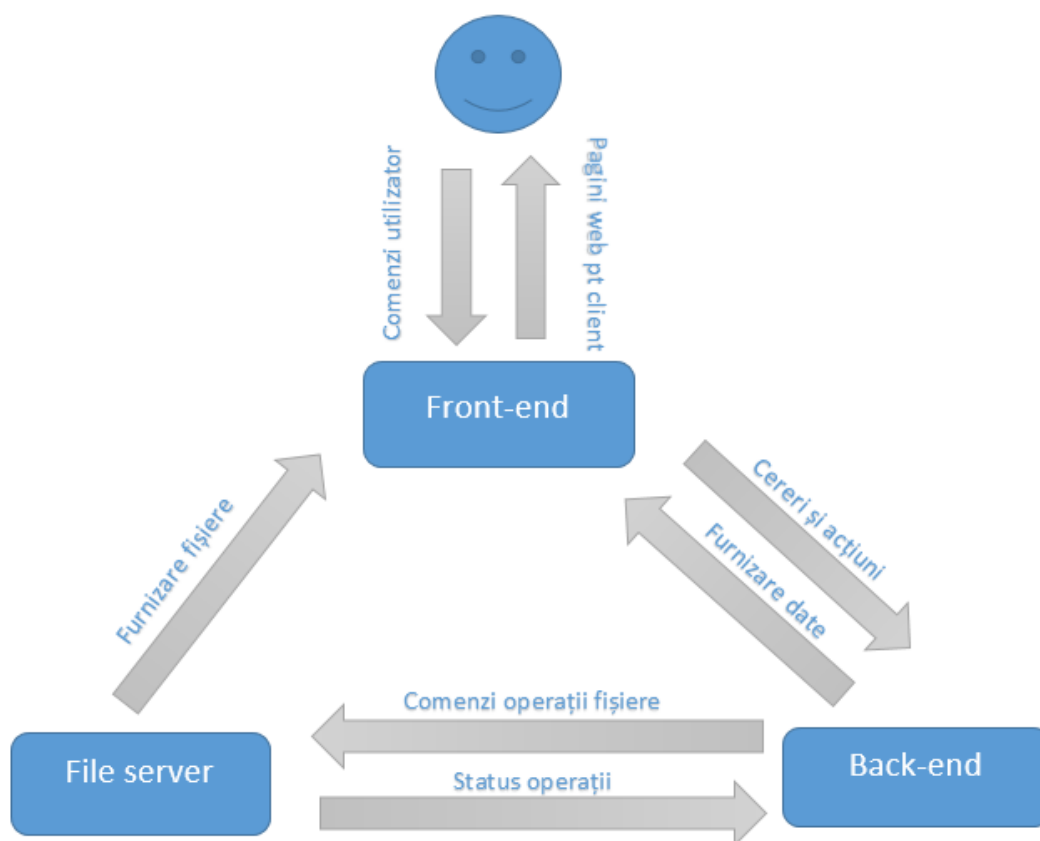


Figura 1.1: Comunicare între server

## 1.1 Scenarii și cazuri de utilizare

Un alt lucru important ce are loc în procesul de realizare al arhitecturii oricărei aplicații este stabilirea modului în care un utilizator va interacționa cu sistemul. În cazul platformei noastre, vom exemplifica aceste cazuri cu ajutorul diagramei din figura 1.2 expusă mai jos.

Primele două acțiuni au utilizatorul sub rolul de vizitator. În acest stadiu, el are posibilitatea de înregistrare (register) și autentificare (login). El nu o să poată accesa alte funcționalități sub acest rol. Odată autentificat, utilizatorul obține acces la acțiunile de bază ale platformei. Acesta are acum posibilitatea de a încărca imagini, de a utiliza

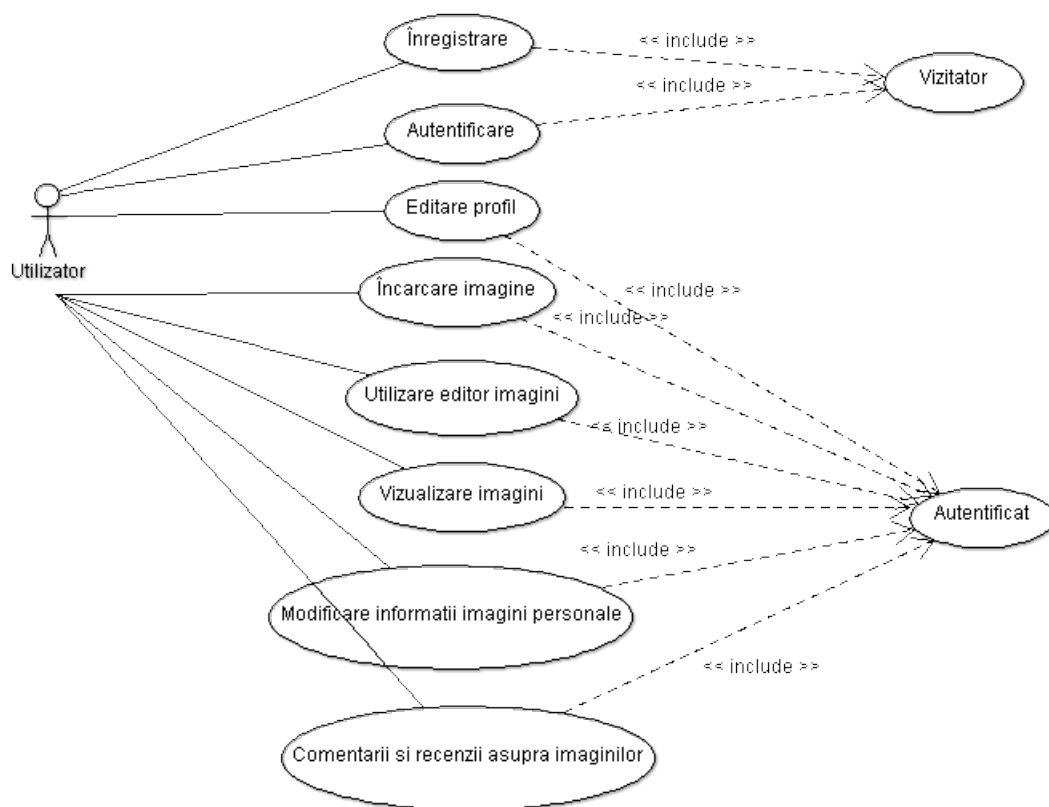


Figura 1.2: Diagrama use-case

editor-ul de imagini, de a-și vizualiza fotografiile sale prin intermediul galeriei, sau ale altor utilizatori navigând spre pagina 'Explore'. Pe lângă aceste lucruri, există posibilitatea de a posta comentarii și recenzii asupra imaginilor încărcate în platformă și de a modifica informațiile atașate profilului sau imaginilor utilizatorului.

## 1.2 Detalii adiționale relevante arhitecturii

Din ce se poate observa mai sus, am stabilit că platforma noastră va fi realizată din trei module principale. Rămâne totuși de stabilit cu ce sisteme este necesar ca aceasta să fie compatibilă. Prin natura ei, fiind bazată pe tehnologii web, din perspectiva utilizatorului, aceasta ar trebui să fie compatibilă cu orice dispozitiv ce are la dispoziție un browser web ce suportă ultimele versiuni de HTML, CSS și JavaScript. În acest sens, rămâne la preferința noastră ce tehnologie folosim pentru a furniza către browser-ul utilizatorului paginile web. Din perspectiva administratorului ce va găzdui această aplicație, ar fi necesar ca platforma să fie cât mai modulară, iar execuția acesteia să se

realizeze fără restricții cu privire la sistemul de operare. În cazul ideal din punct de vedere al performanței, platforma are posibilitatea de a rula pe trei sisteme de calcul individuale. Spre exemplu, activitatea server-ului de fișiere s-ar putea desfășura pe un sistem de calcul ce are la dispoziție o capacitate mare de stocare, cea a back-end-ului poate avea loc pe unul ce are la dispoziție un sistem de stocare de tip SSD (Solid State Drive) ce ar permite accesul mai rapid la informațiile stocate în baza de date, iar server-ul de front-end și-ar putea găsi locul pe un al treilea sistem de calcul, ce prezintă orice tip de specificații, pentru a evita consumul resurselor celorlalte două servere cu cerințe mai mari. Însă, dacă acest lucru nu este posibil, cele trei servere pot rula în orice configurație, chiar și toate trei pe o singură mașină.

În următorul capitol, vom discuta despre ce tehnologii au fost alese pentru a permite implementarea platformei cu privire la detaliile ce tocmai le-am elaborat mai sus.

# Capitolul 2

## Implementare

În acest capitol vom intra în detalii în legătură cu implementarea fiecăruia dintre cele trei module ale proiectului. Fiecare secțiune va începe cu tehnologiile utilizate pentru realizarea modului, iar apoi se va intra în profunzimea implementării funcționalităților.

### 2.1 Detalii de implementare ale modulul de Front-end

#### 2.1.1 Tehnologii utilizate

Așa cum s-a specificat și în capitolul anterior, pentru implementarea modului de front-end, rămâne la latitudinea noastră ce tehnologie alegem pentru a ne ajuta cu transmiterea paginilor web către browser-ul utilizatorului, atât timp cât aceasta este compatibilă cu mai multe sisteme de operare.

Pentru această sarcină, am ales Angular 5. Alegerea a fost bazată pe câteva lucruri puse la dispoziție de către Angular ce permit un timp de dezvoltare mai rapid, și un mediu de dezvoltare mult mai bine structurat.

Primul avantaj în utilizarea framework-ului Angular este faptul că acesta ne pune la dispoziție design-pattern-ul MVC (Model-View-Controller). Acest 'pattern' oferă posibilitatea de a crea interfețe grafice într-un mod elegant, cele trei componente; Model, View și Controller-ul fiind decuplate, lucru ce permite un grad mai ridicat de reutilizare a codului. Componenta Model a MVC-ului se ocupă cu încapsularea datelor unei entități într-o structură de date. Pentru exemplificare, vom folosi drept model - utilizatorul platformei noastre. Componenta model în acest caz conține: nume, prenume, adresa de email, vârsta, țara, orașul și alte detalii legate de acesta, relevante aplicației

noastre. Urmatoarea componentă, View-ul, se ocupă cu reprezentarea informației expusă de model. În cazul nostru, un cod HTML, CSS și JavaScript ce urmează să fie interpretat de către browser într-o interfață grafică. Unul dintre View-urile disponibile în proiectul nostru ce lucrează cu Model-ul utilizator este pagina ce permite editarea profilului. Iar în cele din urmă, componenta Controller se ocupă cu comenzile ce au ca scop manipularea Model-ului. Spre exemplu, funcția de schimbare a adresei de locuință disponibilă pe pagina de editare a profilului, este o parte a Controller-ului ce se ocupă cu preluarea noii adrese din View, și trimiterea unei cereri către back-end pentru ca aceasta să fie modificată în baza de date.

Un alt avantaj este arhitectura bazată pe componente a framework-ului. Sistemul de structurare folosit de Angular permite construirea unei aplicații ce are un grad mic de dependență între componente (cuplaj redus). Acest lucru permite și ierarhizarea proiectului, ceea ce s-ar traduce în ușurința dezvoltatorului de a naviga prin codul sursă.

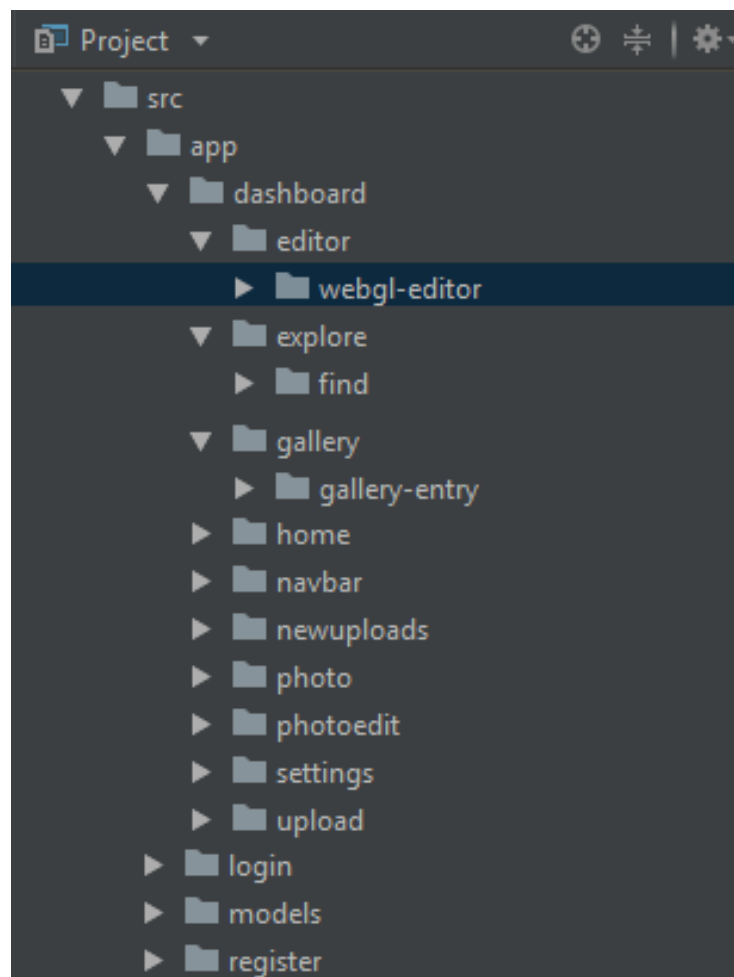


Figura 2.1: Ierarhia modulului front-end excluzând fișierele

Ultimul lucru ce m-a îndreptat spre folosirea Angular în acest proiect a fost TypeScript. Acesta este versiunea mai strictă a lui JavaScript ce suportă tipuri de date, lucru ce oferă posibilitatea detectării unor posibile greșeli încă din faza de scriere a codului.

Ca extensie pentru tot ce oferă Angular, partea de front-end folosește și Bootstrap: un set de template-uri CSS și JavaScript ce ușurează realizarea design-ului paginilor web.

Despre alte tehnologii și librării utilizate în modulul de front-end vom discuta mai în detaliu în prezentarea secțiunilor interfeței cu utilizatorul, ce urmează mai jos.

### 2.1.2 Prezentarea interfeței disponibile utilizatorului

Odată autentificat, utilizatorul este plasat pe pagina principală. Această pagină oferă un flux de notificări cu ultimele activități asociate fotografiilor deja încărcate, câteva link-uri ce oferă acțiuni rapide și butonul de 'Upload' ce trimite utilizatorul pe o pagină unde acesta poate încărca imagini în platformă prin 'Drag and Drop'.

Deasemenea, în partea de sus a paginii se regăsește bara de navigare, element ce își păstrează locul dealungul tuturor paginilor.

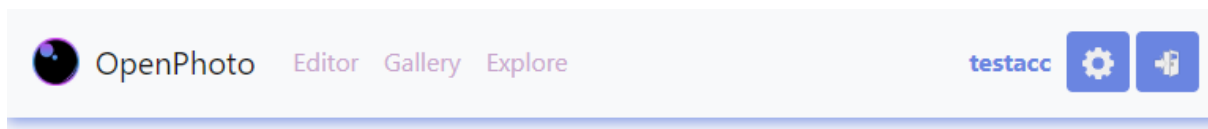


Figura 2.2: Bara de navigare

Rute disponibile de pe bara de navigare (de la stânga spre dreapta):

- Pagina de pornire, disponibilă prin apăsarea logo-ului, OpenPhoto
- Editor, permite navigarea către editor-ul de imagini
- Gallery, permite navigarea către pagina ce afișează toate imaginile încărcate de utilizator
- Explore, pagina ce afișează cele mai noi imagini de pe platformă și cele mai bine evaluate. Deasemenea expune către utilizator posibilitatea de a căuta imagini după anumite filtre, cum ar fi: nume, perioadă de timp și rating.
- Settings, trimite utilizatorul către pagina de editare a profilului.
- Logout, deautentifică utilizatorul.

### 2.1.3 Editor-ul

O funcționalitate importantă ce este pusă la dispoziție către utilizator este editor-ul de imagini, ilustrat în figura 2.3.



Figura 2.3: Exemplu imagine realizată în editor folosind 4 layere

Implementarea acestuia a fost realizată cu ajutorul API-ului JavaScript, WebGL, ce facilitează randarea graficiilor 2D și 3D în interiorul browser-ului web.

În proiectul nostru, toată acțiunea ce se petrece în interiorul editor-ului are loc într-o scenă. În interiorul acestei scene se află imaginile noastre plasate în spațiul tridimensional al acesteia. Pentru o mai bună înțelegere a modului prin care WebGL afișează scena noastră, vom descrie secvența de pași pe care WebGL o urmează pentru a desena un obiect 3D.

Primul pas constă în definirea vertecșilor ce vor alcătui imaginile noastre. Un vertex este reprezentarea unui punct în spațiul 3D. Este nevoie de 4 vertecși pentru a forma un dreptunghi pe care se poate plasa imaginea noastră. În WebGL însă, există doar trei tipuri de primitive: puncte, linii și triunghiuri. Deci, pentru a forma un dreptunghi este nevoie de două triunghiuri ce vor avea două puncte comune. Pentru a instruiți WebGL în ce ordine se vor desena cei 4 vertecși este nevoie de o indexare a vertecșilor. În cazul nostru, sunt necesari 6 indecși ce vor referenția către fiecare dintre cei 4 vertecși necesari formării celor două triunghiuri. Tot în acest pas se vor defini atribute adiționale pentru fiecare vertex, cum ar fi coordonatele UV pentru texturi. Aceste coordonate sunt folosite pentru proiectarea imaginilor 2D pe suprafața unui model 3D.

Următorul pas constă în procesarea vertecșilor definiți în pasul anterior. Acest lucru se întâmplă în interior unui program desemnat să ruleze pe procesorul grafic,

numit Vertex Shader. Acest mic program este invocat de fiecare dată când un vertex este consumat. În vertex shader-ul construit pentru acest proiect, are loc transformarea fiecărui vertex, folosind trei matrici (engl. matrix): World Matrix, View Matrix și Projection Matrix ce au rolul de a proiecta spațiul tridimensional al scenei într-unul bidimensional ce poate fi prezentat pe ecranul utilizatorului.

- Până să ajungă în vertex shader, fiecare vertex definit de noi este considerat a fi în spațiul model. În acest spațiu, fiecare punct este poziționat relativ față de origine. În editorul nostru, ne dorim ca fiecare model să își poată modifica poziția, orientarea și dimensiunea relativ la centrul scenei. Acest lucru se poate înfăptui prin intermediul World Matrix-ului. Cu ajutorul acestei matrici de dimensiune  $4 \times 4$ , putem stoca cele trei transformări esențiale: translația, rotirea și scalarea.
- Următoarea matrice, View Matrix-ul, se ocupă cu poziționarea camerei și orientarea ei către un anumit punct. În realitate, această matrice nu este esențială în procesul nostru de desenare a unei imagini. De exemplu, ne putem imagina două modele aflate în fața noastră, noi aflându-ne în centrul scenei. Pentru a da impresia că noi ne mișcăm spre dreapta cu o unitate avem două opțiuni: fie ne mutăm noi spre dreapta cu o unitate, fie mutăm toată scena spre stânga cu o unitate. Deci, avem următoarele opțiuni în implementare pentru translația, rotația și scalarea (zoom-ul) camerei: aplicăm fiecare transformare asupra fiecărui World Matrix al unui model sau aplicăm transformările unei noi matrici (View Matrix) ce urmează a fi combinată cu world matrix-ul în vertex shader. Din motive de eleganță, s-a ales a doua variantă.
- Ultima matrice prin care vertexii noștri vor avea de călătorit înainte de a fi aplatizați este matricea de proiecție. Projection Matrix-ul are scopul de a aduce vertexii din spațiul camerei în spațiul de proiecție. Acest spațiu este definit ca fiind un cub, ale cărui dimensiuni sunt cuprinse între -1 și 1 pe fiecare dintre cele trei axe. Dacă un vertex se află în interiorul cubului în urma proiecției, acesta se află în raza de acțiune a camerei, iar dacă acesta este înafara cubului, înseamnă că acesta nu poate fi văzut de camera, deci vertex-ul nu va fi regăsit în imaginea bidimensională. Există două tipuri de proiecții, ambele fiind utilizate în proiectul nostru: proiecția ortografică și proiecția de perspectivă.



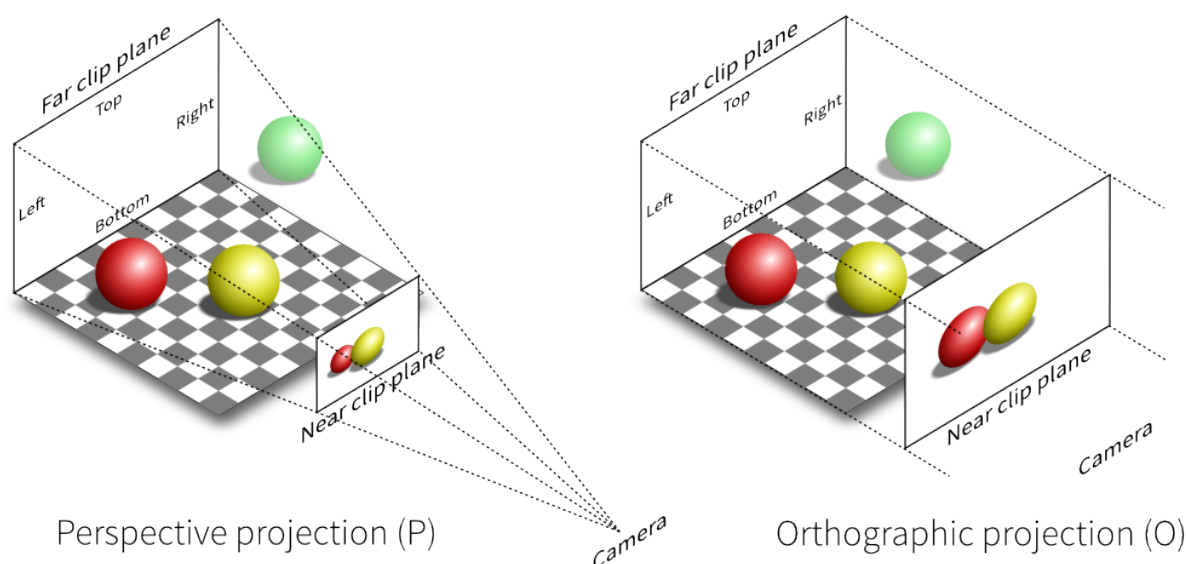


Figura 2.4: Comparație între proiecția ortografică și cea de perspectivă

După cum se poate observa în figura 2.4, principala diferență între cele două tipuri de proiecție când ne referim la imaginea rezultată este senzația de profunzime a imaginii oferită de proiecția de perspectivă și lipsa acesteia în proiecția ortografică. Ambele sunt folosite în avantajul nostru: proiecția ortografică este folosită în ordonarea layerelor pe axa de profunzime fără ca mărimea modelului să fie modificată, iar proiecția de perspectivă este folosită pentru a desena modelele 3D menținând raportul dimensiunilor acestora.

Următorul pas din secvența de desinare WebGL este rasterizarea. Acest lucru este efectuat automat de către procesorul grafic. Din pasul anterior știm că acestui stadiu îi este furnizată o imagine 2D (axa de profunzime a spațiului de proiecție este eliminată), însă această imagine se află într-un spațiu vectorial cuprins între coordonatele -1 și 1 pe cele două axe. Procesul de rasterizare are ca scop transformarea imaginii descrise prin vectori într-o imagine formată dintr-o serie de pixeli.

Ultimul pas constă în procesarea pixelilor, realizată printr-un alt program ce rulează pe procesorul grafic, numit Fragment Shader. Acest program este invocat pentru fiecare pixel consumat. În fragment shader-ul acestui proiect se efectuează diverse operații per pixel ce au rol în implementarea uneltelor de modificare a imaginii, dar și în implementarea luminării obiectelor 3D.

### 2.1.4 Prezentarea uneltelor disponibile în editor



Figura 2.5: Uneltele disponibile editor-ului

Utilizatorului îi sunt puse la dispoziție o serie de unelte ce îl pot ajuta la modificarea imaginilor încărcate de acesta pe platformă. Urmează o scurtă descriere a fiecăreia dintre unelte:

1. Unealta de poziționare - permite utilizatorului repositionarea layer-ului selectat. Acest layer poate fi translatat sus-jos și stanga-dreapta cu ajutorul mouse-ului.
2. Unealta de rotire - facilitează rotirea layer-ului. Layer-ul poate fi rotit cu ajutorul mouse-ului, când acesta este acționat stanga-dreapta.
3. Unealta de redimensionare - permite redimensionarea unui layer. Un layer poate fi redimensionat atât pe lățime, acționând mouse-ul stanga-dreapta, cât și pe lungime, acționându-l pe acesta de sus în jos. Dacă se dorește redimensionarea layer-ului, acesta reținându-și proporțiile inițiale, mouse-ul se va mișca pe diagonala principală.

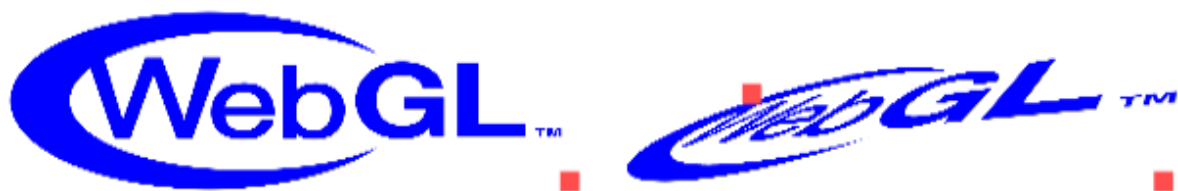


Figura 2.6: Unealtă distorsionare: stânga - imagine originală, dreapta - modificată

4. Unealta de distorsionare - odată selectată această opțiune, în fiecare colț al layer-ului vor apărea niște pătrățele roșii. Fiecare dintre aceste pătrate pot fi mutate cu ajutorul mouse-ului astfel încât să se realizeze efectul de distorsiune asupra layer-ului. O exemplificare a acestei unelte poate fi regăsită în figura 2.6. Pentru implementarea acestei unelte, pe lângă ideea evidentă de re poziționare a vertecșilor în spațiul modelului, este necesară și recalcularea coordonatelor UV ale texturii. Așa cum se poate

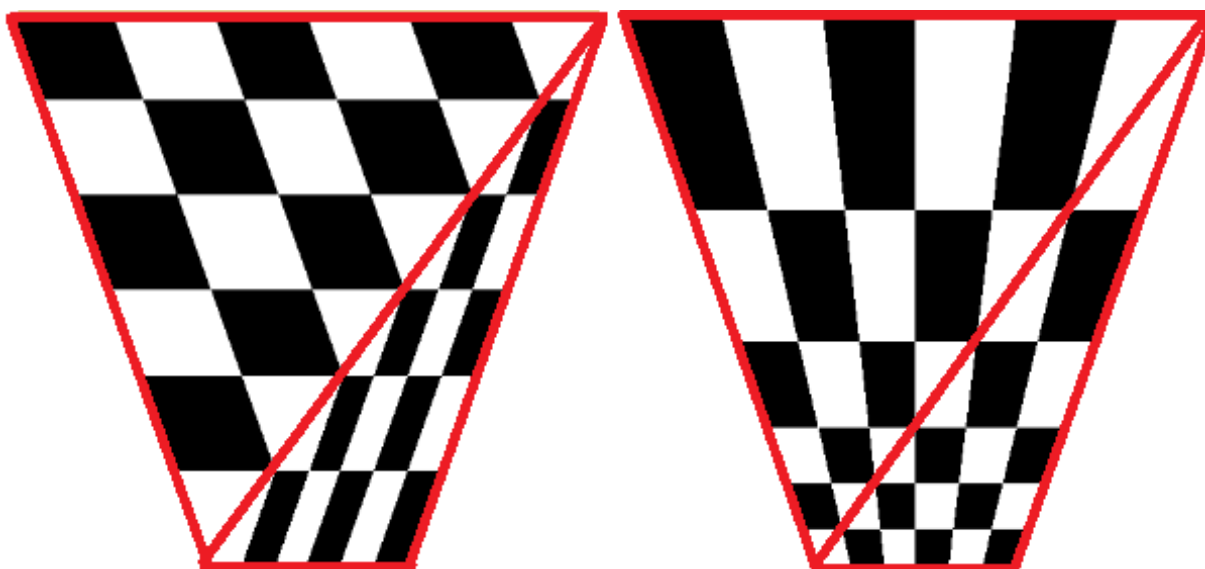


Figura 2.7: Interpolare liniară vs Perspectiv-corectă a unei texturi

observa în figura 2.7, interpolarea liniară (afină) între coordonatele UV, oferită standard de către WebGL nu este îndeajuns pentru a obține efectul dorit. Pentru a obține un rezultat ca în partea dreaptă a figurii este nevoie de a lua în calcul toate cele patru vârfuri ale poligonului, și de a modifica coordonatele UV în funcție de acestea. Pentru a obține acest efect, a fost calculată o pondere pentru modificarea poziției coordonatelor UV pentru fiecare vârf. Aceste ponderi sunt determinate de către distanța fiecărui vârf către centrul de greutate al poligonului. Pentru a determina centrul de greutate a poligonului, este nevoie doar de trasarea celor două diagonale determinate de vârfurile opuse ale poligonului și identificarea locului unde acestea două se intersectează.

5. Unealta de inversare a culorilor - Această unealtă permite utilizatorului inversarea culorilor unei imagini. Acest efect este obținut prin modificarea valorilor din canalele RGB (roșu, verde și albastru), din Fragment Shader. Fiecare canal de culoare poate avea valori cuprinse între 0 și 1. Deci implementarea se rezumă în această linie suplimentară, după efectuarea tuturor efectelor din Fragment Shader:

```
color = vec4(1.0 - color.r, 1.0 - color.g, 1.0 - color.b,  
             clamp(color.a * ubo.opacity, 0.0, 1.0));
```

6. Unealta de ajustare a nuanței, saturației și luminozității - Afișează către utilizator trei slidere ce manipulează cele trei attribute ale imaginii.

7. Unealta de suprascriere a culorii - Modifică valoarea fiecărui pixel cu o valoare RGB determinată de către utilizator. Singurul canal nemodificat este A (alpha - asociat opacității).

8. Unealta de modificare a opacității - Modifică valoarea canalului alpha al fiecărui pixel.

9. Ghid poziționare - Afișează coordonatele fiecărui vârf al poligonului.

10. Unealtă încărcare model 3D - Permite încărcarea unui obiect 3D în scenă. Primul pas în implementarea acestei unelte este alegerea unui format 3D. Pentru acest proiect am decis să fie folosit formatul .obj, din 2 motive: este unul din cele mai populare formate de fișier 3D, acesta fiind suportat de către multe programe de modelare 3D cum ar fi Google SketchUp, 3DS Max sau Maya; iar al doilea criteriu ce m-a convins este faptul că orice fișier .obj poate fi deschis cu orice editor de text, lucru ce permite o parsare mai ușoară a datelor. Un dezavantaj al stocării datelor în format text versus format binar este legat de performanță. O parsare a textului va lua întotdeauna mai mult timp decât încărcarea directă a datelor într-o structură, însă, pentru acest proiect, ne așteptăm să fie încărcate obiecte 3D ce conțin prea multe detalii. O scurtă prezentare a formatului .obj:

În fișierul .obj, fiecare linie începe cu o etichetă ce ajută la clasificarea datelor:

- v 0.0 1.0 2.0 - descrie un vertex și cele 3 coordonate ale acestuia (x, y, z)
- vt 0.0 1.0 - reprezintă o coordonată UV
- vn 0.0 1.0 0.5 - descrie vectorul normală

- $f_1/2/3/4/5/6/7/8/9$  - descrie un triunghi. Acest triunghi este format dintr-o serie de vertecși, coordonate UV și normale, sub forma  $f_v/vt/vn \ v/vt/vn \ v/vt/vn$ .

După parsare, aceste date se vor regăsi într-o structură ce va descrie mulțimea tuturor vectorilor. Această structură va fi apoi asociată unui Vertex Buffer, ce va putea fi folosit pentru desenarea obiectului de către procesorul grafic. Pentru randarea obiectelor 3D astfel încât acestea să își păstreze proporțiile originale, este nevoie de folosirea proiecției în perspectivă. O problemă ce este generată de o implementare naivă a proiecției în perspectivă în proiectul nostru, este incompatibilitatea acesteia cu proiecția ortogonală, folosită de layere. Dacă ar fi să folosim fără ajustări aceste două proiecții în același timp, am putea observa că obiectele 3D vor fi mereu acoperite de către layere, indiferent de ordinea acestora. Pentru a depăși această limitare, au fost folosite Framebuffer. Un framebuffer este un obiect folosit de WebGL pentru a descrie o imagine. Acest obiect poate avea mai multe atașamente, însă, în proiectul nostru, ne interesează doar cel de culoare. Ideea este de a asocia fiecărui obiect 3D încărcat în scenă câte un framebuffer și un model de layer. Când se va ajunge la pasul de desenare a scenei, obiectele 3D vor lua prioritate la desenare, însă acestea vor fi randate pe framebuffer-ul asociat lor. Odată ce acest lucru a avut loc, fiecărui model îi va fi asignată textura framebuffer-ului, rezultând layer-ul final. Folosind această tehnică, avem și avantajul compatibilității fiecărui efect grafic generat de uneltele descrise anterior, fără nici un efort suplimentar. Rezultatul este similar cu partea stângă a imaginii de jos:

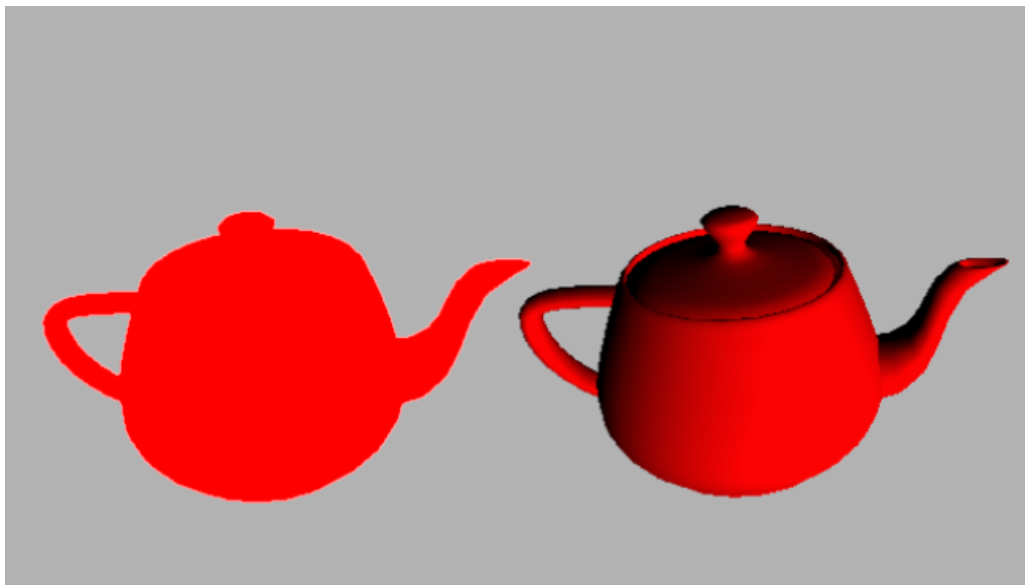


Figura 2.8: Iluminarea scenei

După cum se poate observa, tot ce am făcut până acum nu este îndeajuns pentru

a obține un rezultat satisfăcător. Din acest motiv, este nevoie de un pas final, acesta fiind iluminarea scenei. Pentru a obține o aproximare a modului prin care lumina se deplasează în realitate, fără a necesita resurse hardware exagerate, este îndeajuns să folosim o parte din iluminarea Phong. În mod normal, iluminarea Phong este compusă din trei componente: lumina ambientală, lumina difuză și cea speculară. Însă, pentru a păstra fluiditatea editor-ului, putem renunța la componenta speculară, lucru ce ne aduce un rezultat asemănător părții din dreapta a figurii 2.8. Este destul de ușor de implementat tot acest lucru prin intermediul Fragment Shader-ului:

$$O = (A + D); \quad A = C * a; \quad D = C * R * I;$$

După cum se poate vedea în formula de mai sus, culoarea finală (O) este formată din suma celor două componente: componenta ambientală (A) și componenta difuză (D). Componenta ambientală este obținută prin înmulțirea culorii inițiale (C) cu un factor ambiental (a), în cazul nostru, 0.2, pentru a obține culoarea dorită în părțile ce nu sunt atinse de sursa noastră de lumină virtuală. Componenta difuză este alcătuită din culoarea inițială (C), intensitatea sursei de lumină (I) și (R) ce reprezintă produsul scalar între vectorul normală și vectorul ce indică direcția luminii, amândoi vectori fiind normalizați.

11. Unealta de rotire a obiectelor 3D - Această unealtă permite rotirea unui obiect 3D cu ajutorul mouse-ului și a tastelor x, y și z, pentru selectarea axei de rotire dorite. În cod, acest lucru este obținut prin rotirea World Matrix-ului asociat obiectului 3D.

12. Salvarea imaginii - Afișează o fereastră modală ce permite utilizatorului să specifice un nume pentru imaginea creată în editor ce urmează a fi salvată în galerie. Pentru implementarea acestei funcționalități este îndeajuns să folosim funcția din backend pentru încărcarea imaginilor normale, pentru canvas-ul nostru.

```
this.glContext = this.canvas.getContext('webgl2',
                                         {preserveDrawingBuffer:true});
```

Un alt lucru ce trebuie făcut pentru ca tot procesul de salvare a imaginii să funcționeze corespunzător, este activarea atributului `preserveDrawingBuffer` atunci când construim contextul WebGL. Acest atribut comunică API-ului că imaginea din buffer ar trebui păstrată, curățarea buffer-ului fiind făcută doar explicit.

## 2.1.5 Galeria

În această secțiune a platformei, utilizatorul își poate vizualiza imaginile sale. Pagina principală a galeriei afișează toate imaginile încărcate de utilizator sub formă de pictograme (thumbnail-uri).

Selectarea unei pictograme de pe pagina principală va trimite utilizatorul către pagina de prezentare a fotografiei. Aici, primul lucru ce poate fi întâlnit este imaginea propriu-zisă, ce poate fi expandată sau restrânsă prin intermediul suprapunerii cursorului peste aceasta. Mai jos, se află secțiunea de detalii. În această secțiune sunt afișate informații precum titlul imaginii, descrierea, data încărcării, rating-ul și persoanele din fotografie (în cazul în care algoritmul de detecția facială a găsit persoane). Ultima parte a paginii conține secțiunea de comentarii.

## 2.1.6 Secțiunea Explore

În această secțiune sunt prezentate toate fotografiile prezente pe platformă, de către toți utilizatorii.

Primul rând al paginii afișează într-un carusel de imagini cele mai noi fotografii încărcate pe platformă.

Al doilea rând dă posibilitatea utilizatorului de a căuta fotografii cu ajutorul unor filtre. Acesta poate căuta fotografii după nume, interval de timp și interval de rating.

Ultimul rând este similar primului, cu excepția că acesta afișează fotografiile cu cel mai bun rating.

## 2.2 Detalii de implementare ale modulului de Back-end

### 2.2.1 Tehnologii utilizate

Modulul de back-end are ca scop servirea tuturor cererilor celorlalte două servere. Paradigma de programare utilizată în crearea acestui modul este REST (Representational State Transfer). În această paradigmă, rezultatul procesării unei cereri direcționează spre obținerea unei reprezentări a unei resurse. Formatul reprezentării poate fi desemnat de mai multe tipuri MIME (Multipurpose Internet Mail Extensions). În proiectul nostru, va fi folosit tipul `application/json`. Browser-ul web

poate interacționa cu aceste resurse prin intermediul verbelor: GET, POST, PUT, DELETE, PATCH și OPTIONS.

Ca limbaj de programare, modulul de back-end folosește C#, parte din framework-ul de dezvoltare a aplicațiilor .NET Core. Acest mediu de programare oferă multe facilități, una importantă fiind cea a pachetelor NuGet.

NuGet este un manager de pachete ce permite dezvoltatorilor instalarea ușoară a modulelor software ce pot fi folosite pentru a obține diverse funcționalități în cod.

Două din pachetele importante în proiectul nostru sunt `AspNetCore.Server` și `EntityFrameworkCore`. `AspNetCore.Server` ne pune la dispoziție o modalitate ușoară și rapidă de a pune bazele unui server de tip REST. Cel de-al doilea pachet, `EntityFrameworkCore`, permite comunicarea cu baza de date `SQLServer` prin intermediul unei mapări bazate pe obiecte.

În `EntityFramework`, o clasă C# poate fi mapată către baza de date pentru a reprezenta o tabelă, coloanele acesteia fiind descrise de către atributele clasei. Orice obiect al acestei clase poate fi adăugat ca rând în baza de date, sau modificat, prin intermediul unei încapsulări de metode ce o numim repository.

```
public class Account
{
    [Required, Key]
    public int Id { get; set; }
    [Required]
    public string FirstName { get; set; }
    [Required]
    public string LastName { get; set; }
    [Required]
    public string Email { get; set; }
    [Required]
    public string Password { get; set; }

    public string Address { get; set; }
    public string Country { get; set; }
    public string City { get; set; }
}
```



Mai sus avem drept exemplu tabela Accounts. Cheia primară și coloanele cărora nu pot fi asignate valori NULL sunt descrise prin ajutorul anotărilor: Key și Required.

### 2.2.2 Structura

Modulul back-end se împarte în 4 mari categorii:

- **Controllers** - o mulțime de controllere ce au sarcina de a gestiona cererile efectuate de front-end și de a furniza resursele necesare fiecărei cereri.
- **Middleware** - implementarea componentelor software ce au rolul de a se atașa procedurii de cereri și răspunsuri.
- **Models** - clasele ce descriu tabelele bazei de date.
- **Repositories** - o colecție de clase ce au rolul de a intermedia lucrul cu baza de date.

Aplicația noastră dispune de 5 controllere: Account, ce se ocupă cu servirea cererilor legate de datele unui utilizator al platformei; Activity, controller ce furnizează date despre activitățile de pe platformă (comentarii, rating-uri); Photo, controller-ul ce se ocupă de toate acțiunile legate de imagini; Token, ce are responsabilitatea de a gestiona autentificarea și Upload care se ocupă cu procesul de încărcare a imaginilor pe platformă.

Pe secțiunea de Middleware, proiectul conține JWT (JSON Web Token). JWT oferă posibilitatea de a transmite în mod sigur informații între două părți, folosind un obiect JSON. Aceste informații pot fi verificate deoarece sunt semnate digital. În aplicația noastră, JWT este folosit pentru autorizare. Un utilizator odată autentificat, primește un token JWT, ce are rolul de a permite accesul către diverse rute și servicii ale platformei.

În partea de Modele, cele patru clase ce descriu fiecare tabelă din baza de date.

Iar ultima componentă, Repositories, conține patru repozitorii, asociate fiecărui model. Fiecare repozitoriu conține metode fundamentale precum: Add, pentru adăugarea unui obiect nou; GetById pentru obținerea unui obiect din baza de date folosind un ID de referință; Update, pentru actualizarea atributelor unui obiect. Pe lângă aceste metode, un repozitoriu poate conține și alte metode de specialitate.

## 2.3 Detalii de implementare ale server-ului de fișiere

### 2.3.1 Tehnologii utilizate

Similar cu back-end-ul, server-ul de fișiere este programat în limbajul C#, ce aparține de .NET Core. O altă dependență a acestei aplicații este Python, versiunea 3.6, necesară pentru invocarea scriptului folosit în detecția facială, despre care vom vorbi mai târziu.

### 2.3.2 Rolul

În asamblul proiectului nostru, server-ul de fișiere are trei scopuri:

- Reducerea din sarcinile back-end-ului, care deja se ocupă de funcții critice aplicației noastre
- Furnizarea imaginilor către client în grade de calitate diferite, unul fiind cel original, iar cel de-al doilea fiind folosit pentru pictograme (thumbnail).
- Rularea detecției faciale asupra tuturor imaginilor noi încărcate, și furnizarea datelor obținute către back-end, odată cu terminarea procesului de încărcare.

Procesul de încărcare a unei imagini începe cu o cerere făcută de către client-ul web către back-end. Back-end-ul, odata ce a obținut această cerere, el o înaintează către server-ul de fișiere, printr-un serviciu accesibil doar de acesta. Server-ul de fișiere odată ce a primit cererea de upload, generează o cheie ce poate fi folosită pentru încărcarea imaginilor pe acesta. Această înlănțuire de cereri între serverele platformei este pusă în aplicare din motive de securitate. Lucrul pe care vrem să îl prevenim prin acest protocol este accesul direct la server-ul de fișiere, fără cunoștința back-end-ului. Fără aceste măsuri, o persoană rău intenționată ar putea trimite comenzi de încărcare a unor imagini către server, cu scopul de a umple spațiul de stocare în zadar.

Imediat după primirea cheii de acces, se poate iniția încărcarea propiu-zisă a imaginilor. Imaginile se vor serializa într-un obiect JSON, iar acestui obiect îi va fi atașată cheia primită.

În server-ul de fișiere, această cheie este consumată iar stocarea imaginii începe. Primul pas este crearea pictogramei. Orice pictograme ce vrem să fie afișate pe paginile web ale platformei trebuie să îndeplinescă condiția de a avea lungimea și lățimea sub

200 pixeli. Pentru a obține acest lucru, fiindu-ne dată o imagine de orice dimensiune, trebuie calculată o valoare ce o vom numi factor de restrângere. Folosind acest factor, putem selecta un număr arbitrar de pixeli în formație de pătrat. Din acești pixeli, putem calcula folosind media aritmetică valoarea fiecărui canal RGBA ce va alcătui un pixel din pictogramă.

Următorul pas este analizarea imaginii cu scopul de a detecta fețe. Acest lucru este implementat folosind o bibliotecă disponibilă în Python, numită OpenCV. Algoritmul de detectare al feței folosit în proiectul nostru este Viola-Jones. Acest algoritm se folosește de o serie de caracteristici ce pot fi detectate cu ajutorul unor suprafețe dreptunghiulare. Aceste caracteristici se numesc caracteristici Haar. Aceste suprafețe dreptunghiulare pot fi de două tipuri, fie întunecate, fie luminate. Pentru ca algoritmul să determine valoarea unei caracteristici Haar pe o anumită porțiune de imagine, se vor însuma pixelii dreptunghiurilor luminate, apoi aceasta se va scădea din suma pixelilor din dreptunghiurile întunecate.

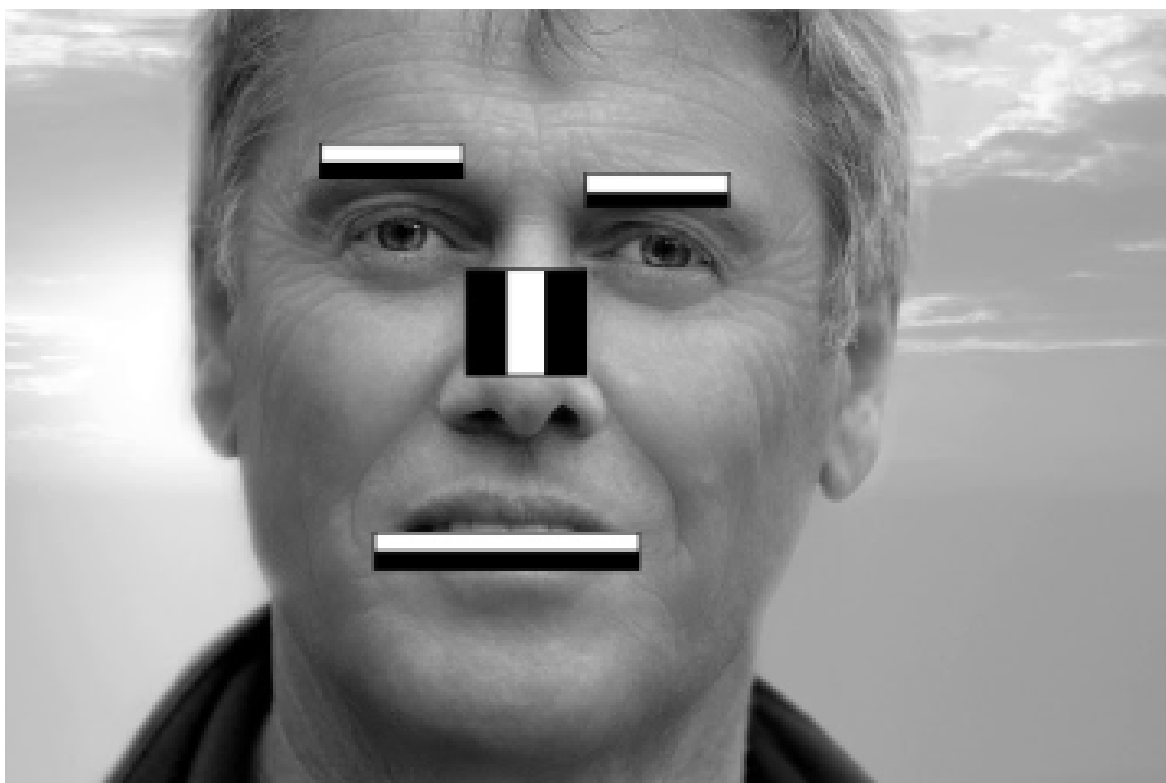


Figura 2.9: Exemplificare caracteristici Haar

În figura de mai sus putem observa câteva exemple de caracteristici Haar:

- Fruntea va fi mai deschisă la culoare decât sprâncenele
- Podul nasului va fi mai deschis la culoare decât baza

- Buzele vor fi mai închise la culoare decât dinții.

Faptul că folosim biblioteca OpenCV scoate la iveală o nouă dificultate. Aceasta constă în incorporarea scriptului Python ce se va ocupa de detecția fețelor, în mediul nostru deja creat în C#. Soluția ce s-a adoptat pentru acest proiect a fost de a concepe scriptul Python în așa fel încât input-ul și output-ul să persiste indiferent dacă scriptul a rulat încă sau nu. Input-ul, calea către imagine, se va transmite ca argument tip linie de comandă, putând fi citit cu ușurință folosind `sys.argv`. După rularea algoritmului output-ul va fi direcționat către un fișier text: `output.data`.

Astfel, mai rămâne doar pasul de invocare a scriptului și citirea fișierului `output.data` după execuția acestuia. Acest pas a fost implementat prin crearea unui proces nou, Command Prompt.

```
Process process = new Process();
ProcessStartInfo processStartInfo = new ProcessStartInfo();
processStartInfo.CreateNoWindow = true;
processStartInfo.FileName = "cmd.exe";
processStartInfo.RedirectStandardInput = true;
processStartInfo.RedirectStandardOutput = true;
processStartInfo.UseShellExecute = false;
process.StartInfo = processStartInfo;
process.Start();

process.StandardInput.WriteLine("python .\\face_detect.py [img]");
process.StandardInput.Flush();
process.StandardInput.Close();

process.WaitForExit();
```

În codul de mai sus este exemplificat procedeul de creare a noului proces. Acesta se crează fără fereastră, iar bufferele de input și output sunt redirecționate către aplicația noastră. După ce pornim procesul, vom scrie în buffer-ul de input comanda de invocare a scriptului python cu calea imaginii noastre dată ca argument, iar apoi vom aștepta ca procesul să se închidă, urmând să preluăm datele din fișierul creat de script.

Acești pași fiind efectuați, server-ul de fișiere va trimite informațiile despre imagine (adresa unde aceasta poate fi găsită, atât în format original, cât și thumbnail; împreună cu datele rezultate din detecția facială) către back-end, acestea fiind înregistrate în baza de date.

# Capitolul 3

## Rularea aplicației

În rândurile ce urmează va fi prezentat un ghid de instalare a platformei. Pentru ca aceasta să ruleze corect, toate cele trei servere trebuie să fie instalate și executate corespunzător.

### 3.1 Server-ul Back-end

Primul pas ce trebuie efectuat este instalarea .NET Core 2. Acest lucru este necesar pentru compilarea și rularea proiectului. În cazul în care se dorește doar executarea server-ului, fișierele compilate fiind deja furnizate, este îndeajuns doar instalarea .NET Core Runtime.

O altă dependență a acestei aplicații este SQL Server. Aceasta trebuie instalat dacă nu se regăsește pe sistemul unde va fi rulată aplicația.

În cazul în care se dorește modificarea port-ului la care va rula aplicația, acest lucru se poate face modificând următoarea linie din fișierul Program.cs:

```
var server = new WebHostBuilder()  
    .UseKestrel()  
    .UseStartup<Program>()  
    .UseUrls("http://localhost:5000")  
    .Build();
```

În mod normal, cele trei servere ale proiectului nostru folosesc port-uri diferite: Portul :5000 este folosit de back-end, portul :5001 este folosit de server-ul de fișiere, iar :4200 este folosit de către front-end. În cazul în care pe sistemul unde va fi găzduită

aplicația mai există alte programe ce folosesc același port, va fi necesară schimbarea unuia.

Acum că aceste detalii au fost discutate, putem trece la rularea aplicației. Acest lucru se poate face executând instrucțiunea "dotnet backend.dll" în linia de comandă.

Figura 3.1: Fereastra server-ului Back-end

Dacă rezultatul din fereastră este similar cu cel din figura de mai sus, înseamnă că server-ul rulează corespunzător.

Pentru a închide server-ul, se va selecta fereastra acestuia și se va acționa combinația de taste CTRL + C.

## 3.2 Sever-ul de fișiere

Server-ul de fișiere fiind dezvoltat în mediul .NET Core, este similar cu cel de back-end. În cazul acestuia, dependența de SQL Server nu mai există, însă, apare una nouă, anume Python 3.6. Odată ce aceasta a fost instalat, se poate rula server-ul, în mod similar ca la back-end, cu următoarea comandă:

```
dotnet fileserver.dll
```

Similar cu back-end-ul, acest server poate fi închis acționând combinația de taste CTRL + C.

### 3.3 Server-ul Front-end

Pentru instalarea server-ului de front-end sunt necesari o serie de pași mai diferiți.

Primul pas constă în instalarea Node.js și npm. Odată ce aceste lucruri au fost aduse pe sistem, se poate începe instalarea Angular CLI. Acest lucru se poate face prin deschiderea unei linii de comandă și introducerea comenzii:

```
npm install -g @angular/cli
```

După ce și acest lucru a fost realizat, va trebui să instalam toate dependențele front-end-ului. Acest lucru poate fi înfaptuit ușor, prin navigarea către directorul proiectului și deschiderea unei linii de comandă din acest loc, unde vom introduce comanda:

```
npm install
```

Pentru a modifica port-ul la care rulează front-end-ul, avem mai multe opțiuni. O primă opțiune, ce ne oferă o soluție persistentă a modificării port-ului ar fi modificarea fișierului `.angular-cli.json` din directorul central al proiectului și adăugarea următoarelor linii în secțiunea "defaults":

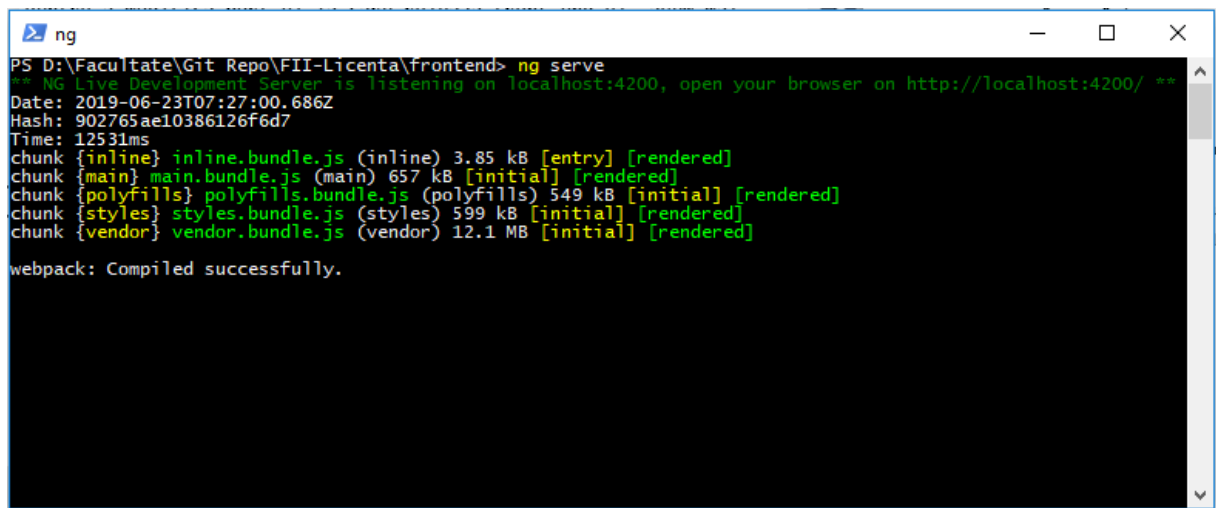
```
"serve": {  
  "port": 4201  
}
```

O altă opțiune este rularea aplicației cu un parametru adițional în care se va specifica port-ul dorit. În mod normal, fără acest parametru, aplicația va rula pe port-ul 4200.

Aplicația poate fi acum rulată prin introducerea comenzii "ng serve" sau "ng server -port (nr\_port)" în linia de comandă deschisă din directorul proiectului.

Imediat după ce a fost introdusă această comandă, server-ul va începe procesul de încărcare a tuturor resurselor necesare, și va afișa un mesaj similar cu cel din Figura 3.2 când acesta este pregătit să servească cereri.





```
PS D:\Facultate\Git Repo\FII-Licenta\frontend> ng serve
** NG Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
Date: 2019-06-23T07:27:00.686Z
Hash: 902765ae10386126f6d7
Time: 12531ms
chunk {inline} inline.bundle.js (inline) 3.85 kB [entry] [rendered]
chunk {main} main.bundle.js (main) 657 kB [initial] [rendered]
chunk {polyfills} polyfills.bundle.js (polyfills) 549 kB [initial] [rendered]
chunk {styles} styles.bundle.js (styles) 599 kB [initial] [rendered]
chunk {vendor} vendor.bundle.js (vendor) 12.1 MB [initial] [rendered]
webpack: Compiled successfully.
```

Figura 3.2: Fereastra server-ului Back-end

Pentru a închide server-ul, se va acționa combinația de taste CTRL + C. Server-ul va cere o confirmare pentru această comandă. Această confirmare poate fi comunicată cu Y pentru închidere sau N pentru revocare.

Toate serverele fiind puse în funcțiune, putem acum accesa platforma noastră prin navigarea către <http://localhost:4200/>, dacă port-ul implicit al front-end-ului nu a fost schimbat.

# Concluzii

Având în vedere lucrurile prezentate în paginile anterioare, putem observa că implementarea platformei de gestionare și editare a imaginilor, folosind tehnologii precum Angular și .NET Core, s-a desavârșit cu succes.

În implementarea acestei platforme am folosit cunoștințe teoretice și practice dobândite în facultate, dar și din studiu propriu.

Consider că acest sistem pe care l-am creat se poate dezvolta în continuare, și poate deveni mult mai complex.

În primul rând, editor-ul poate fi îmbunătățit prin adăugarea mai multor unelte de modificare a imaginilor. Câteva idei în acest sens ar putea fi îndeplinite prin introducerea suportului pentru desenarea pe un layer, folosind diverse efecte ce imită ustensile de scris precum creionul, stiloul, marker-ul, s.a.m.d; introducerea uneltei de ștergere a unei arii determinate de către utilizator dintr-un layer; suport pentru text; și suport pentru efecte mai complexe precum blur-ul gaussian.

O altă cale de dezvoltare a aplicației poate fi extinderea galeriei și a secțiunii explore. Lucruri ce ar putea îmbunătăți această parte a platformei ar putea fi introducerea de categorii în care pot fi clasificate imaginile și suportul pentru mai multe date relevante unei imagini, cum ar fi locația unde s-a produs aceasta.

Din punctul meu de vedere, consider că lucrarea și-a atins scopul de implementa sistemul dorit.

# Bibliografie

[1] B. Sabin-Corneliu, "Dezvoltarea de aplicații Web prin REST"

<https://profs.info.uaic.ro/busaco/teach/courses/web/presentations/web11ServiciiWeb-REST.pdf>

[2] An introduction to WebGL

<https://thoughtbot.com/blog/an-introduction-to-webgl>

[3] RasterTek Direct3D and OpenGL materials

<http://www.rastertek.com/>

[4] Viola-Jones object detection

[https://en.wikipedia.org/wiki/Viola%E2%80%93Jones\\_object\\_detection\\_framework](https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework)

[5] Wavefront .obj file

[https://en.wikipedia.org/wiki/Wavefront\\_.obj\\_file](https://en.wikipedia.org/wiki/Wavefront_.obj_file)

[6] Texture mapping and perspective correctness

[https://en.wikipedia.org/wiki/Texture\\_mapping](https://en.wikipedia.org/wiki/Texture_mapping)

[7] World, View and Projection Transformation Matrices

[http://www.codinglabs.net/article\\_world\\_view\\_projection\\_matrix.aspx](http://www.codinglabs.net/article_world_view_projection_matrix.aspx)

[8] Basic Lighting - Phong

<https://learnopengl.com/Lighting/Basic-Lighting>