UNIVERSITATEA “LUCIAN BLAGA” DIN SIBIU

FACULTATEA DE INGINERIE

DEPARTAMENTUL DE CALCULATOARE ŞI INGINERIE ELECTRICĂ

**PROIECT DE DIPLOMĂ**

Îndrumător: Conf.univ.dr.mat. Radu George CREŢULESCU

Absolvent: Șufană Adrian-Daniel

Specializarea: Tehnologia Informației

* Sibiu, 2023 –

UNIVERSITATEA “LUCIAN BLAGA” DIN SIBIU

FACULTATEA DE INGINERIE

DEPARTAMENTUL DE CALCULATOARE ŞI INGINERIE ELECTRICĂ

# APLICAȚIE ORGANIZARE PORTOFOLIU

Îndrumător: Conf.univ.dr.mat. Radu George CREŢULESCU

Absolvent: Șufană Adrian-Daniel

Specializarea: Tehnologia Informației

**Cuprins**

[APLICAȚIE ORGANIZARE PORTOFOLIU 2](#_Toc134275884)

[**1.** **Prezentare** 5](#_Toc134275885)

[**1.1** **Noțiuni introductive** 5](#_Toc134275886)

[**1.2** **Motivația alegerii temei** 5](#_Toc134275887)

[**1.3** **Scop și obiective** 6](#_Toc134275888)

[**1.4** **Realizări similare acestei teme** 6](#_Toc134275889)

[**2** **Teorie și tehnologii utilizate** 8](#_Toc134275890)

[**2.1** **Visual Studio Code** 8](#_Toc134275891)

[**2.2** **Arhitectura MERN** 9](#_Toc134275892)

[**2.2.1** **MongoDB** 9](#_Toc134275893)

[**2.2.2** **Express.js** 10](#_Toc134275894)

[**2.2.3** **React** 10](#_Toc134275895)

[**2.2.4** **Node.js** 11](#_Toc134275896)

[**2.3** **API-uri / Librării** 11](#_Toc134275897)

[**2.3.1** **React Router și Express API** 11](#_Toc134275898)

[**2.3.2** **Hooks** 13](#_Toc134275899)

[**2.3.3** **Mongoose** 14](#_Toc134275900)

[**2.3.4** **Multer** 15](#_Toc134275901)

[**2.3.5** **JWT** 16](#_Toc134275902)

[**2.3.6** **EXIF** 16](#_Toc134275903)

[**2.3.7** **DotEnv** 17](#_Toc134275904)

[**3** **Realizarea aplicației** 18](#_Toc134275905)

[**3.1** **Mediul de dezvoltare** 18](#_Toc134275906)

[**3.2** **Dezvoltarea serverului** 20](#_Toc134275907)

[**3.2.1** **Configurare fișier .env** 20](#_Toc134275908)

[**3.2.2** **Conectarea la baza de date** 20](#_Toc134275909)

[**3.2.3** **Configurare server** 21](#_Toc134275910)

[**3.2.4** **Configurarea rutelor** 24](#_Toc134275911)

[**3.2.4.1** **Configurarea metodelor de tip controller** 24](#_Toc134275912)

[**3.2.5** **Configurarea securității cu JWT** 29](#_Toc134275913)

[**3.2.6** **Configurarea modelelor** 31](#_Toc134275914)

[**3.3** **Dezvoltarea clientului** 34](#_Toc134275915)

[**3.3.1** **Configurarea rutelor** 34](#_Toc134275916)

[**3.3.2** **Crearea temelor alb/negru** 36](#_Toc134275917)

[**3.3.3** **Modul înregistrare** 39](#_Toc134275918)

[**3.3.4** **Modul logare** 41](#_Toc134275919)

[**3.3.5** **Pagina Social Media** 42](#_Toc134275920)

[**3.3.6** **Pagina Galerie** 44](#_Toc134275921)

[**3.3.6.1** **Încărcare imagini** 48](#_Toc134275922)

[**3.3.6.2** **Metode gestionare distribuire/aprecieri/elimare imagini** 50](#_Toc134275923)

[**3.3.6.3** **Extragerea metadata din imagini folosind exif api** 54](#_Toc134275924)

[**3.3.6.4** **Extragerea culorilor dominante din imagine** 55](#_Toc134275925)

[**3.3.6.5** **Filtrare utilizând metadata** 56](#_Toc134275926)

[**3.3.6.6** **Filtrare utilizând culorile dominante** 58](#_Toc134275927)

[**3.3.6.7** **Sortare** 61](#_Toc134275928)

[**3.3.6.8** **Creearea modurilor de vizualizare grid a galeriei** 61](#_Toc134275929)

[**3.3.7** **Modul rapoarte** 61](#_Toc134275930)

[**4** **Concluzii și dezvoltări ulterioare** 61](#_Toc134275931)

[**4.1** **Concluzii** 61](#_Toc134275932)

[**4.2** **Dezvoltări ulterioare** 61](#_Toc134275933)

[**5** **Bibliografie** 62](#_Toc134275934)

[**5.1.1.1** 62](#_Toc134275935)

[**5.2** **Webliografie** 62](#_Toc134275936)

1. **Prezentare**
   1. **Noțiuni introductive**

În viața de zi cu zi, peisajele și arhitectura din jurul nostru continuă să uimească prin modul în care sunt percepute de către noi, oamenii. Dar aceste imagini capturate de ochii noștrii rămân doar amintiri in momentul în care nu mai sunt subiectul principal vizual. Din fericire există modalități prin care aceste cadre pot fi păstrate fizic sau digital. Acest lucru este realizat prin fotografie.

Fotografia este un concept vechi, prima fotografie fiind realizată în anul 1827 de către fizicianul francez Joseph-Nicéphore Niépce. El a pus temelia acestei arte care până în prezent a evoluat la un nivel foarte înalt atât din punct de vedere calitativ dar și cantitativ. Dacă prima fotogafie realizată nu era decât o imagine foarte neclară, în momentul de față imaginile sunt tot mai detaliate, pline de culoare și de emoții transpuse prin intermediul unui obiectiv.

Fotografia a schimbat percepția oamenilor asupra acestor peisaje. Fotografia reprezintă arta și tehnica de a captura momente din viața noastră, folosindu-se de lumină. Prin intermediul fotografiei putem captura o varietate de peisaje, momente unice și speciale care pot fi partajate și apreciate de public, sau păstrate în propria galerie.

* 1. **Motivația alegerii temei**

Alegerea temei denumită “Aplicație de organizare portofoliu” a pornit de la o pasiune personală pentru fotografie și de la o necesitate pentru cunoașterea în detaliu a tehnicilor complexe necesare pentru realizarea unor fotografii corecte. Pe scurt, aplicația a fost creată pentru a oferii o platformă online unde fotografii amatori și profesioniști pot să își expună lucrările și să le împărtășească publicului.

În plus, galeria oferă o oportunitate fotografilor de a-și dezvolta abilitățile utilizând detaliile extrase din spatele fiecărei fotografii expuse în aplicație, astfel își pot perfecționa munca. Utilizând această idee, fotografii sunt îndemnați automat la aprofundarea cunoștințelor legate de tehnica fotografiei și la realizarea de imagini cât mai corecte și impresionante

Un ultim motiv, dar de o importanță la fel de mare ca celelalte, este interacțiunea între oameni pasionați de același domeniu. Astfel se păstrează o colectivitate unită prin intermediul căreia oricine poate să fie îndemnat să își imbunătățească tehnicile prin intemediul feedback-ului primit. Totodată fotografii pot colabora în mod facil, împărtășind experiențele. Această interacțiune poate să comduca la dezvoltarea unei comunități mari, active, unde membrii se pot suscine reciproc și se pot aprecia unii pe altții.

În concluzie, motivul pentru care aplicația a fost creeată este legat de pasiunea pentru fotografie și de dorința de a creea o colectivitate în intermediul căreia această pasiune poate fi împărtășita.

* 1. **Scop și obiective**

Scopul principal este de a creea un mediu online unde fotografii profesioniști sau amatori pot să își împărtășească creațiile cu publicul interesat de acest domeniu. Aplicația este în principal destinată publicului care deține cunoștințe legate de fotografie deoarece dispune de noțiuni tehnice care necesită documentare pentru a fi înțelese.

În același timp, aplicația poate fi utilizată și de persoane care nu au cunoștințe de fotografie, dar care doresc să admire fotografiile uimitoare realizate de fotografi din toată lumea. Pentru aceștia aplicația va servii ca o galerie simplă dar plină de imagini spectaculoase.

Pentru îndeplinirea acestui scop, galeria online își propune să îndeplinească următoarele obiective:

* Determinarea tehnologiilor care ne permit implementarea acestei aplicații
* Descrierea aplicației
* Implementarea aplicației care are la bază două obiective principale:
* Să pună la dispoziție un mediu ușor și sigur unde să poată încărca și partaja imagini
* Să puna la dispoziție opțiuni diverse de organizare și sortare a fotografiilor pe baza anumitor criterii definite de noțiuni generale precum dată, locație dar și prin criterii specifice precum metadata-ul fiecarei fotografii
* Să pună la dispoziție un sistem de feedback pentru utlizatorii care încarcă fotografii în aplicație și le distribuie publicului
* Să pună la dispoziție o modalitate prin care utilizatorii pot distribuii imaginile din galeria privată pentru a fi vizualizate și analizate de ceilalți utilizatori
* Realizarea documentației
  1. **Realizări similare acestei teme**

Există o multitudine de platforme online care se axează pe acest domeniu și care oferă un mediu virtual pentru distribuirea fotografiilor personale ale utilizatorilor. O diferență majoră între acestea și aplicația “Aplicație de organizare portofoliu” este informația tehnică legată de parametrii utilizați la realizarea fotografiei și posibilitatea sortării imaginilor bazată pe acești parametrii.

Toate aplicațiile care vor fi prezentate în următoarele rânduri permit sortarea și filtrarea imaginilor, dar nu cu ajutorul a parametrilor tehnici din metadata-ul fotografiilor.

Câteva dintre exemplele cele mai cunoscute sunt:

* Flickr[WWW11]
* Instagram[WWW12]
* DevianArt[WWW13]

1. **Teorie și tehnologii utilizate**
   1. **Visual Studio Code**

Visual Studio Code este un editor text dezvoltat de Microsoft pentru diferite platforme precum Windows, macOS și Linux. Aplicația a ajuns foarte populară datorită diferitelor funcționalități puse la dispoziție utilizatorului precum modulul de debugging, prin completarea inteligentă a codului, funcție cunoscută și sub denumirea de autocompletare a codului, prin compatibilitatea sa cu Git și nu în ultimul rând prin multitudinea de extensii disponibile în propria galerie din aplicație.  
Din punct de vedere al limbajelor suportate, Visual Studio Code poate fi folosit cu o multitudine de limbaje de programare precum C/C#/C++/Java/JavaScript/Node.js/Python precum și multe altele.

VS Code a fost prima dată pus la dispoziția publicului in anul 2015 de către Microsoft. Datorită compatibilității dezvoltate cu github, codul sursă al aplicației a fost pus disponibil pe GitHub câteva luni mai târziu, iar una dintre cele mai importante funcționalități ale aplicației, anume extensiile, au fost puse la dispoziție un an mai târziu dupa lansare, în 2016.

La cațiva ani de la debutul impresionant al aplicației, în anul 2021, în colaborare cu GitHub, în VS Code este introdus GitHub Copilot. Acest tool reprezintă un pas foarte important deoarece devine o funcționalitate utilizată la nivel mondial cu beneficii semnificative pentru dezvoltatori datorită suportului impresionant oferit. GitHub Copilot este descris ca fiind un tool bazat pe AI care oferă sugestii de autocompletare a codului pentru programator. Acesta reprezintă un model antrenat pe toate limbajele de programare care apar în bazele de date ale companiei. Având în vedere dimensiunea informațiilor disponibile pe GitHub, modelul este antrenat la un nivel foarte înalt. Totuși, calitatea sugestiilor oferite poate să difere deoarece unele limbaje de programare sunt mai puțin utilizate decât altele și astfel rezulta mai puține informații disponibile pentru model.

În anul 2022, VS Code a fost nominalizat ca fiind cel mai popular IDE. Ținând cont că există aproximativ 70000 de alte tool-uri similare, această nominalizare este una foarte importantă și pune în evidență versatilitatea aplicației.[WWW01]

Avantaje față de alte IDE-uri:

* Varietatea extensiilor oferite
* Gratuit și open-source
* Nivelul la care poate fi personalizată aplicația
* Ușor de utilizat
* Debugging al codului direct în aplicație
* Comunitatea activă
  1. **Arhitectura MERN**

MERN reprezintă o combinație de tehnologii utilizate pentru dezvoltarea aplicațiilor WEB. Reprezintă un acronim pentru MongoDB, Express.js, React și Node.js. Arhitectura MERN ajută dezvoltatorii sa construiască o arhitectură pe trei nivele(front end, back end, database), utilizând JavaScript și JSON.[WWW02]

Diagramă MERN Stack



Figură 1 Diagramă MERN Stack

În mare, MERN reprezintă o soluție full-stack pentru dezvoltatori datorită arhitecturii. Arhitectura MERN este utilizată deoarece crează o legatură simplă între cele trei niveluri de zevoltare, front end, back end și baza de date. Astfel, baza de date stochează date JSON și funcționează deosebit de bine cu Node.js făcând stocarea, manipularea și reprezentarea datelor JSON extrem de versatilă la fiecare nivel de dezvoltare al aplicației. React.js și Express.js care rulează pe Node.js contribuie la completarea aplicațiilor JavaScript finalizând noțiunea de full-stack. Express.js reprezintă un framework care rulează în paralel cu serverul, bazându-se pe cereri și răspunsuri HTTP, iar React.js este un framework front end în JavaScript utilizat la creearea interfețelor pentru utilizator.

* + 1. **MongoDB**

MongoDB reprezintă unul dintre cele 3 niveluri ale arhitecturii MERN. Este o platformă care pune la dispoziție servicii și unelte necesare dezvoltării rapide a aplicațiilor, la nivelul și performanța cerută de utilizatori. A fost fondată în anul 2007 și funcționează pe principiul stocării datelor diferit de bazele de date SQL. Astfel în loc să stocheze date în tebele construite din rânduri și coloane, fiecare înregistrare într-o bază de date MongoDB reprezintă un document BSON, pe scurt o reprezentare binară a datelor. Ulterior aplicațiile pot primii aceste documente într-un format JSON. Reprezintă o modalitate foarte flexibilă oferind variații în structura documentelor, astfel un document poate avea alte documente în interiorul acestuia.

Motivul pentru care utilizatorii aleg să utilizeze MongoDB în schimbul unei baze de date SQL este dat de formatul utilizat, și anume JSON. Acest format mapează obiecte native în majoritatea limbajelor moderne de programare deci devine automat o alegere naturală pentru majoritatea utilizatorilor.

MongoDB a fost dezvoltată pentru utilizatori care construiesc aplicații web și oferă modalități foarte simple de implementare, conectare și utilizabilitate. [WWW03]

* + 1. **Express.js**

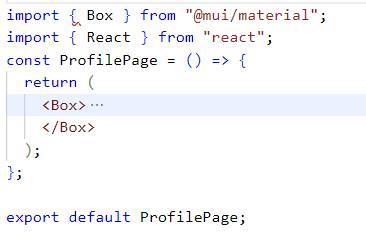
Express.js reprezintă un framework pentru dezvoltarea aplicațiilor împreună cu Node.js pus la dispoziția publicului din anul 2010. Încă de la început a fost pusă la dispoziție ca un software open-source, desemnat dezvoltării aplicațiilor web și a API-urilor, fiind nominalizat ca standard framework pentru Node.js.

Cele trei puncte cheie pe care acest framework se bazează sunt

* Performață maximă
* Utilizabilitatea HTTP
* Routarea
  + 1. **React**

React reprezintă o modalitate simplă și interactivă pentru crearea interfețelor utilizatorilor. React are la bază construirea interfețelor reutilizabile și modulare, folisind o sintaxă specifica numită JSX. Aceasta combină HTML și JavaScript. Utilizând React utilizatorii pot definii starea componentelor, astfel actualizările de stare au loc automat, astfel modificările aduse aplicației sunt construite automate în interfață. Fiind folosit împreună cu React Router, această bibliotecă ajută la o dezvoltare fluidă și ușoară din punct de vedere tehnic al aplicațiilor web.[MOR21]

Exemplu de utilizare a unei componente React implementată în profilePage/index.jsx:



Figură 2 Exemplu utilizare componentă React

* + 1. **Node.js**

Node.js este definit ca un mediu de execuție back end JavaScript utilizat pentru executarea de cod JavaScript în afara unui browser web. Acest tool pune la dispoziția utilizatorilou un mediu în care aceștia pot rula comenzi JavaScript pentru generarea uni conținut dinamic în paginile web.[WWW04]

Rolul principal al acestei componente este de a constuii serverul API care asigură comunicare între acesta, partea de front-end a aplicației și baza de date MongoDB. Principala ocupație a acestei componente este gestionarea cererilor HTTP primite de la client sau a datelor trimise de la baza de date afiliată. Un alt domeniu important în care node.js figurează reprezintă autentificarea si autorizarea. Aici Node.js are rolul de a verifica aceste date și de a proteja datele.

Reprezentând o componentă foarte importantă în cadrul arhitecturii MERN, fiind utilizată pentru gestionarea în mod eficient a cererilor multiple specifice aplicațiilor web, Node.js este esențială în dezvoltarea acestor aplicații, fiind o platformă solidă.

În esență, Node.js asigură o comunicare cât se poate de fluentă între clientul aplicației, server și baza de date MongoDB.

* 1. **API-uri / Librării**
     1. **React Router și Express API**

React Router este o funcționalitate a Express ce ofera posibilitatea navigarii la nivel de client în cadrul aplicației. În aplicațiile web tradiționale, browser-ul realizează o cerere pentru un document dintr-un server web, descarcă datele și le evaluează, iar procesul este repetat de fiecare dată când un utilizator apasă pe un link. [WWW05]

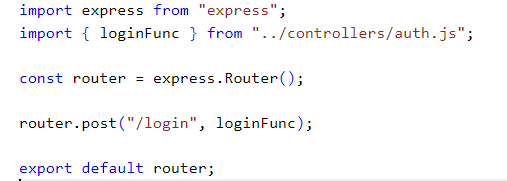
Rutarea la nivel de client are la bază modificarea URL-ului curent fără realizarea altei cereri către serverul aplicației. Această funcționalitate este importantă deoarece aplicația poate afișa altă interfață utilizatorului într-un mod rapid și eficient, utilizând funcția fetch pentru a modifica pagina cu informațiile noi aduse. Astfel se dezvoltă o experiență mult mai rapidă pentru utilizator deoarece browser-ul nu este nevoit să re-evalueze documente JavaScript și CSS de fiecare dată când un nou link este apăsat, sau o nouă interfață trebuie afișată. Routarea la nivel de client este inițializată prin utilizarea tag-ului Router.

Exemplu de implementare a unei componente React Router în cadrul serverului:



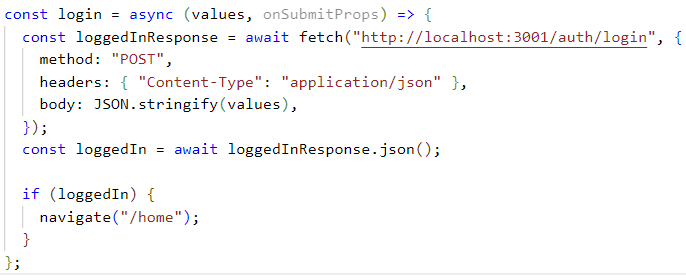
Figură 3 Exemplu utilizare componentă React Router

Exemplu de utilizare a componentei React utilizând express la nivel server:



Figură 4 Exemplu utilizare express la nivel server

Exemplu de utilizare a componentei React utilizând express la nivel client:



Figură 5 Exemplu utilizare express la nivel client

* + 1. **Hooks**

React Hooks reprezintă o modalitate prin care dezvoltatorii pot folosii starea aplicației în diferite componente. În termeni simplii, React Hooks sunt funcții JavaScript capabile să izoleze părți reutilizabile dintr-o componentă funcțională, în acelasi timp avânt capabilitatea de a gestiona efecte secundare. [WWW06]

React pune la dispoziție un număr variat de hooks, precum:

* useState: Acest hook este utilizat pentru a gestiona statusul unei componente. Returnează o valoare de stare de tip boolean, adevărat sau fals, și o funcție utilizată pentru a modifica starea acestei variabile.
* useEffect: Acest hook gestionează efectele secundare ale aplicației precum apelurile API, timere și multe altele.

Exemplu utilizare useState:



Figură 6 Exemplu utilizare useState

Exemplu utilizare useEffect:



Figură 7 Exemplu utilziare useEffect

* + 1. **Mongoose**

Mongoose reprezintă o librărie pentru MongoDB și NodeJS. Este definită ca o librărie ODM(Object Data Modeling), fiind utilizată la validarea schemelor, la gestionarea relațiilor dintre date și la translatarea între obiectele din cod și reprezentarea acestora în MongoDB.

Diferența de stocare a datelor între Mongo și o bază de date SQL:[WWW07]

* Mongo

{  
 “Id”: 1,  
 “FirstName” : “Sufana”

“LastName” : “Adrian”

“Phone”:[{

“Home”:”0777777777”

}]

}

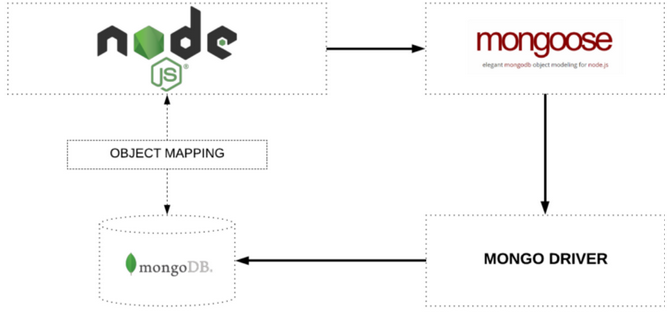
* SQL

Tabel 1.1 Person

|  |  |  |
| --- | --- | --- |
| Id | FirstName | LastName |
| 1 | Sufana | Adrian |

Tabel 1.2 Phone\_Number

|  |  |  |  |
| --- | --- | --- | --- |
| PersonId | PhoneId | PhoneNumber | Type |
| 1 | 1 | 0777777777 | Home |

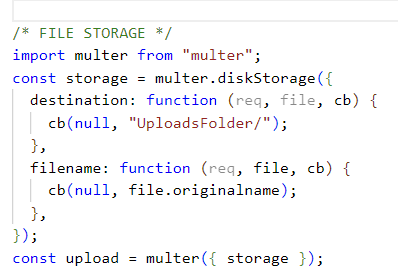


Figură 8 Maparea obiectelor între Node și MongoDB prin intermediul Mongoose

* + 1. **Multer**

Multer poate fi definit ca un integrator a cărui job este de a gestiona încărcarea de imagini în Node.js într-o manieră mult mai simplă. În momentul în care un client trimite un fișier către server utilizând o cerere HTTP, Multer preia acest fișier și îl stochează în server. Este foarte utilă în cazul încărcării fișierelor de tip imagine într-o aplicație web, și oferă posibilitatea de a încărca fișiere în diverse formate precum JPEG, PNG sau fișiere video de tip mp4. Prin intermediul acestui API se pot specifica specificații precum dimensiunea maximă a fișierului sau numărul maxim de fișiere care pot fi încărcate în același timp. [WWW08]

Exemplu utilizare Multer:



Figură 9 Exemplu utilziare Multer

* + 1. **JWT**

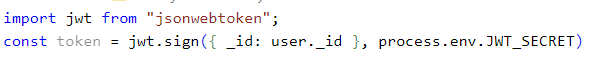
JWT sau JSON Web Token, reprezintă un standard care definește o modalitate compactă de a transmite informații între utilizatori a unui obiect JSON, într-un mod sigur. Această informație poate fi verificată și validată deoarece este definită cu o semnatură digitală, utilizând o pereche de chei secretă/publică utilizând RSA. Prin utilizarea acestei metode de securizare și criptare se asigură autenticitatea și confidențialitatea informațiilor.

Un token JWT este generat pe server în momentul în care un user se autentifică și apoi este transmis clientului, iar apoi token-ul poate fi inclus în fiecare cerere HTTP făcută de către server, astfel acesta validează și autorizează acțiunile clientului.[WWW09]

Un token JSON Web Token este alcătuit din trei părți

1. Header 🡪 Prezintă informații despre tipul token-ului și despre algoritmul utilizat pentru semnarea datelor (SHA256 sau RSA)
2. Payload 🡪 Prezintă informații despre user-ul autentificat precum id-ul acestuia sau rolul lui
3. Semnătura 🡪 Semnătura token-ului folosind cheia secretă

Exemplu utilizare JWT:



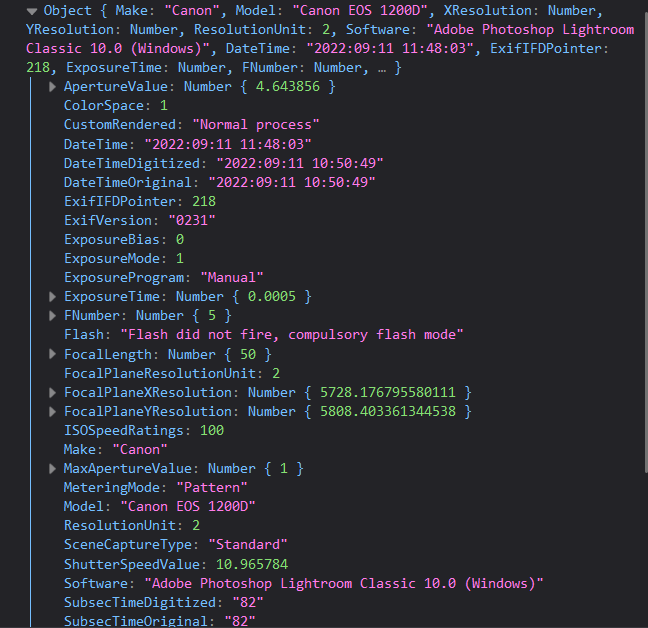
Figură 10 Exemplu utilizare JWT

* + 1. **EXIF**

EXIF-js este o librărie JavaScript care oferă p modalitate prin care se extrage metadata din fișiere de timp imagi într-un browser web. Această librărie permite accesul și manipularea informației de tip EXIF(Exchangeable image file format) care este incorporată în majoritatea fișierelor de tip imagini precum JPEG.[WWW10]

Dacă este folosită într-o aplicație React, Exif-js poate fi folosită pentru a extrage metadata din imaginile care sunt încărcate sau primite de la un API. Metadata-ul primit include informații despre timpul camerei utilizate, data și ora la care s-a realizat fotografia, coordonate GPS, dimensiunile imaginii, spațiul de culori, informații despre cine deține imaginea și multe alte. Aceste informații sunt utile pentru aplicații care realizează sortarea și filtrarea imaginilor, sau afișarea lor bazată pe aceste criterii.

Exemplu METADATA extras folosind EXIF-JS:



Figură 11 Exemplu Metadata extras folosind EXIF-JS

* + 1. **DotEnv**

Dotenv este o librărie JavaScript care permite utilizarea unor variabile dintr-un fișier .env în aplicația Node.js. Aceste variabile au o valoare dinamică și pot fi definite în afara codului aplicației, precum într-un string de conexiune pentru o bază de date sau o cheie a unui API, și ulterior accesate de către aplicație la rulare.

Folosind dotenv, este posibilă stocarea variabilelor confidențiale în acest fișier .env care ulterior vor și încărcate în aplicația Node.js cu o configurație simplă. Prin folosirea acestei modalități, datele sensibile sunt separate de cod și mult mai ușor de utilizat.

Exemplu de utilizare dotenv:



Figură 12 Exemplu utilizare dotenv

Conținut fișier .env:

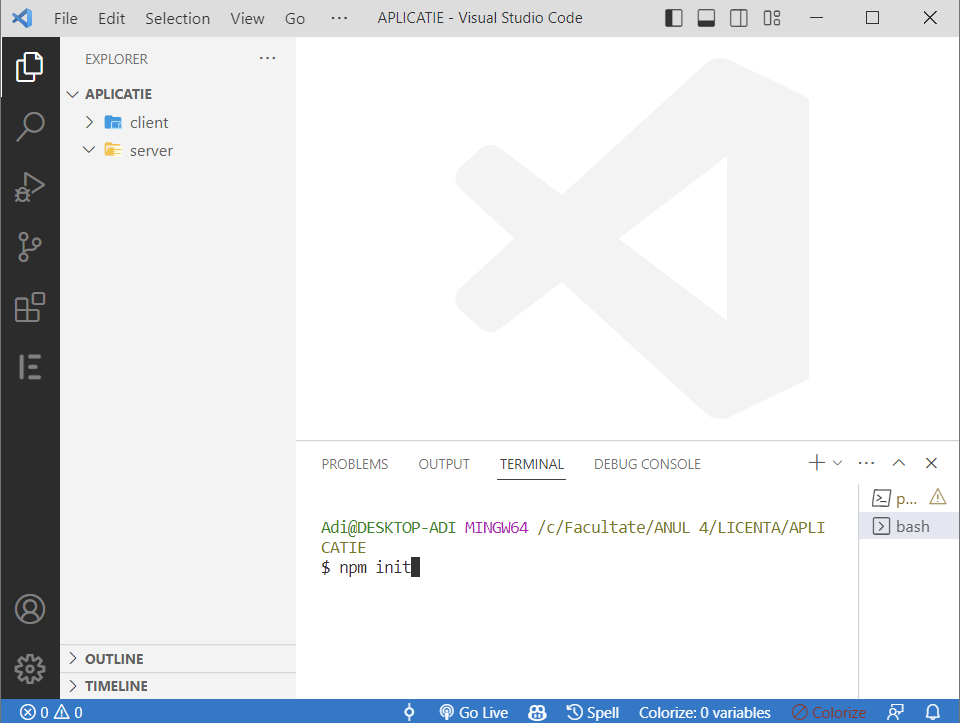


Figură 13 Conținut fișier .env

1. **Realizarea aplicației**
   1. **Mediul de dezvoltare**

Inițierea aplicației este realizată în câțiva pași ușor de urmat, bine definiți și complet necesari pentru acest tip de framework. Pașii care trebuie urmați în vederea realizării acestui proiect sunt:

1. Primul pas în vederea realizării aplicației este creearea unui folder în aplicația Visual Studio Code care va primii numele proiectului. Ulterior, acest folder va conține alte 2 subfoldere, unul denumit client și celălalt server, unde se vor definii fișierele specifice atât pentru partea de client a aplicației cât și pentru partea de server(Figura 3.1)
2. După ce instalarea acestor pachete a luat sfârșit, se inițializează o nouă aplicație Node.js utilizând comanda npm init. Această comandă creează un fișier package.json în interiorul folderului aplicației și îl va popula cu informații precum numele, descrierea și versiunea aplicației și a pachetelor instalate. (Figura 3.1)
3. Următorul pas care trebuie făcut în vederea dezvoltării aplicației instalarea pachetului Node.js de pe internet care vine împreună cu un instrument numit npm , folosit ulterior la instalarea celorlalte pachete necesare pentru o aplicație MERN. Aceste pachete sunt MongoDB,Express.js,React. Pentru instalarea lor se va rula în consola aplicației(ctrl + ~) comanda npm install express mongoose react react-router-dom dotenv[WWW14]. Aceste librării sunt complet necesare pentru rularea și buna funționare a aplicației. (Figura 3.2)



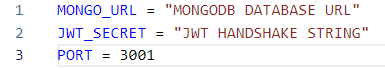
Figură 14 Pas 1. Inițierea mediului de dezvoltare



Figură 15 Pas 2. Instalarea pachetelor necesare

* 1. **Dezvoltarea serverului**
     1. **Configurare fișier .env**

Fișierul .env este alcătuit din câteva variabile de stare utilizate pentru a realiza conexiunea cu baza de date, pentru a realiza handshake-ul între cheia privată și publică a JWT și pentru a stabilii portul local pe care aplicația urmează să fie rulată.

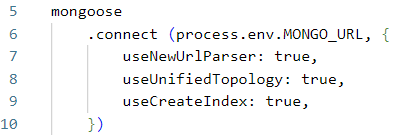


Figură 16 Exemplu complex conținut fișier .env

În codul de mai sus variabila MONGO\_URL este definită ca fiind URL-ul de conexiune între baza de date și aplicație, variabila JWT\_SECRET reprezintă cheia pentru a semna și verifica token-urile JSON Web Token și variabila PORT defineste numărul de port utilizat.

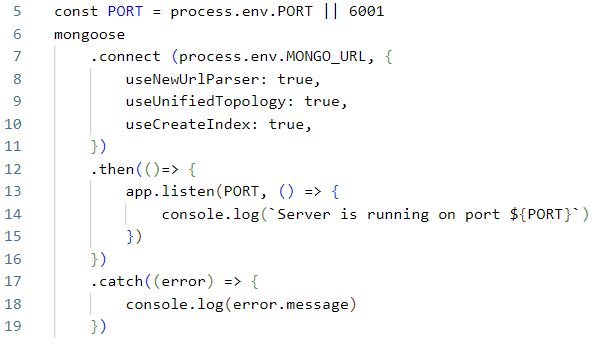
* + 1. **Conectarea la baza de date**

Aplicația necesită o conexiune stabilă cu baza de date MongoDB pentru a asigura o comunicație între cererile serverului și aplicația client. Astfel se utilizează librăria mongoose prin intermediul căreia se realizează conexiunea la baza de date. Acest lucru este realizat utilizând două variabile de mediu extrase din fișierul .env, și anume MONGO\_URL și PORT. Aceste 2 variabile definesc URL-ul unic al bazei de date la care aplicația încearcă să se conecteze și portul pe care aceasta va rula local.



Figură 17 Conectarea la baza de date

După ce conexiunea este validată rulând codul din imaginea de mai sus, serverul va incepe să asculte pe variabila PORT sau pe portul 6001, în cazul în care aceasta nu este definită. În cazul în care conexiunea la baza de date nu poate fi stabilită, se va afișa un mesaj de eroare în consolă.

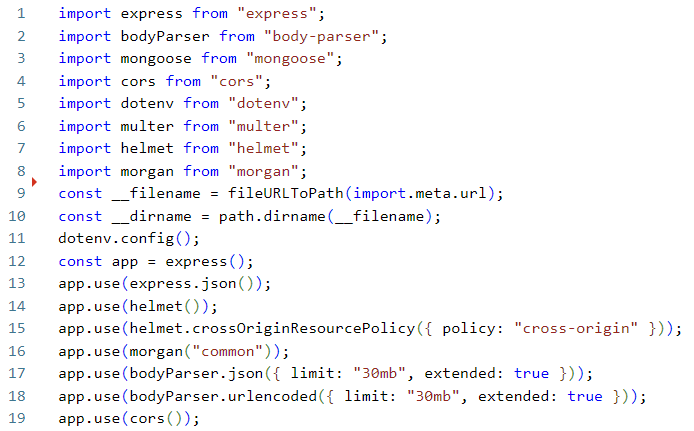


Figură 18 Conectarea la baza de date finală

* + 1. **Configurare server**

Primul pas în cadrul configurării serverului aplicației constă în definirea modulelor necesare pentru creearea serverului și încărcarea pachetelor necesare. Apoi sunt făcute câteva configurări, precum setarea directorului curent, încărcarea fișierului .env care va fi configurat ulterior și care va gestiona variabilele de mediu ale aplicației, și definirea variabilelor de configurație.

Ulterior se definește o instanță a aplicației Express și se configurează modulele morgan,helmet și cors care au ca rol asigurarea securității și menținerea performanței optime a serverului.



Figură 19 Definire Express și configurare module

Metoda helmet.crossOriginResourcePolicy este utilizată pentru a introduce un set de protecții de securitate pentru aplicație, astfel se permite resurselor să fie încărcate de pe alte domenii, dar în același timp controlează modul în care aceste resurse pot fi folosite de browser.

Pachetul bodyParser este folosit pentru decodificarea datelor trimise între client și server prin intermediul cererilor HTTP. În acest mod, aceste pachete pot fi utilizate ușor în aplicație. Astfel, utilizând bodyParser.json și bodyParser.urlencoded se configurează modul în care datele sunt analizate de la client în corpul cererilor HTTP și le face disponibile în obiectul req.body.

Funcția bodyParser.json are ca rol configurarea cererilor astfel acestea, primite de la client în format JSON, vor fi analizate și ulterior transformate în obiecte JavaScript, și setează limita datelor pimite la 30 megabytes. Funcția bodyParser.urlencoded are același rol ca și funcția de mai sus, doar că lucrează cu obiecte urlencoded.

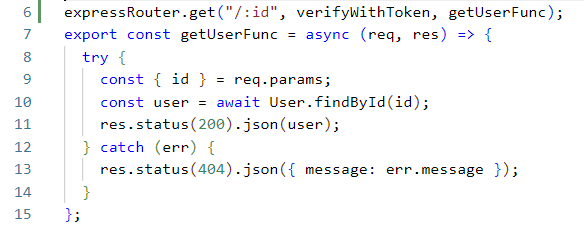
Ulterior, pentru a gestiona fișierele în cadrul aplicației, se configurează middleware-ul multer care are ca scop asigurarea unui mediu ușor de folosit pentru încărcarea fisierelor media.

În plus, pe lângă multer, sunt configurate rutele necesare autentificării și postării fișierelor media în galeria aplicației. Pentru a configura multer, se definesc câțiva parametrii necesari precum tipul opiectul multer.diskStorage pentru a specifica locația de stocare a imaginilor încărcate în aplicație și modul în care acestea ar trebuii să fie numite. Mai exact, se specifică directorul destinație de pe disc, în cazul de față `public/assets`, urmat de numele original al fișierului.



Figură 20 Stocare fișiere și routare

Rutele din codul de mai sus sunt utilizate pentru a gestiona cererile HTTP primite de la client și pentru trimiterea înapoi a răspunsurilor corespunzătoare. Prima rută /auth/register va fi utilizată pentru a permite unui viitor utilizator al aplicației să își creeze un cont în aplicație folosind funția registerFunt care urmează să fie prezentată la capitolul 3.2.5. Următoarea rută declarată se ocupă de creearea unui post în galeria aplicației și funcționează prin intermediul funcției createPostInFeedFunc care va fi prezentată la capitolul 3.2.5. În cadrul rutei de creeare a unei postări, intervine și JWT, verificat prin intermediul funcției verifyWithToken, funcție definită la capitolul 3.2.3. Următoarea rută /auth, se ocupă cu logarea utilizatorilor care au deja un cont creat în aplicație, ruta /users constă într-o funcție .get din cadrul expressRouter unde se apelează funcția getUserFunc prin intermediul căreia se aduc toate datele despre utilizator.



Figură 21 Metoda getUserFunc apelată pe expressRouter

Metoda getUserFunc este utilizată pentru a se obține detalii referitoare la un anumit utilizator al aplicației. În prima linie din cadul metodei se obține parametrul principal definit ca id-ul utilizatorului, urmat de metoda findById(id) care caută în baza de date utilizatorul cu respectivul id.

Această metodă este furnizată de către Mongoose și este utilizată pentru a găsii un document din baza de date bazat pe id-ul acestuia. Utilizând un bloc try-catch, dacă id-ul utilizatorului este valabil și utilizatorul este găsit, se returnează un răspuns HTTP cu codul 200 care reprezintă succes în operația inițiată și de asemenea, returnează obiectul utilizatorului în format JSON.

Dacă id-ul nu este valabil, deci utilizatorul nu a fost găsit, se intră pe ramura catch a blocului try-catch și se returnează un răspuns cu codul de eroare 404 urmat de un obiect JSON care cuprinde mesajul cu eroarea propriu-zisă.

Ulterior, ultima dar poate cea mai importantă rută din cadrul acestei aplicații este ruta /posts, care este alcătuită din 3 tipuri de operații CRUD: get, patch și delete. Aceste operații de tip router asigură citirea, modificarea și stergerea datelor referitoare la postările utilizatorilor în cadrul galeriei.

* + 1. **Configurarea rutelor**



Figură 22 Configurarea routelor

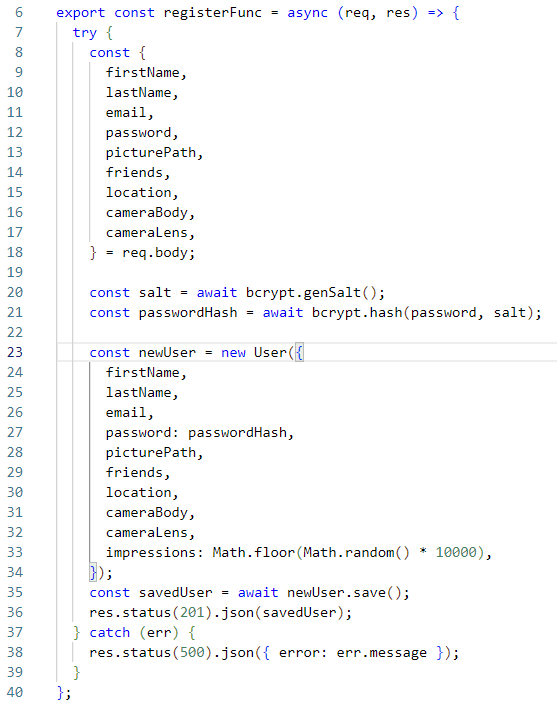
Fiecare operație CRUD prezentată în codul de mai sus este verificată securizat prin intemediul JWT și este realizată apelând funcția definită pentru fiecare tip de operație în parte. Aceste funcții sunt prezentate la capitolul 3.2.5.

* + - 1. **Configurarea metodelor de tip controller**

În dezvoltarea apicației, aceste metode de tip controller sunt utilizate pentru a manipula datele din baza de date, implicit pentru a modifica starea acestora și astfel pentru a defini anumite funcționalități puse la dispozitie utilizatorului. S-au folosit trei fișiere de tip denumite semnificativ, auth.js, posts.js și users.js.

Fiecare fișier conține funcțiile necesare astfel:

* În fișierul auth.js s-au implementat metodele registerFunc() și loginFunc().



Figură 23 Metoda registerFunc

Metoda din codul de mai sus este utilizată pentru înregistrarea unui nou utilizator în aplicație. Utilizând cererea req.body sunt preluate datele utilizatorului, precum numele, prenumele, email, parola și ceilalți parametrii care trebuie completați pentru a putea finaliza înregistrarea acestuia în baza de date, parola utilizatorului fiind criptată utilizând metoda bcrypt.hash.

După ce aceste date au fost completate în formularul de înregistrare și parola a fost criptată, un nou utilizator va fi creat în baza de date, iar obiectul lui JSON va conține toate informațiile preluate. La finalul cererii este returnat un cod de stare și utilizatorul este creat, iar în cazul unei erori, se returnează un mesaj de eroare care ajută utilizatorul să înțeleagă de ce nu a funcționat înregistrarea.



Figură 24 Metoda loginFunc

Metoda loginFunc din codul de mai sus funcționează similar cu metoda utilizată la înregistrare, diferența majoră e că utilizând această metodă se verifică valabilitatea datelor de conectare create de către utilizator, și anume email-ul acestuia și parola. Astfel, utilizând cererea req.body, metoda primește datele email și parolă din baza de date și apoi verifică email-ul prezentat de către utilizator în momentul completării cererii de logare.

Dacă acest user nu există, adică dacă email-ul dat nu este regăsit în baza de date, se returnează mesajul de eroare. În momentul în care email-ul a fost validat ca fiind prezent în baza de date, se începe verificarea valabilității parolei asignate acelui email, folosind funcția bcrypt.compare.

Dacă se returnează un răspuns negativ, utilizatorul va fi întâmpinat de un mesaj de eroare corespunzător, iar în cazul în care parola este validă și autentificarea este cu succes, se generează un token JWT folosind funcția jwt.sing() și se elimină parola din obiectul utilizatorului. Ultimul pas constă într-un mesaj de status care va fi returnat la finalul cererii.

* În fișierul posts.js s-au implementat medodele utilizate pentru gestionarea postărilor în cadrul galeriei.

Principala metodă dezvoltată este createPostInFeedFunc(). Prin intermediul ei utilizatorul poate creea o postare în cadrul aplicației.

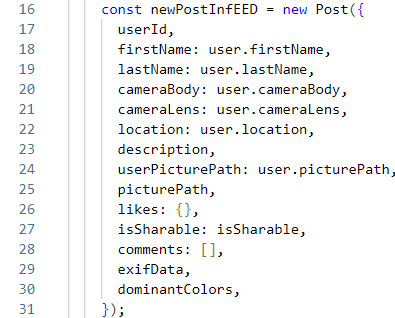


Figură 25 Metoda createPostInFeedFunc

Metoda preia informațiile necesare precum id-ul utilizatorului, descrierea pozei, informațiile referitoare la metadata-ul exif și culorile dominante din imagine, iar apoi utilizează id-ul utilizatorului pentru a găsii informațiile despre acesta pentru a se stabili cine creează această postare în fluxul aplicației.

După ce aceste informații au fost preluate si o nouă postare a fost creeată, metoda returnează toate postările din feed înapoi.

Datele importate sunt definite cu ajutorul modelului Post descris la capitolul 3.2.7, iar modul lui de implementare în această metodă este prezentat mai jos. Pe scurt, se inițiază o nouă instanță a modelului Post cu proprietățile predefinite. Aceste proprietăți sunt extrase din obiectul req.body al metodei.

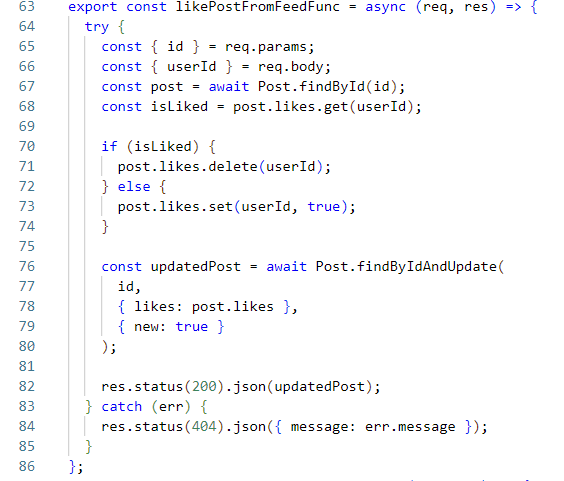


Figură 26 Exemplu apel al modelului Post

Următoarele metode dezvoltate pentru gestionarea postărilor sunt:

* + getPostsFromFeedFunc()
  + getPostsByUserFunc()
  + likePostFromFeedFunc()
  + removePostFromFeedFunc()
  + deletePostFunc().

Aceste metode au o funcționalitate similară, din acest motiv va fi prezentată doar metoda likePostFromFeedFunc().



Figură 27 Metoda likePostFromFeedFunc

Această metodă este utilizată pentru a valida o apreciere asupra unei postări din cadrul fluxului aplicației. Metoda preia datele din req.body precum id-ul postării, id-ul utilizatorului care încearcă să aprecieze postarea, și ulterior caută postarea utilizând id-ul definit. Pasul următor constă într-o verificare a unicității variabilei isLiked.

Această verificare este necesară pentru a prevenii un utilizator să aprecieze aceeași postare de mai multe ori. După validare se inserează în vectorul like al modelului Post id-ul utilizatorului care a apreciat postarea și flag-ul true. La final se actualizează pentru ca modificarea făcută să fie valabilă și se returnează statusul metodei updatedPost.

Celelalte metode înșiruire mai sus au o funcționalitate similară. Astfel metodele „getPostsFromFeedFunc” și „getPostsByUserFunc” sunt utilizate pentru a returna toate postările din baza de date în fluxul galeriei, diferența fiind dată de tipul metodei. Metoda „getPostsFromFeedFunc” returnează toate postările distribuite din galerie în fluxul aplicației, în timp ce cealaltă metodă returnează doar postările utilizatorului logat, acestea fiind returnate în pagina Galerie a aplicației, nu în flux.

Metodele share, remove și delete au ca scop principal:

* + distruibuirea imaginii în flux unde va fi vizibilă tuturor utilizatorilor logați în aplicație
  + eliminarea imaginii din flux pentru a nu mai fi vizibilă tuturor utilizatorilor logați în aplicație
  + respectiv ștergerea completă a imaginii din aplicație și din baza de date
* În fișierul users.js este implementată metoda „getUserFunc”.

Această metodă a fost implenentată cu scopul de a permite utilizatorului să afișeze lista lui de prieteni, dar afișarea propriu-zisă a listei nu s-a realizat în cadrul acestei versiuni ale aplicației.

Metoda are principiul de funcționare similar cu getPostsFromFeedFunc, doar că în loc să returneze postările din baza de date, returnează toți utilizatorii, adică toate id-urile utilziatorilor care au setat flag-ul de prietenie pozitiv.

* + 1. **Configurarea securității cu JWT**

Pentru a asigura o securitate sporită asupra cererilor HTTP, aplicația prezentată folosește ca metoda de verificare tokenul JWT. JSON Web Token este un standard care asigură transmiterea infromațiilor între client și server într-un mod securizat. În cadrul aplicației, JWT este utilizat pentru diferite operații precum validarea unui utilizator autentificat și pentru validarea fiecărei operațiuni pe care un utilizator o poate face asupra unei postări.

Primul pas pentru a putea utilziza token-ul JWT a reprezentat importarea acestei librării în cadrul aplicației, conform imaginii de mai jos.



Figură 28 Inserare JWT

Majoritatea metodelor utilizate în aplicație au ca criteriu middleware de verificare implementat folosind JWT, astfel se creează un nivel de securitate sporit împotriva unei serii de vulnerabilități comune ale autentificării, respectiv autorizării, în cadrul aplicației.



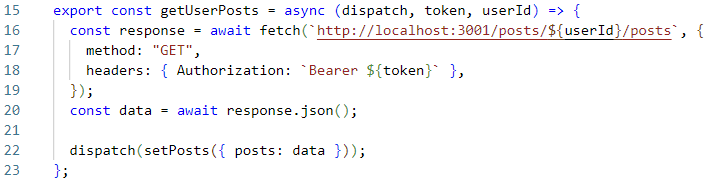
Figură 29 Metoda verifyWithToken

Metoda din codul prezentat mai sus este cea utilizată în cadrul aplicației pentru validarea autentificării utilizatorului în aplicație, folosind JWT. Această metodă extrage token-ul din antetul “Authorization” din cadrul cererii HTTP și apoi verifică utilizând cheia secretă ”process.env.JWT\_SECRET” regăsită în fișierul .env al aplicației.

După ce verificarea a luat sfârșit, dacă token-ul este valid și acesta a trecut de procesul de decodare și verificare, informațiile care doresc a fi extrase despre utilizator sunt stocare în obiectul “req.body”, iar middleware-ul este continuat către funcția următoare prin apelul “next()”

Ultimul pas executat în metoda de mai sus reprezintă validarea răspunsului primit. Acest lucru este soluționat prin returnarea unui mesaj de eroare și un cod de stare corespunzător(403 – acces interzis , 500 – eroare internă de server).

Un exemplu de utilizare a token-ului JWT într-o metodă utilizată în aplicație o reprezintă metoda getUserPosts. În interiorul metodei, JWT este folosit în bodyul cererii GET, unde acesta reprezintă token-ul care este trimis ca header în cadrul cererii pentru a obține postările unui anumit utilizator.



Figură 30 Exemplu utilziare token în metoda getUserPosts

Tokenul este adăugat la header-ul Authorizațion și ulterior este trimit către server împreună cu cererea. Tokenul de autorizare este verificat de către server pentru a se valida permisiunea utilizatorului de a accesa această resursă, iar în cazul în care este validat, serverul va returna postările acestui utilizator.

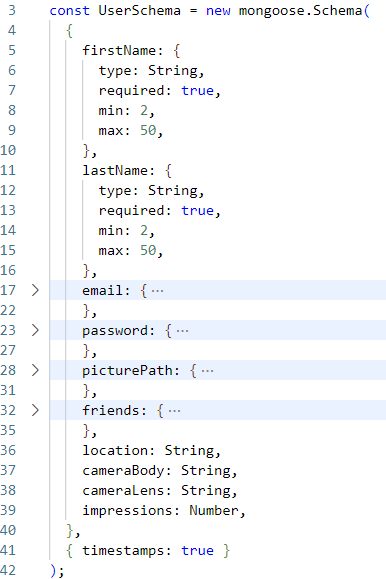
În cazul în care apare o problemă, un mesaj de eroare corespunzător va fi returnat.

* + 1. **Configurarea modelelor**

Schemele mongoose utilizate în aplicație au fost creeate cu ajutorul librăriei “mongoose” și se împart în două mari categorii:

1. Post Schema
2. User Schema

Post Schema:



Figură 31 User schema

Principalul model implementat în aplicație este modelul “UserSchema”. Această schemă, precum și schema “PostSchema” sunt utilizate pentru a definii structura obiectelor care urmează să fie stocate în baza de date MongoDB. În exemplul din *Figură 29*  schema definește un număr de câmpuri și tipurile parametrilor din aceste câmpuri de date care sunt așteptate de către obiectele utilizator stocate în baza de date.

Câmpurile creeate de această schemă sunt:

* firstName
* lastName
* email
* password
* picturePath
* friends
* location
* cameraBody
* CameraLens
* Impressions

Fiecare câmp prezentat mai sus are ca proprietate tipul parametrului utilizat. Astfel unele elemente precum numele și prenumele sunt definite ca obiect de tip string, iar pe lângă tipul variabilei, există încă câțiva parametrii precum minimul și maximul numărului de caractere permis, și variabila “required” care definește parametrul “firstName” ca fiind necesar pentru a putea creea contul utilizatorului.

În schimb, alte elemente precum locația sau tipul apartului foto utilizat sunt definite doar ca parametrii de tip string, deoarece nu este nevoie de alți parametrii care să diferențieze aceste câmpuri. Într-un mod similar este definit și câmpul „friends”, diferența este că tipul de date utilizat va fi un vector.

UserSchema:



Figură 32 Post Schema

Modelul Post este dezvoltat într-un mod foarte similar cu modelul „UserSchema” deoarece și aici se definesc în mod similar un număr de parametrii utilizați prentru a creea structura obiectului Post.

Astfel, acest model cuprinde un număr similar de parametrii, cu câteva completări:

* Description
* userPicturePath
* likes
* comments
* exifData
* dominantColors

Similar cu celălalt model, aici anumite elemente sunt definite simplu, de tip string. Există însă câteva implementări noi precum:

* „exifData”
* „dominantColors”.

Metadata-ul exif reprezintă o librărie descrisă la capitolul 2.3.6, iar parametrul „exifData” reprezintă un vector de obiecte JSON, în cadrul căruia fiecare elemnt va conține un obiect JSON cu un id, populat cu toate datele extrase dintr-o imagine (*Figură 11*).

Câmpul „dominantColors” este definit ca un obiect care este alcătuit din două proprietăți, „Type” și „Default”. „Type” este un vector de obiecte în care fiecare obiect are trei proprietăți:

* „r”
* „g”
* „b”

Fiecare dintre aceste trei proprietăți reprezintă un cod de culoare din intervalul [0,255] al câmpurilor de culori. Aceste date sunt ulterior extrase automat din imagine în momentul în care utilizatorul încarcă imaginea în aplicație folosind o librărie specială. Aceste coduri de culori sunt ulteror utilizate pentru a filtra imaginile folosind un algoritm de diferențiere între culoarea aleasă de utilizator pentru a filtra imaginile cu culorile dominant asemănătoare cu culoarea aleasă, și culorile stocate în obiectul „dominantColors”.

* 1. **Dezvoltarea clientului**
     1. **Configurarea rutelor**

Pentru a putea configura corect aplicația React, este necesar să definim componenta App prezentată în figura de mai jos.



Figură 33 Configurarea rutelor

În metoda prezentată se configurează o componentă React care definește o aplicație web.Componenta are ca scop principal definirea rutelor utilizate în aplicație pentru a asigura o navigare ușor de utilizat de către utilizator. Componenta conține o serie de elemente React precum:

* useSelector
* createTheme
* useMemo
* BrowserRouter
* Routes

Astfel, folosind aceste componente se definesc modurile de navigare între paginile aplicației. Folosim „useSelector” pentu a permite componentelor React să extragă date și să le utilizeze local în componenta, „useMemo” fiind de asemenea utilizată pentru a optimiza performanța de randare a componentelor. Acest lucru este necesar deoarece în general o componentă se reîncarcă de fiecare dată cand orice proprietate este modificaă, inclusiv starea sa locală, astfel „useMemo” permite memorarea acestor valori calculate anterior, și dacă nu există nici-o modificare a proprietăților, se poate utiliza valoarea predefinită în loc să se reîncarce și aceasta, crescând astfel performanța aplicației semnificativ. De exemplu, dacă elementul „mode” nu are parte de modificări, se va returna valoarea deja memorată, fără să fie nevoie de recalcularea obiectului temei de fiecare dată când acea componentă este reîncărcată.

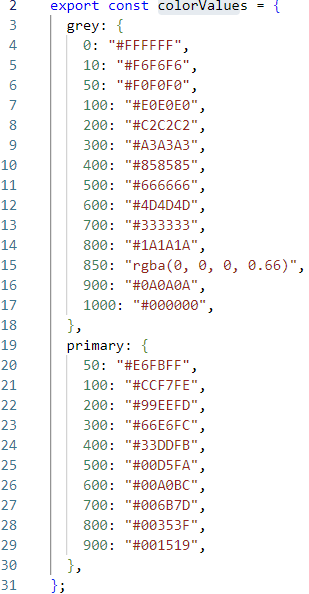
Componenta „createTheme” este folosită în corelație cu metoda „themeSettings” pentru a stabilii tipul de temă afișată, adică temă întunecată sau temă luminoasă. Acest modul este prezentat în următorul capitol.

Componenta „BrowserRoutes” împreună cu „Routes” acceptă instanțe „Route” utilizate pentru desemnarea modurilor de navigare în cadrul aplicației. Astfel se stabilesc rute pentru navigarea între cele trei pagini importante ale aplicației:

* Pagina flux, adică pagina „HomePage”
* Pagina galerie, adică pagina „ProfilePage”
* Pagina de autentificare, adică pagina „LoginPage”
  + 1. **Crearea temelor alb/negru**

În cadrul aplicației s-au definit două tipuri de teme între care utilizatorul poate opta. O temă este cea întunecată, iar cealaltă temă este cea luminoastă. S-a optat la această variantă deoarece majoritatea aplicațiilor folosesc o temă întunecată din motive legate de durata de viață a bateriei și de impactul pe care lumina emisă de ecrane o are asupra ochilor utilizatorului. Deșii tema întunecată oferă beneficii mari, există persoane care preferă o temă luminoasă, care poate părea mai ușor de citit și vizualizat, de aceea există și această variantă.

Cele două teme sunt dezvoltate în jurul unei teme de culori alese pentru a varia între culorile întunecate și luminoase ale aplicației, deoarece daca tema aleasă este cea întunecată, fundalul devine negru, dar textul va trebuii să fie vizibil, deci alb. În același mod se execută și tema luminoasă.



Figură 34 Constante de culori

În figura de mai sus este prezentat modul în care aceste culori sunt definite. Astfel aceste culori reprezintă valori ale obiectului „colorValues” și sunt reprezentate în două variante:

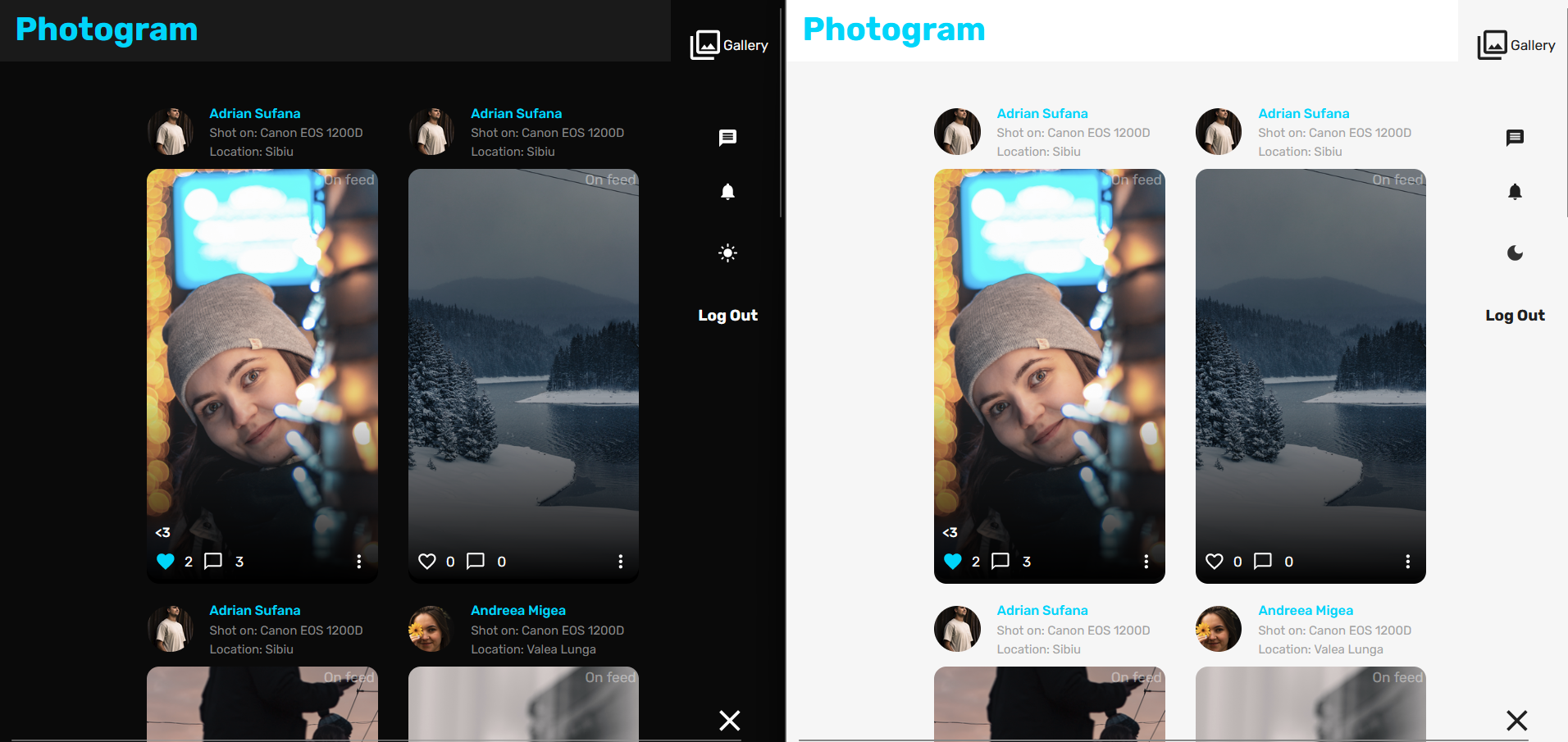
* Grey
* Primary

Pentru fiecare variantă sunt asignate o serie de noanțe de culori, definind o valoare numerică de la 0 la 1000, care reprezintă intensitatea culorii, iar la fiecare intensitate este asignat codul de culoare corespunzător hexazecimal.

Ulterior aceste valori ale intensității culorilor sunt utilizate pentru a definii obiectele metodei „themeSettings”. Astfel se stabilesc două moduri ale paletei de culori, modul „dark” care reprezintă modul întunecat, și ulterior modul luminos. Pe ramura „dark” a acestei metode, se definesc încă trei obiecte care reprezintă tipul culorii, precum „primary”, „neutral”, „background”.

Aceste obiecte sunt utilizate pentru a diferenția tipurile de elemente din cadrul aplicației și pentru o ușoară aplicare a culorilor. În ultimul rând, aceste trei obiecte tocmai prezentate primesc un număr de parametrii, fiecare fiind populat cu o valoare de intensitate definită în paleta de culori prezentată mai sus. Într-un mod similar se creează și obiectele de culori pentru modul luminos, diferența fiind în definirea diferitelor culori și a diferitelor intensități.

Pentru a modifica tipul temei aplicate utilizatorul nu trebuie decât să apese pe butonul asignat acestei funcționalități din bara de navigare a aplicației.



Figură 35 Exemplu vizualizare temă dark vs temă light

Pe scurt, culorile din modul luminos reprezintă inversul culorilor din modul întunecat. Mai jos se prezintă modul de implementare a acestei metode.



Figură 36 Metoda themeSettings pentru modul întunecat

Aceste teme de culori sunt ulterior utilizate pe întreaga structură a aplicației, ele fiind aplicate pe fiecare obiect și text în parte, într-un mod similar cu exemplul următor:

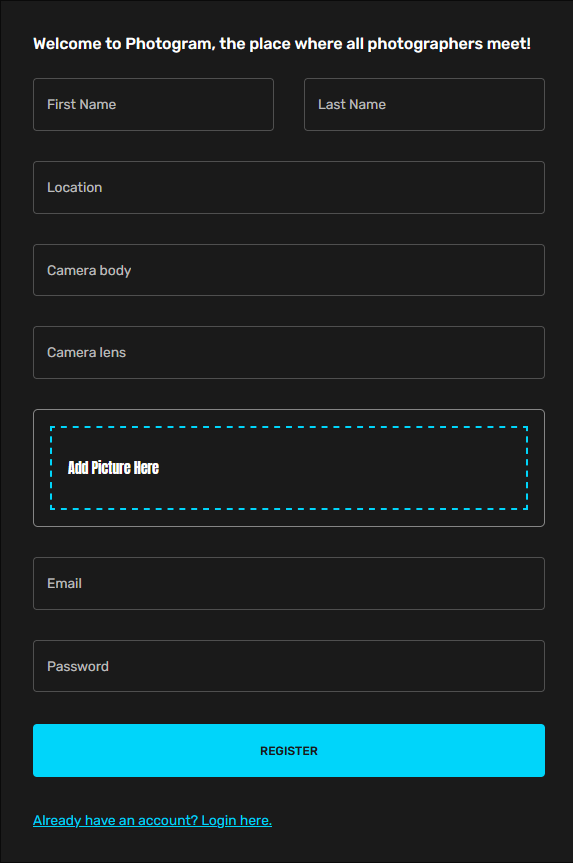
const primaryDarkColor = theme.palette.primary.dark;

Acesastă linie de cod este ulterior utilizată pentru a stabilii culoarea textului „Photogram” din bara de navigare a aplicației. După aplicarea acestei culori în tag-ul de stye al elementului care va afișa textul, logo-ul aplicației va fi afișat în culoarea primary.dark, adică culoarea de intensitate 200, cu codul hexazecimal „#99EEFD”.

* + 1. **Modul înregistrare**

Pentru o funcționare corectă și plăcută a aplicației, este necesar ca un utilizator nou care dorește să utilizeze aplicația să își poată creea un cont pentru ca ulterior să se poată loga în galerie pentru a distribuii mai multe poze. Astfel aplicația dispune de un modul de înregistrare implementat pe baza metodelor de register prezentate la capitolul 3.2.4.1.

Pagina cu care este înâmpinat utilizatorul este pagina „LoginPage”. Aici utilizatorul, poate apăsa pe butonul Log In pentru a naviga la formularul de logare, iar ulterior poate accesa formularul de înregistrare unde își poate creea contul.



Figură 37 Formular înregistrare

Acest formulat este creeat utilizând componente de tip <Box> împreună cu componente de tip <TextField> pentru definirea câmpurilor de input pe care utilizatorul trebuie să le completeze.

{isRegister && (

<>

<TextField

label="First Name"

onBlur={handleBlur}

onChange={handleChange}

value={values.firstName}

name="firstName"

error={

Boolean(touched.firstName) && Boolean(errors.firstName)

}

helperText={touched.firstName && errors.firstName}

sx={{ gridColumn: "span 2" }}

/>

În codul prezentat mai sus se prezintă modul în care câmpul „FirstName” este definit și utilizat. Variabila de stare „isRegister” este utilizată pentru a definii tipul de formular care trebuie afișat pe ecranul utilizatorului. Această variabilă este modificată în momentul în care utilizatorul schimbă modul formularului folosind butonul de tip text din *Figură 35*: „Already have an account? Login here.”. Parametrii aplicați câmpului TextField sunt utilizați pentru a stabilii diferite configurări ale textului afișat și pentru definirea metodei care este execitată în momentul în care câmpul „First Name” este completat. După ce utililizatorul completează toate câmpurile necesare înregistrării și apasă butonul „Register”, se execută metoda „register” care creează un obiect de tip „FormData” utilizat pentru a introduce câmpurile aferente formularului de înregistrare ca un obiect JSON în baza de date. Acest lucru se realizează printr-o cerere aplicată rutei „/auth/register” care apelează metoda aferentă din server pentru a introduce aceste date în colecția din baza de date MongoDB. După ce această metodă s-a executat fără erori, contul utilizatorului a fost creat, iar acesta poate continua cu logarea în propriul cont din aplicație.

const formData = new FormData();

for (let value in values) {

formData.append(value, values[value]);}

formData.append("picturePath", values.picture.name);

const savedUserResponse = await fetch(

"http://localhost:3001/auth/register",

{

method: "POST",

body: formData,

}

);

* + 1. **Modul logare**

După ce utilizatorul a realizat cu succes înregistrarea propriului cont, acesta se poate loga cu email-ul stabilit și parola acestuia în aplicație, utilizând formularul de Log In. După ce câmpurile necesare logării au fost completate și buton de log in a fost apăsat, metoda „login” a componentei react începe validarea credențialelor oferite de către utilizator. Astfel se așteaptă date de la ruta „/auth/login”, iar în momentul în care datele au fost primite, acestea sunt stocare într-o variabilă de stare. Această variabilă de stare este setată în funcție de răspunsul primit de la metoda controller apelată aferentă logării din codul node.js al aplicației.

const loggedIn = await loggedInResponse.json();

onSubmitProps.resetForm();

if (loggedIn) {

dispatch(

setLogin({

user: loggedIn.user,

token: loggedIn.token,

})

);

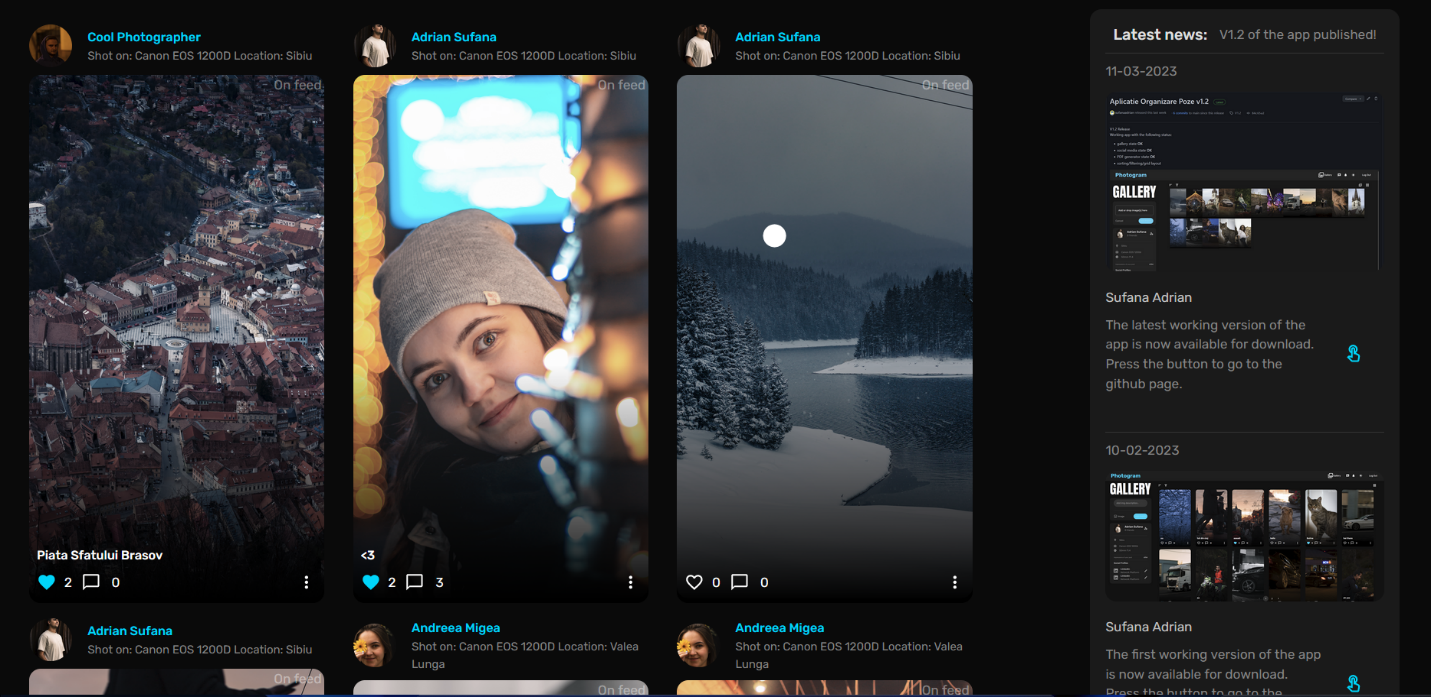
navigate("/home");

}

Dacă variabila are o valoare pozitivă, se utilizează metoda „dispatch” pentru a apela metoda „setLogin” care realizează conectarea folosind userul definit și tokenul de verificare.

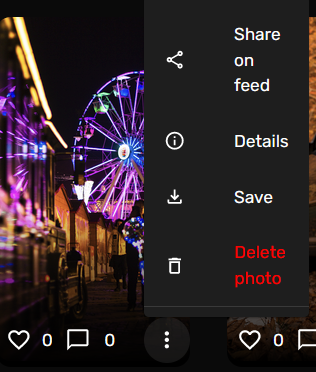
* + 1. **Pagina Social Media**

Aplicația prezentată se împarte în 2 mari pagini. Una dintre acestea este pagina Social Media, unde orice utilizator conectat are posibilitatea să vizualizeze postările distribuite de diferiți utilizatori.



Figură 38 Pagina Social Media

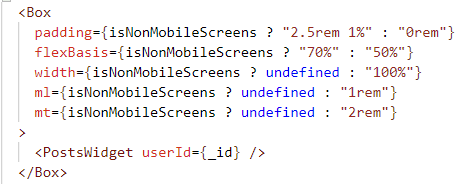
Ideea din spatele paginii se bazează pe nivelul de inspirație la care pot apela ceilalți utilizatori ai aplicației, având la dispoziție creațiile celorlați fotografi din diferite locuri ale lumii. Imaginile vizibile în această pagină sunt doar imaginile care au fost distribuite de către un utilizator din galeria acestuia, iar în termeni tehnici, distribuirea unei imagini se realizează prin setarea flagului „isSharable” pe „true”, utilizând butonul din *figura 37*. Acest parametru de stare este definit în modelul „Post” al aplicației și este actualizat în timp real odată cu apăsarea butonului „Share on feed”, respectiv „Remove from feed”. În același timp, utilizatorii au posibilitatea să vizualizeze metadataul din imagini folosind butonul „Details” *din figura 37*.



Figură 39 Buton "Share on feed"

Prin apăsarea butonului „Details, peste fotografie va fi afișată o listă cu metadataul extras din fiecare imagine în parte, împreună cu cele 3 culori dominante ale imaginii. Astfel, utilizatorii pot studia acești parametrii pentri a-și îmbunătății propriile fotografii. Aceste date sunt pot fi utilizate ulterior în cadrul paginii „Galerie” pentru a filtra imaginile.

Pentru a creea aceste două pagini s-a folosit un principiu simplu, în care se randează postările utilizatorilor în funcție de flagul „isSharable” a fiecărei imagini în parte. Randarea întregii pagini funcționează prin apelul widgetului numit „PostsWidget”, vizibil în figura de mai jos. Acest widget este utilizat pentru a se evita scrierea de cod repetitiv atât pentru pagina Galerie cât și pentru pagina Social Media.

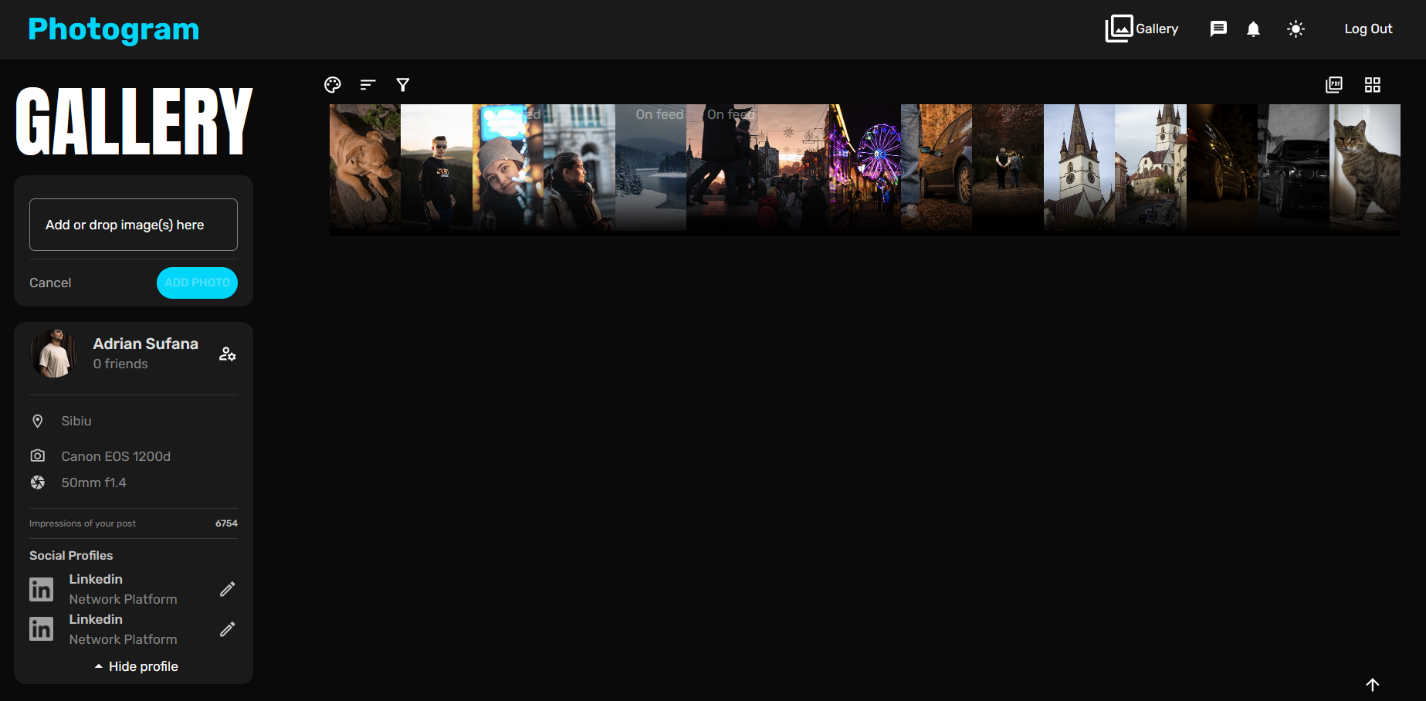


Figură 40 Afișarea postărilor în pagina Social Media

Se utilizează un element „<Box>” în interiorul căruia se apelează widgetul „<PostsWidget>. Variabila „isNonMobileScreens” prezentă în elementul „<Box>” are ca rol definirea modului de vizualizare pentru ecrane mobile sau ecrane standard 16:9. Prezentarea modului de funcționare și implementare a acestui widget este definit la capitolul 3.3.6.

* + 1. **Pagina Galerie**

Pagina „Galerie” reprezintă elementul principal al aplicației prezentate. În cadrul acestei pagini utilizatorul are la dispoziție toate funcționalitățile aplicației, și practic aici acesta își poate încărca imaginile personale fără ca acestea să fie vizibile publicului.

******

Figură 41 Pagina Galerie

În cadrul acestei pagini utilizatorul dispune de un număr de funcționalități care pot fi aplicate asupra imaginilor:

* Modul încarcare imagini
* Meniu sortare imagini
* Meniu filtrare imagini
* Meniu distribuire/ștergere/descărcare/detalii imagine
* Buton generare PDF cu metadataul imaginilor
* Buton schimbare mod vizualizare din grid mic în grid mare

Pagina este dezvoltată într-un mod similar cu pagina „Social Media”, astfel se utilizează același widget apelat și în cealaltă pagină, diferența fiind modul în care anumițe elemente sunt afișate și parametrii cu care acest widget este apelat.

<PostsWidget

userId={userId}

sortCriteria={sortCriteria}

filterCriteria={filterCriteria}

colorCriteria={colorCriteria}

xl={xl}

/>

Widgetul prezentat este apelat diferit, deoarece acesta are nevoie de câțiva parametrii în plus față de pagina social media. Motivul este nevoia de sortare și filtrare a imaginilor în funcție de diferiți parametrii inclusiv metadata cât și culorile dominante. Astfel parametrii „sortCriteria”, „filterCriteria”, ‚colorCriteria” și „xl” vor fi utilizați în cadrul algoritmului de sortare, filtrare și definire a stilului de afișare a imaginilor.

Widgetul „PostsWidget” conține tot algoritmul de sortare și filtrare al imaginilor. Acești algoritmi sunt prezentați în capitolele următoare. Widgetul mapează imaginile în funcție de parametrii stabiliți pe fiecare postare în parte și returnează un obiect de tip „PostWidget” populat cu acești parametrii stabiliți.

<PostWidget

postId={\_id}

postUserId={userId}

name={`${firstName} ${lastName}`}

description={description}

cameraBody={cameraBody}

cameraLens={cameraLens}

location={location}

picturePath={picturePath}

userPicturePath={userPicturePath}

likes={likes}

isSharable={isSharable}

comments={comments}

exifData={exifData}

isLargeGrid={isLargeGrid}

isProfile={isProfile}

dominantColors={dominantColors}

/>

În același timp, în cadrul afișării paginii „Galerie” se apelează și obiectul „SortMenu”, care reprezintă meniul prezent deasupra imaginilor. Acest meniu reprezintă un numpr de cinci butoane:

* Buton „Color Picker”
* Buton sortare
* Buton filtrare
* Buton generare PDF
* Buton schimbare mod vizualizare

Meniul este afișat prin apelare widgetului acestuia în modul următor:

<SortMenu

onSortCriteriaChange={setSortCriteria}

onFilterCriteriaChange={setFilterCriteria}

onColorCriteriaChange={setColorCriteria}

onXLChange={setXl}

/>

Parametrii prezenți sunt utilizați pentru gestionarea metodelor care modifică valorile parametrilor „sortCriteria”, ‚filterCriteria”, „colorCriteria” și „xl”. Acești parametrii sunt apelați în cadrul modulului de apelare al widgetului „PostsWidget’ prezentat mai sus.

Fiecare buton din meniu este definit simplu, utilizând tagul „Popper” prin care se realizează afișarea listei aferente fiecărei funcționalități în parte, într-un mod similar cu codul de mai jos, care afișează primul meniu pentru sortarea imaginilor.

<Paper>

<List>

<ListItem>

<ListItemText style={{ color: palette.primary.dark }}>

Sort by:

</ListItemText>

</ListItem>

<ListItem

onClick={() => setCriteriaClick("dateUp")}

sx={{

"&:hover": {

transition: "all 0.3s",

transform: "scale(1.1) ",

backgroundColor: palette.primary.light,

cursor: "pointer",

},

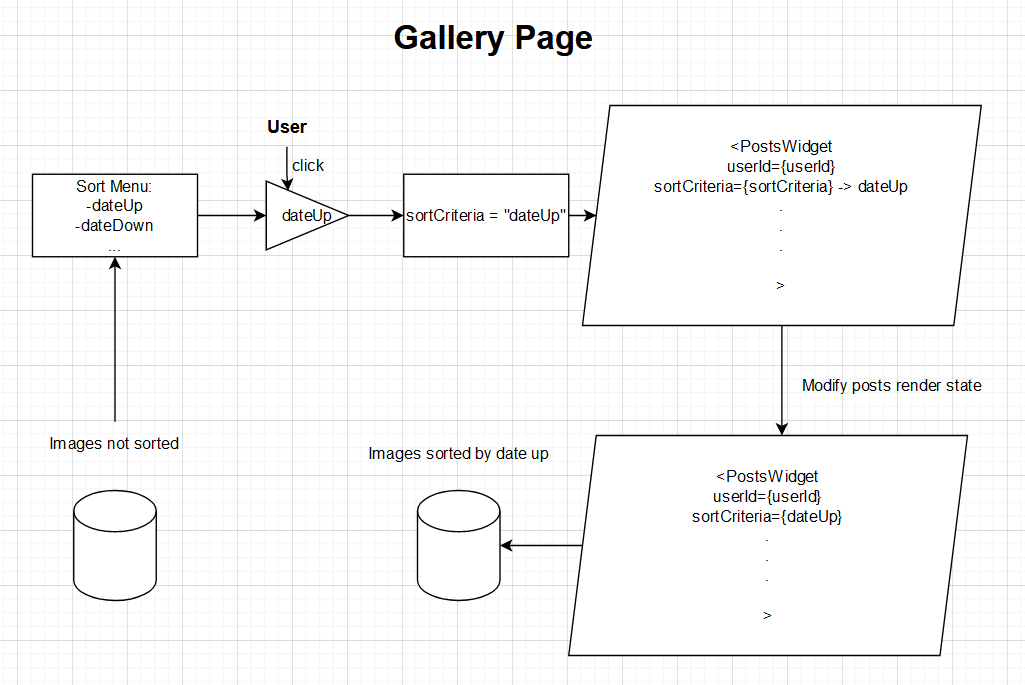
}}

>

Metoda „setCriteriaClick” este utilizată pentru a definii parametrul „sortCriteria”. În exemplul prezentat, prin apăsarea butonului care stabilește sortarea imaginilor după data la care au fost adăugate în mod crescător, se apelează metoda și se setează parametrul egal cu „dateUp”. Acest parametru este ulterior primit în blocul widgetului „PostsWidget”, fiind ulterior utilizat în algoritmul de filtrare al imaginilor prezentat la capitolul 3.3.6.5. Într-un mod foarte similar sunt implementate și celelalte meniuri ale acestei componente, fiecare fiind gestionate de către o metodă specifică pecum metoda „setCriteriaClick”.

Pe scurt, întregul proces de transmitere a parametrilor de filtrare sau sortare pleacă de aici. Componenta „SortMenu” definește valoarea parametrului, de exemplu „dateUp”. Această valoare este ulterior transmisă prin variabila aferentă apelată împreună cu componenta „PostsWidget”, iar apoi imaginile sunt re-afișate în funcție de aceste valori în momentul apelării componentei.

Schema de funcționare a galeriei este una simplă, bazată pe 2 tipuri de widgeturi. Astfel principalul widget este „PostWidget”, care primește parametrii de la componenta „SortMenu”. Apoi este apelat de widgetul „PostsWidget”, care la rândul lui este apelat în cadrul paginii „Galerie”, iar imaginile sunt afișate corespunzător.



Figură 42 Schemă proces sortare imagini

Se utilizează acest stil de dezvoltare deoarece se dorește o implementare ușoară. Astfel „PostWidget” poate fi apelat oriunde, iar în cadrul lui se definesc următoarele elemente:

* Metodele pentru ștergerea, distribuirea, salvarea imaginilor
* Metoda pentru afișarea meniului de sortare, filtrare
* Metoda pentru afișarea imaginii pe ecran complet
* Metoda pentru actualizarea numărului de aprecieri
* Definirea stilului imaginii
* Diferențierea diferitelor funcționalităților bazate pe stadiul flagului „isSharable”
* Afișarea secțiunii de comentarii și a numărului de aprecieri
  + - 1. **Încărcare imagini**

Pentru a gestiona procesul de încărcare a imaginilor în aplicație se utilizează un simplu formular care este alcătuit dintr-un spațiu de selectare a imaginii, un buton de anulare și un buton de adăugare imagini.

În spațiul numit „Add or drop image(s) here” utilizatorul poate selecta imaginile pe care dorește să le încarce în galerie, sau mai simplu, poate să folosească funcția de drop-down, cu butonul de anulare poate întrerupe rapid procesul de încărcare al imaginilor, eliminând astfel toate imaginile selectate în lista de încărcare, iar prin butonul de adăugare se apelează metoda care gestionează adăugarea imaginilor în galerie, deci implicit creearea unei postări definit ca o schemă „PostSchema”.

Utilizatorul are de asemenea posibilitatea să adauge o descriere specifică pentru fiecare imagine în parte, și poate să elimine imaginile una câte una din lista de încărcare. Codul următor prezintă modul în care inputul din spațiul de selectare al imaginii este gestionat:

<Typography

sx={{

"&:hover": {

cursor: "pointer",

transition: "all 0.3s",

transform: "scale(1.1) ",

},

}}

>

Add or drop image(s) here

</Typography>

)}

{!hasImage && <input {...getInputProps()} />}

Odată cu selectarea imaginilor care doresc a fi încărcate în galerie, butonul „Add photo” devine utilizabil. Metoda (*Figura 43*) care este apelată prin apăsarea acestui buton are ca rol gestionarea procesului de creeare a unui nou „formData” care conține toate informațiile noii postări. Acest process se realizează prin creearea unei cererei de tip POST către „/posts” utilizând metoda „fetch()”. Din cauza că utilizatorul poate introduce mai multe imagini în același timp, se utilizează un vector de tip „images”, în care fiecare element va reprezenta o imagine din cele selectate pentru a fi încărcate. Astfel pentru ca fiecare imagine din acest vector să fie încărcată, request-ul de tip POST trebuie efectuat pe fiecare element al vectorului, utilizând o buclă „pentru”, având ca index de parcurgere „images[i], unde i reprezintă index-ul imaginii din lista de imagini adăugate de utilizator.

După ce request-ul este efectuat și se primește un răspuns pozitiv care confirmă ca încărcarea imaginii a fost un succes, se setează câteva variable de stare care sunt utilizate în cadrul gestionării modului de afișare a anumitor elemente din cadrul acestui formular de adăugare imagini:

* setHasImage(!hasImage) 🡪 utilizată pentru a se stabilii starea butonului „Add photo”
* setIsLoading(false) 🡪 utilizată pentru a schimba butonul „Add photo” în „Loading..” până ce toate imaginile au fost încărcate
* setImages([]) 🡪 utilizată pentru a reseta vectorul de imagini după ce imaginile au fost încărcate

În ultimul pas al procesului de încărcare al imaginilor, se apelează din nou metoda „getUserPosts” pentru a aduce în interfața galeriei noile imagini proaspăt încărcate. Metoda realizează un request de tip GET către „/posts/${userId}/posts, și în apelarea ei se utilizează trei parametrii:

* dispatch: metodă utilizată pentru a apela „setPosts()” cu datele postării
* token: token-ul de autorizare pentru a putea realiza handshake-ul
* userId: id-ul utilizatorului care încarcă imaginile



Figură 43 Metoda handlePost()

* + - 1. **Metode gestionare distribuire/aprecieri/elimare imagini**

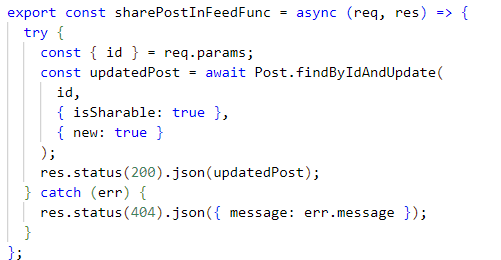
Pentru a oferii o experiență bună între utilizator și aplicație, s-au introdus câteva funcționalități aplicabile pe fiecare imagine în parte:

* Distribuirea imaginii în flux
* Vizualizarea informațiilor metadata și a culorilor dominante
* Descărcarea imaginii
* Ștergerea imaginii
* Aprecierea imaginii

Gestionarea modului de distrubuire a imaginilor în flux s-a realizat simplu, utilizând o varaibilă de stare numită „isSharable”. Această variabilă este stocată ca parametru al imaginii în baza de date și este actualizată în momentul în care utilizatorul apasă butonul „Share on feed” pe imaginea selectată. Astfel se păstrează modul de vizualizare al imaginii chiar dacă aplicația este închisă. Starea acestei variabile este modificată utilizând o metodă de tip patch a librăriei express router.

La creearea request-ului de tip PATCH prin „/:id/share”, se apelează metoda „sha”ePostInFeedFunc" care este prezentată în figura de mai jos. Modul de funcționare al acestei metode este unul simplu, astfel prin intermediul ei se apelează metoda aplicabilă pe baza de date „findByIdAndUpdate” prin intemediul căreia se stabilește id-ul postării unde modificarea va avea loc, și se modifică starea variabilei „isSharable” în „true”. Dacă modificarea a avut succes, se returnează un răspuns pozitiv.

Într-un mod aproape identic se apelează și metoda care se ocupă cu eliminarea imaginii din flux, adică pe scurt, cu schimbarea variabilei „isSharable” din „true” în „false”.



Figură 44 Metoda sharePostInFeedFunc

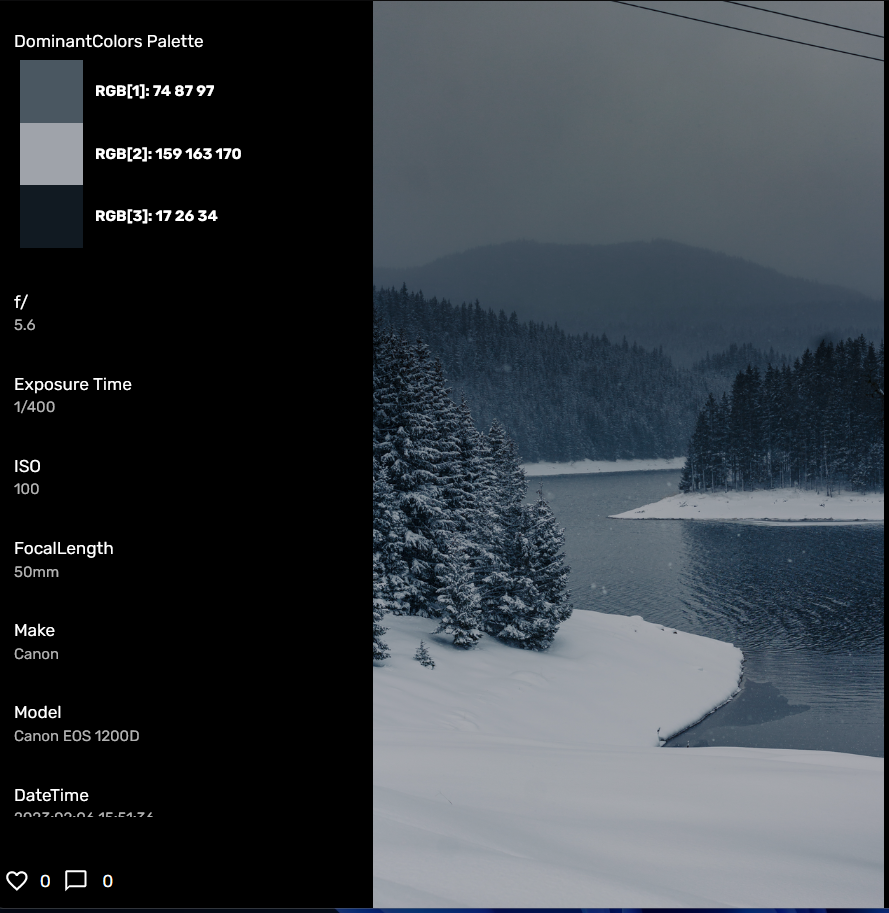
La nivel client metoda apelată odată cu apăsarea butonului „Share on feed” este metoda „patchSharable” care se ocupă cu creearea request-ului de tip fetch către „/posts/id/share”. După ce request-ul s-a finalizat, butonul „Share on feed” devine „Remove from feed’, variabila „isSharable” devine „true”, iar imaginea este prezentă în flux unde poate fi vizualizată, descărcată și apreciată de toți utilizatorii aplicației. Dacă utilizatorul decide că imaginea trebuie să fie eliminată din flux, apasă butonul „Remove from feed”, iar după executarea pașilor prezentați mai sus, imaginea va fi eliminată din flux.

Unul dintre principalele motive pentru care aplicația a fost realizată a fost nevoia de a oferii utilizatorilor un mod prin care să își îmbunătățească tehnicile de fotografie. Astfel implementarea modului de vizualizare a informațiilor de tip metadata și a culorilor dominante din imagine a fost o necesitate. Această listă de informații este ușor vizibilă de către utilizatori atât în flux-ul aplicației, cât și în galerie. Modul în care sunt extrase aceste date este prezentat la capitolul următor.

Aplicația dispune de două moduri de vizualizare grid al imaginilor, unul mic și unul mare. Din cauza acestui lucru, aplicația dispune de două moduri de vizualizare a metadata-ului:

* Mod grid mic 🡪 afișare pe tot ecranul
* Mod grid mare 🡪 afișare deasupra imaginii selectate

Dacă grid-ul utilizat este unul mic, informațiile sunt afișate pe tot ecranul, dând click pe imaginea unde utilizatorul dorește să vizualizeze informațiile. În acest mod, imaginea este deschisă pe tot ecranul, iar în stânga ei este prezentată lista, precum în imaginea următoare.



Figură 45 Mod vizualizare informații cu grid mic

Pentru a stabilii dimensiunea grid-ului este verificată variabila de stare „isLargeGrid”. Această variabilă este modificată în funcție de valoarea variabilei „xl” care poate fi 1 sau 2. Interschimbarea între aceste două valori este realizată prin apăsarea butonului aferent modificării dimensiunii gridului din interfața aplicației. În acest mod, valoarea variabilei „isLargeGrid’ definește două moduri de afișare a listei. Această listă este creată utilizând elementul „List” al librăriei „@mui/material”, iar fiecare element din ea reprezintă un „ListItem”.

{isFullScreen && isLargeGrid && (

<div>

<Typography

style={{

position: "fixed",

bottom: 0,

left: 0,

marginBottom: "2.5rem",

color: "white",

fontSize: "small",

fontWeight: "500",

}}

color={main}

sx={{ ml: "0.5rem" }}

>

{description === "undefined" ? "" : description}

</Typography>

<List......

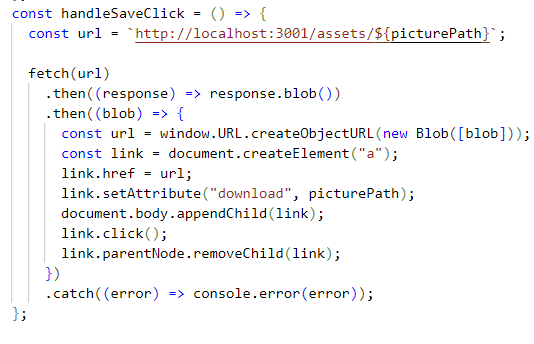
..

..

Codul de mai sus prezintă modul în care este stabilit aspectul listei în funcție de variabila ‚isSharable", și apelarea elementului „List”.

A treia funcționalitate descrisă este modul în care este gestionat apelul butonului „Save” prin intermediul căruia utilizatorul poate salva o imagine din aplicație. Această funcționalitate este implementată deoarece oferă o versatilitate mai mare fotografilor, oferindu-le posibilitatea de a vizualiza imaginile și offline.

Butonul „Save” apelează metoda „handleSaveClick”, ea fiind responsabilă cu gestionarea descărcării imaginii.



Figură 46 Metoda handleSaveClick

În cadrul apelului se utilizează o anumită adresă url care definește imaginea care urmează să fie descărcată, utilizând „picturePath” care reprezintă locația imaginii, implicit denumirea acesteia. Se utilizează o metodă FETCH aplicată pe url-ul tocmai definit, pentru a extrage un obiect „Blob” din răspunsul HTTP. Prin utilizarea acestui obiect se obține conținutul binar primit prin metoda „fetch”, urmată de convertirea răspunsului HTTP într-un obiect „Blob”. După ce obiectul a fost creat, variabila url definește un link temporar care permite desscărcarea imaginii prin atributul „download”. După ce descărcarea a început, acest link temporar este eliminat.

A patra funcționalitate reprezintă posibilitatea utilizatorilor să aprecieze diferite imagini din flux-ul aplicației. Acesta descrie încă un mod de interacțiune între utilizatorii aplicației. Modul de funcționare este unul similar cu cel descris la distribuirea imaginilor folosind variabila „isSharable”, deoarece se utilizează într-un mod similar, parametrul likes din baza de date al schemei postării. Astfel se utilizează prin apelarea metodei „patchLike” care creează un request de tip PATCH către „/posts/id/like”.

Request-ul este întâmpinat de metoda „likePostFromFeedFunc”. Această metodă, într-un mod similar cu metoda care modifica variabila „isSharable”, modifică variabila „isLiked”, incrementându-i valoarea de fiecare dată când un utilizator apreciază postarea.

try {

const { id } = req.params;

const { userId } = req.body;

const post = await Post.findById(id);

const isLiked = post.likes.get(userId);

if (isLiked) {

post.likes.delete(userId);

} else {

post.likes.set(userId, true);

}

const updatedPost = await Post.findByIdAndUpdate(

id,

{ likes: post.likes },

{ new: true }

);

De asemenea, metoda verifică și dacă id-ul utilizatorului se regăsește în obiectul „isLiked”, iar dacă există deja, utilizatorului nu îi este permis să aprecieze postarea din nou,

* + - 1. **Extragerea metadata din imagini folosind exif api**

Cel mai important pas în realizarea aplicației a fost extragerea metadata, automat, la fiecare imagine care este încărcată în aplicație. Acest lucru permite afișarea acestor date în lista prezentată la capitolul precendent.

Datele sunt extrase exact în momentul în care imaginea este procesată pentru încărcare, în cadrul metodei „handlePost”. Librăria utilizată pentru a obține aceste informații se numește „EXIF-JS’.

if (images[i]) {

const exifData = await new Promise((resolve, reject) => {

EXIF.getData(images[i], function () {

resolve(EXIF.getAllTags(this));

});

});

formData.append("exifData", JSON.stringify(exifData));

Librăria creează un request de tip Promise, după ce vectorul cu imagini „images[i]” a fost stabilit. Utilizând metoda „EXIF.getData” apelată pe acest vector de imagini, se extrag toate elementele metadata într-un obiect de tip JSON numit „exifData’. Acest obiect este ulterior descompus și utilizat în afișarea elementelor în lista de informații. Următorul cod reprezintă un exemplu de afișarea a timpului de expunere în lista de informații:

<ListItem>

<ListItemText

primary="Exposure Time"

secondary={"1/" + 1 / exifDataObject.ExposureTime}

/>

</ListItem>

Obiectul „exifData” conține o varietate de date, fiecare fiind ușor accesibilă apelând „exifData.DENUMIRE”.

Aceste informații au fost ulterior implementate și în PDF-ul exportat din aplicație, modul de funcționare și implementare fiind descris la capitolul 3.3.7.

* + - 1. **Extragerea culorilor dominante din imagine**

Culorile dominante din imagini reprezintă următorul pas spre descrierea în detaliu al imaginilor. Aceste date sunt obținute într-un mod foarte similar cu obiectul „exifData”, deoarece se utilizează o librărie, iar obținerea lor se realizează tot în metoda „handlePost”.

const colorArray = await new Promise((resolve, reject) => {

const img = new Image();

img.crossOrigin = "Anonymous";

img.src = URL.createObjectURL(images[i]);

img.onload = () => {

const colorArray = colorThief.getPalette(img, 3);

resolve(colorArray);

};

});

const dominantColors = [

{ r: colorArray[0][0], g: colorArray[0][1], b: colorArray[0][2] },

{ r: colorArray[1][0], g: colorArray[1][1], b: colorArray[1][2] },

{ r: colorArray[2][0], g: colorArray[2][1], b: colorArray[2][2] },

];

Similar cu „EXIF-JS”, librăria „colorThief” funcționează tot pe un request de tip Promise, care creează un url temporar pe fiecare imagine, acesta fiind utilizat ulterior în metoda „onload” apelată pe fiecare imagine. Aici se extrage în obiectul „colorArray” o paletă de culori care conține trei vectori, fiind clasificați după nivelul de dominanță al culorilor extrase.

Acești trei vectori sunt ulterior descompuși în obiectul „dominantColors” pentru a fi introduși în formularul „formData” într-un mod ușor de înțeles și de utilizat. Obiectul „dominantColors” al schemei postării va fi populată în modul următor :

for (let i = 0; i < dominantColors.length; i++) {

const color = dominantColors[i];

formData.append(`dominantColors[${i}][r]`, color.r);

formData.append(`dominantColors[${i}][g]`, color.g);

formData.append(`dominantColors[${i}][b]`, color.b);

}

Aceste valori rgb reprezintă trei culori, roșu, verde și albastru. Fiecare valoare este reprezentată de un număr între 0 și 255, iar combinația acestor trei valori reprezintă culoarea finală. Pasul următor în aplicabilitatea acestor culori este filtrarea folosind intervale de culori bazate pe aceste valori extrase din imagine, procedeu descris la captiolul 3.3.6.6.

* + - 1. **Filtrare utilizând metadata**

În cadrul aplicației sunt disponibile 2 tipuri de filtrarea a imaginilor în cadrul galeriei. Principalul mod de filtrare este reprezentat de filtrarea bazată pe datele disponibile în obiectul „exifData” extras utilizând librăria „EXIF-JS”. Aceste date sunt primite ca parametru în definirea widgetului „PostWidget” în variabila „filterCriteria”.

const PostsWidget = ({

userId,

sortCriteria,

filterCriteria,

colorCriteria,

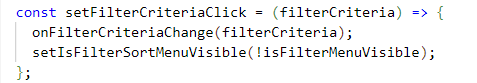
xl,

}) => {.....

Procesul de filtrare începe în cadrul meniului „SortMenu”. Aici se definesc metodele care modifică variabila „filterCriteria”, și valorile care pot fi trimise prin intermediul acestei variabile. În cadrul meniului de filtrare sunt disponibile următoarele tipuri de filtrare:

* Is on feed 🡪 Filtrează imaginile bazat pe variabila „isSharable”
* Location
* ISO
* Aperture
* Exposure
* Flength
* Make
* Model
* Date

Modul prin care aceste valori sunt definire este destul de simplu, varaibila „filterCriteria” fiind modificată folosind metoda următoare:



Figură 47 Metoda setCriteriaClick

Codul următor reprezintă definirea elementului „Is on feed” din cadrul meniului de sortare. Evenimentul „onClick” stabilește metoda apelată în momentul în care acest element este apăsat. În acest caz, metoda apelată este metoda din figura 47, ea stabilind astfel valoarea variabilei „filterCriteria” la textul „isOnFeed”.

Pasul următor spre filtrarea imaginilor după valoarea variabilei „isSharable” constă în gestionarea acestei variabile „filterCriteria” în interiorul widgetului „PostsWidget’. Aici, venită ca parametru „PostsWidget” este utilizată în metoda „filteredAndSortedPosts”. În cadrul acestei metode este gestionat atât modul de filtrare, cât și modul de sortare al imaginilor.

const filteredAndSortedPosts = Array.isArray(posts)

? [...posts]

.map((post) => ({

...post,

exifDataObject: JSON.parse(post.exifData),

}))

.sort((a, b) => {

switch (sortCriteria) {

case "dateUp":

return a.createdAt > b.createdAt ? -1 : 1;

case "dateDown":

return a.createdAt < b.createdAt ? -1 : 1;

case "likes":

const aLikesCount = Object.keys(a.likes).length;

const bLikesCount = Object.keys(b.likes).length;

return bLikesCount - aLikesCount;

default:

return a.createdAt > b.createdAt ? -1 : 1;

}

})

.filter((post) => {

if (!filterCriteria) return post;

if (filterCriteria === "showAll") return post;

if (filterCriteria === "isOnFeed") return post.isSharable;

if (filterCriteria.startsWith("f/")) {

const [, value] = filterCriteria.split("/");

return post.exifDataObject.FNumber === Number(value.trim());

}

Codul prezentat descrie pe scurt modul în care este gestionată variabila „filterCriteria”, care în exemplul curent va fi populată cu stringul „isOnFeed”. Astfel metoda mapează inițial toate postările din aplicație, extrage obiectul exif ca obiect JSON, și apoi verifică valoarea variabilei „filterCriteria”. În funcție de valoarea acestuia, este apelat codul de pe ramurile „if” ale metodei.

În exemplul curent, deoarece variabila „filterCriteria” este „isOnFeed”, ramura care verifică această valoare va fi apelată, iar metoda va returna „post.isSharable”. În acest mod vor fi afișate doar imaginile care au parametrul „isSharanle” egal cu „true”. Într-un mod aproape identic, se filtrează și după ceilalți parametrii disponibili în meniul de filtrare, diferența fiind stringul primit ca parametru în variabila „filtrerCriteria”.

Complexitatea crește puțin la opțiunile de filtrare care permit use-ului să aleagă valoarea cu care se va filtra. De exemplu, în cadrul valorii timpului de expunere, variabila „filterCriteria” va fi trimisă ca parametru cu valoarea din linia de cod prezentată mai jos, unde „1 / exposureInput” reprezintă valoarea trimisă de utilizator ca input.

onClick={() => setFilterCriteriaClick("1/" + 1 / exposureInput)}

Ulterior această valoarea trebuie descompusă în obiectul „PostsWidget” astfel:

if (filterCriteria.startsWith("1/")) {

const [, value] = filterCriteria.split("/");

return post.exifDataObject.ExposureTime === Number(value.trim());

}

* + - 1. **Filtrare utilizând culorile dominante**

Pentru o experiență mai versatilă, culorile dominante extrase din imagine sunt utilizate în cadrul aplicației pentru a filtra postările în funcție de un interval stabilit, bazat pe datele care sunt regăsite în obiectul „dominantColors” al fiecărei imagini. Modul de funcționare al acestei filtrări este similar cu cel descris la capitolul precedent. Procesul de filtrare după culorile dominante pornește de la meniul „SortMenu”, unde este implementat butonul "ColorPicker”. Acest buton, prin apăsarea lui de către utilizator, deschide o mică fereastră care permite utilizatorului să aleagă o culoare.

{isColorPickerVisible && (

<div style={{ backgroundColor: selectedColor, borderRadius: "3%" }}>

<SketchPicker

color={selectedColor}

onChange={(color) => setSelectedColor(color.hex)}

/>

<div style={{ display: "flex", justifyContent: "center" }}>

<Button

style={{

borderRadius: "3%",

color: "white",

}}

size="small"

onClick={() => setColorPickerValue(selectedColor)}

>

Filter

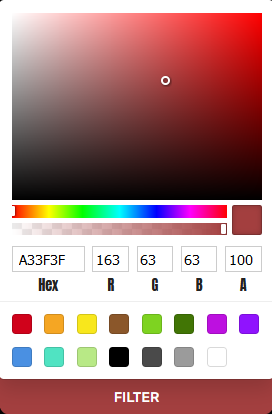
</Button>

</div>

</div>

)}

În codul prezentat mai sus, este utilizat elemntul „SketchPicker” al librăriei „react-color”. El afișează acest panou prin care utilizatorul poate să aleagă o culoare la alegere.



Figură 48 Fereastră color-picker

După ce utilizatorul a ales o culoare, el poate apăsa butonul „FILTER”. În momentul în care acest buton este apăsat, metoda "setSelectedColor” prezentă în codul prezentat anterior este apelată, și astfel se setează variabila „selectedColor” cu o valoare de culoare. În exemplul prezentat, varaibila „selectedColor’ va avea valorile: [163,63,63]. Pasul următor în filtrarea în funcție de această culoare constă în trimiterea acestei variabile ca parametru în obiectul „PostsWidger” prin varibila „colorCriteria”, care este primită ca parametru în corpul obiectului. Ulterior începe procesul propriu-zis de filtrare:

if (colorCriteria !== "all") {

const dominantColors = post.dominantColors;

let colorInRange = false;

for (let i = 0; i < dominantColors.length; i++) {

const color = `${dominantColors[i].r},${dominantColors[i].g},${dominantColors[i].b}`;

colorInRange = isColorInRange(color, colorRangeWithThreshold);

if (colorInRange) {

break; // Exit the loop if a color is found within range

}

}

if (colorInRange) {

return post;

} else {

return null;

}

} else {

return post;

}

Culorile dominante sunt inițial extrare în variabila „dominantColors”. Pentru fiecare culoare din cele trei se parcurge un algoritm de selecție. Culorile sunt trimise ca parametru în metoda prin care se verifică dacă culoarea selectată de utilizator aparține intervalului stabilit prin algoritmul de definire a acestui interval de culori.

const colorRangeWithThreshold = colorRange

? [

Math.max(colorRange[0] - colorRange[0] \* 0.3, 0),

Math.min(colorRange[0] + colorRange[0] \* 0.3, 255),

Math.max(colorRange[1] - colorRange[1] \* 0.3, 0),

Math.min(colorRange[1] + colorRange[1] \* 0.3, 255),

Math.max(colorRange[2] - colorRange[2] \* 0.3, 0),

Math.min(colorRange[2] + colorRange[2] \* 0.3, 255),

]

: null;

Metoda de mai sus definește modul în care intervalul de verificare este creat. Astfel, pentru fiecare culoare în parte dintre cele trei dominante, se calculează un interval bazat pe un procentaj stabilit de 30%. De exemplu dacă culoarea cea mai dominantă este [90, 60, 120], intervalul de verificare pentru această culoare va fi [[63,117], [42,78], [84,156]]. Același proces se repetă și pentru celelalte două culori dominate extrase. În următorul pas, aceste valori creeate în metoda de mai sus, sunt trimise ca parametru în metoda finală de verificare „isColorRange”, împreună cu culoarea venită ca parametru în variabila „colorCriteria”. În această metodă se face o simplă comparație între intervalele create și culoarea din variabila „colorCriteria’. De exemplu dacă culoarea aleasă de utilizator este [163,63,63], verificare va fi următoarea:

* [63<163>117]
* [42<63<78]
* [84>63<156]

Din cauza primei și celei de a treia verificări, valoarea culorii alese de utilizator nu respectă intervalul stabilit, deci imaginea curentă nu va fi afișată în momentul filtrării.

* + - 1. **Sortare**
      2. **Creearea modurilor de vizualizare grid a galeriei**
    1. **Modul rapoarte**

1. **Concluzii și dezvoltări ulterioare**
   1. **Concluzii**
   2. **Dezvoltări ulterioare**
2. **Bibliografie**

[CRE15] Cretulescu, R., Morariu, D., Dezvoltarea aplicatiilor WEB, Editura Universitatii “Lucian Blaga” Sibiu, 2015

[HAV18] Haverbeke Marijn, Eloquent JavaScript 3rd edition, 2018

[MOR21] Morgan Joe, How To Code in React.js, DigitalOcean, New York City, USA 2021

* 1. **Webliografie**
* [WWW01] <https://en.wikipedia.org/wiki/Visual_Studio_Code> - ultima accesare ianuarie 2023
* [WWW02] <https://www.mongodb.com/mern-stack> - ultima accesare ianuarie 2023
* [WWW03] <https://www.mongodb.com/docs/> - ultima accesare ianuarie 2023
* [WWW04] <https://ieeexplore.ieee.org/abstract/document/5617064> - ultima accesare ianuarie 2023
* [WWW05] <https://www.w3schools.com/react/react_router.asp>
* [WWW06] <https://legacy.reactjs.org/docs/hooks-intro.html>
* [WWW07] <https://mongoosejs.com/docs/api/model.html>
* [WWW08] <https://blog.logrocket.com/multer-nodejs-express-upload-file/>
* [WWW09] <https://www.rfc-editor.org/rfc/rfc7519>
* [WWW10] <https://www.deviantart.com/>
* [WWW11] <https://www.flickr.com/>
* [WWW12] <https://www.instagram.com/>
* [WWW13] <https://www.deviantart.com/>
* [WWW14][https://blog.devgenius.io/how-to-get-started-with-the-mern-stack](https://blog.devgenius.io/how-to-get-started-with-the-mern-stack-the-easy-way-b9758fe45956)
* [FIGURA2.1]<https://www.mongodb.com/mern-stack>
* [FIGURA2.2]<https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/>