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](#_Toc132981471)

[**1.** **Prezentare** 6](#_Toc132981472)

[**1.1** **Noțiuni introductive** 6](#_Toc132981473)

[**1.2** **Motivația alegerii temei** 6](#_Toc132981474)

[**1.3** **Scop și obiective** 7](#_Toc132981475)

[**1.4** **Realizări similare acestei teme** 8](#_Toc132981476)

[**2** **Teorie și tehnologii utilizate** 8](#_Toc132981477)

[**2.1** **Visual Studio Code** 8](#_Toc132981478)

[**2.2** **Arhitectura MERN** 9](#_Toc132981479)

[**2.2.1** **MongoDB** 10](#_Toc132981480)

[**2.2.2** **Express.js** 11](#_Toc132981481)

[**2.2.3** **React** 11](#_Toc132981482)

[**2.2.4** **Node.js** 11](#_Toc132981483)

[**2.3** **API-uri / Librării** 12](#_Toc132981484)

[**2.3.1** **React Router și Express API** 12](#_Toc132981485)

[**2.3.2** **Hooks** 14](#_Toc132981486)

[**2.3.3** **Mongoose** 14](#_Toc132981487)

[**2.3.4** **Multer** 15](#_Toc132981488)

[**2.3.5** **JWT** 16](#_Toc132981489)

[**2.3.6** **EXIF** 17](#_Toc132981490)

[**2.3.7** **DotEnv** 17](#_Toc132981491)

[**2.3.8** **Adauga restu api aici daca mai sunt** 18](#_Toc132981492)

[**3** **Realizarea aplicației** 18](#_Toc132981493)

[**3.1** **Mediul de dezvoltare** 18](#_Toc132981494)

[**3.2** **Dezvoltarea propriu-zisă a aplicației** 20](#_Toc132981495)

[**3.2.1** **Configurare fișier .env** 20](#_Toc132981496)

[**3.2.2** **Conectarea la baza de date** 20](#_Toc132981497)

[**3.2.3** **Configurare server** 21](#_Toc132981498)

[**3.2.4** **Configurarea rutelor** 23](#_Toc132981499)

[**3.2.5** **Configurarea modelelor** 24](#_Toc132981500)

[**3.2.6** **Configurarea securității cu JWT** 24](#_Toc132981501)

[**3.2.7** **Configurarea funcțiilor de tip controller** 24](#_Toc132981502)

[**3.2.8** **Register/Login** 24](#_Toc132981503)

[**3.2.9** **Pagina Social Media** 24](#_Toc132981504)

[**3.2.10** **Pagina Galerie** 24](#_Toc132981505)

[**3.2.10.1** **Încărcare imagini** 24](#_Toc132981506)

[**3.2.10.2** **Extragerea metadata din imagini folosind exif api** 24](#_Toc132981507)

[**3.2.10.3** **Filtrare utilizând metadata** 24](#_Toc132981508)

[**3.2.10.4** **Sortare** 24](#_Toc132981509)

[**3.2.10.5** **Creearea modurilor de vizualizare grid a galeriei** 24](#_Toc132981510)

[**3.2.11** **Funcții aplicabile pe imaginea postată** 25](#_Toc132981511)

[**3.2.12** **Modul rapoarte** 25](#_Toc132981512)

[**4** **Concluzii și dezvoltări ulterioare** 25](#_Toc132981513)

[**4.1** **Concluzii** 25](#_Toc132981514)

[**4.2** **Dezvoltări ulterioare** 25](#_Toc132981515)

[**5** **Bibliografie** 25](#_Toc132981516)

[**5.1** **Webliografie** 25](#_Toc132981517)

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[1]
* Instagram[2]
* DevianArt[3]

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.

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.

Diagramă MERN Stack



Figura 2.1 – 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.

* + 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. Fiint folosit împreună cu alte api-uri precum React Router, această bibliotecă ajută la o dezvoltare fluidă și ușoară din punct de vedere tehnic al aplicațiilor web.

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

import { Box } from "@mui/material";

import { React } from "react";

const ProfilePage = () => {

return (

<Box>

</Box>

);

};

export default ProfilePage;

* + 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.

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.

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:

function App() {

  return (

    <div *className*="app">

      <BrowserRouter>

          <Routes>

            <Route

*path*="/"

*element*={<LoginPage />}

/>

            <Route

*path*="/home"

*element*={<HomePage />}

            />

            <Route

*path*="/profile/:userId"

*element*={<ProfilePage />}

            />

          </Routes>

      </BrowserRouter>

    </div>

  );

}

export default App;

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

import express from "express";

import { loginFunc } from "../controllers/auth.js";

const router = express.Router();

router.post("/login", loginFunc);

export default router;

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

 const login = async (*values*, *onSubmitProps*) => {

    const loggedInResponse = await fetch("http://localhost:3001/auth/login", {

      method: "POST",

      headers: { "Content-Type": "application/json" },

      body: JSON.stringify(*values*),

    });

    const loggedIn = await loggedInResponse.json();

*onSubmitProps*.resetForm();

    if (loggedIn) {

      navigate("/home");

      );

    }

  };

* + 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.

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:

import { useState } from "react";

const [checkSomething, setcheckSomething] = useState(0);

Exemplu utilizare useEffect:

import { useEffect} from "react";

useEffect(() => {

    if (checkSomething))) {

     setcheckSomething(false);

    } else {

    }

  }, [checkSomething]);

* + 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:

* 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 |

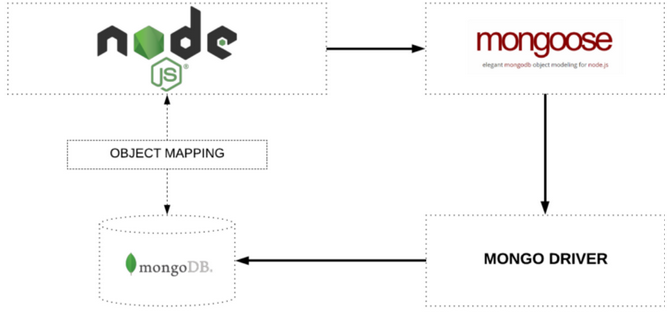


Figura 2.2 – Maparea obiectelor între Node și MongoDB prin intermediul Mongoose

* + 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.

Exemplu utilizare Multer:

*/\* FILE STORAGE \*/*

import multer from "multer";

const storage = multer.diskStorage({

  destination: function (*req*, *file*, *cb*) {

    cb(null, "UploadsFolder/");

  },

  filename: function (*req*, *file*, *cb*) {

    cb(null, *file*.originalname);

  },

});

const upload = multer({ storage });

* + 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.

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:

import jwt from "jsonwebtoken";

const token = jwt.sign({ id: user.\_id }, process.env.JWT\_SECRET);

* + 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.

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:

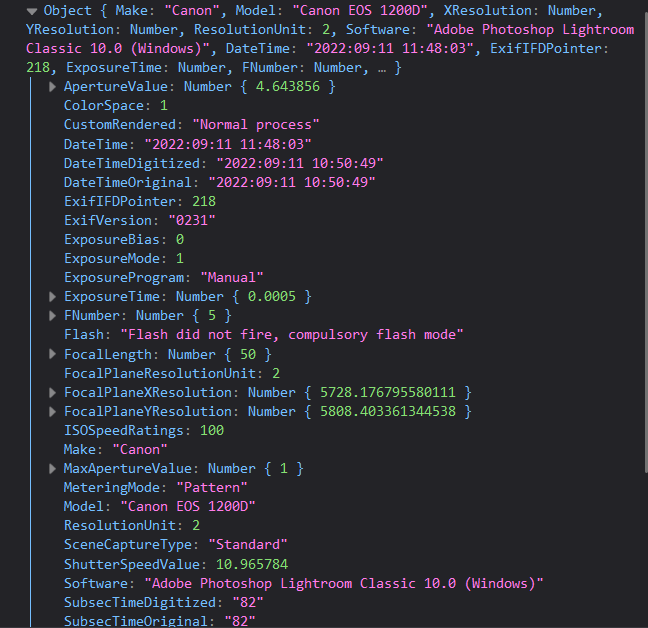


Figura 2.3 – Exemplu Metadata extras folosind EXIF-JS

* + 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:

import dotenv from "dotenv";

const PORT = process.env.PORT;

Conținut fișier .env:

PORT = 3001

* + 1. **Adauga restu api aici daca mai sunt**

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[1]. Aceste librării sunt complet necesare pentru rularea și buna funționare a aplicației. (Figura 3.2)

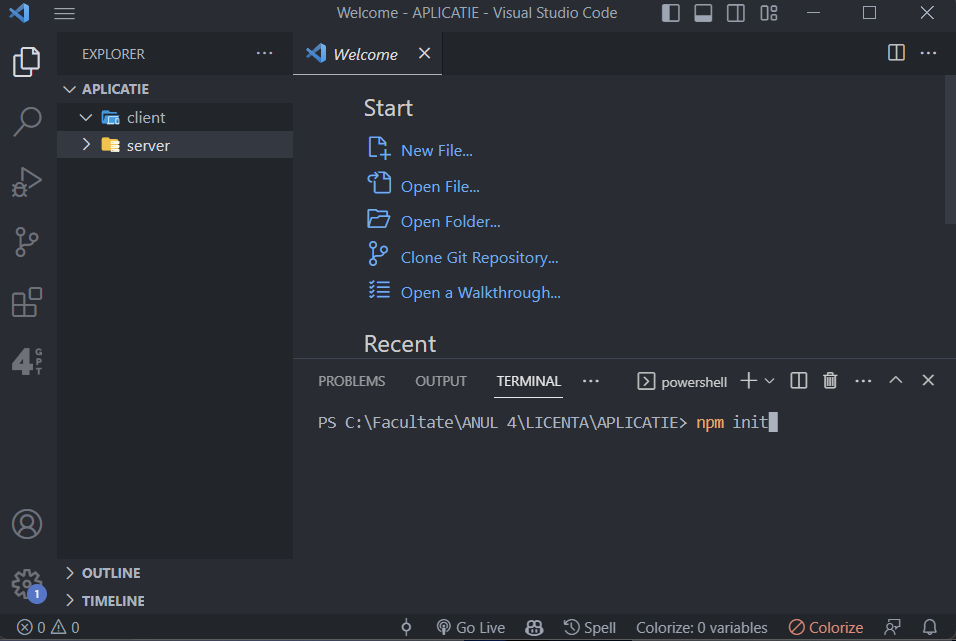


Figura 3.1 – Pas 1. Inițierea mediului de dezvoltare

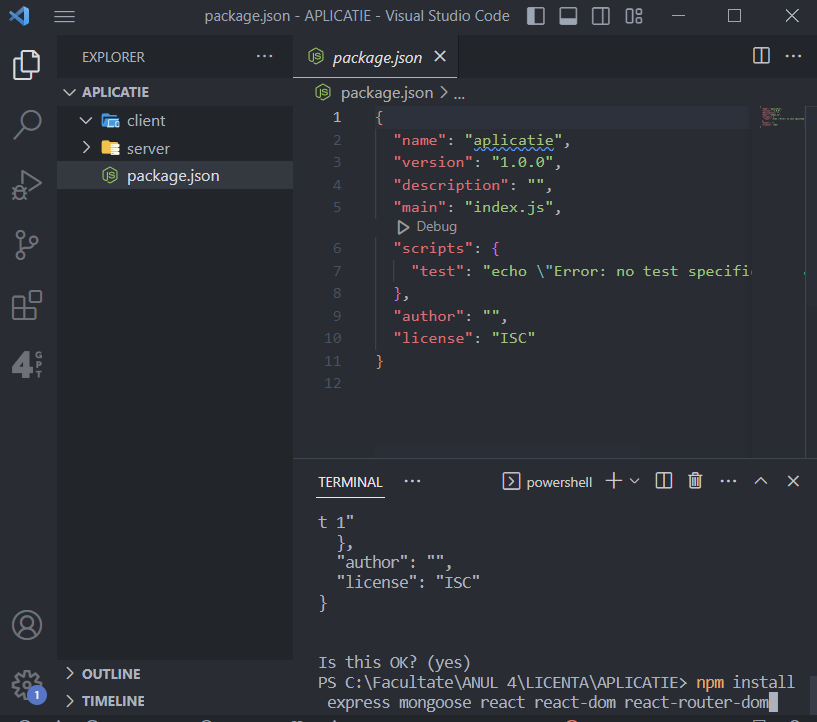


Figura 3.1 – Pas 2. Instalarea pachetelor necesare

* 1. **Dezvoltarea propriu-zisă a aplicației**
     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.

MONGO\_URL = `MONGODB DATABASE URL`

JWT\_SECRET = `JWT HANDSHAKE STRING`

PORT = 3001

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**
    2. **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.

import express from "express";

import bodyParser from "body-parser";

import mongoose from "mongoose";

import cors from "cors";

import dotenv from "dotenv";

import multer from "multer";

import helmet from "helmet";

import morgan from "morgan";

const \_\_filename = fileURLToPath(import.meta.url);

const \_\_dirname = path.dirname(\_\_filename);

dotenv.config();

const app = express();

app.use(express.json());

app.use(helmet());

app.use(helmet.crossOriginResourcePolicy({ policy: "cross-origin" }));

app.use(morgan("common"));

app.use(bodyParser.json({ limit: "30mb", extended: true }));

app.use(bodyParser.urlencoded({ limit: "30mb", extended: true }));

app.use(cors());

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.

*/\* FILE STORAGE \*/*

const storage = multer.diskStorage({

  destination: function (*req*, *file*, *cb*) {

    cb(null, "public/assets");

  },

  filename: function (*req*, *file*, *cb*) {

    cb(null, *file*.originalname);

  },

});

const upload = multer({ storage });

*/\* ROUTES WITH FILES \*/*

app.post("/auth/register", upload.single("picture"), registerFunc);

app.post(

  "/posts",

  verifyWithToken,

  upload.single("picture"),

  createPostInFeedFunc

);

*/\* ROUTES \*/*

app.use("/auth", authRoutes);

app.use("/users", userRoutes);

app.use("/posts", postRoutes);

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.

expressRouter.get("/:id", verifyWithToken, getUserFunc);

*/\* get the user exp func\*/*

export const getUserFunc = async (*req*, *res*) => {

  try {

    const { id } = *req*.params;

    const user = await User.findById(id);

*res*.status(200).json(user);

  } catch (err) {

*res*.status(404).json({ message: err.message });

  }

};

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**

*/\* READ \*/*

router.get("/", verifyWithToken, getPostsFromFeedFunc);

router.get("/:userId/posts", verifyWithToken, getPostsByUserFunc);

*/\* UPDATE \*/*

router.patch("/:id/like", verifyWithToken, likePostFromFeedFunc);

router.patch("/:id/share", verifyWithToken, sharePostInFeedFunc);

router.patch("/:id/removeShare", verifyWithToken, removePostFromFeedFunc);

*/\* DELETE \*/*

router.delete("/:id/deletePost", verifyWithToken, deletePostFunc);

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 modelelor**
    2. **Configurarea securității cu JWT**
    3. **Configurarea funcțiilor de tip controller**
    4. **Register/Login**
    5. **Pagina Social Media**
    6. **Pagina Galerie**

* + - 1. **Încărcare imagini**
      2. **Extragerea metadata din imagini folosind exif api**
      3. **Filtrare utilizând metadata**
      4. **Sortare**
      5. **Creearea modurilor de vizualizare grid a galeriei**
    1. **Funcții aplicabile pe imaginea postată**
    2. **Modul rapoarte**

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

* [WWW01]<https://www.flickr.com/>
* [WWW02]<https://www.instagram.com/>
* [WWW03]<https://www.deviantart.com/>
* [WWW04]<https://datatracker.ietf.org/doc/html/rfc7519>

* [FIGURA2.1]<https://www.mongodb.com/mern-stack>
* [FIGURA2.2]<https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/>
* [1][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)