**Departamentul .....................................................**

**Programul de studii: Calculatoare**

***IANCU George-Alexandru***

PROIECT DE DIPLOMĂ

**Conducător științific:**

Asistent Universitar DANCIU Gabriel Mihail

Brașov 2020

**Departamentul .....................................................**

**Programul de studii: Calculatoare**

***IANCU George-Alexandru***

Aplicație Web pentru monitorizarea calității aerului

**Conducător științific:**

Doctor Asistent Universitar DANCIU Gabriel Mihail

Brașov 2020

**Cuprins**

Cuprins

[1. Introducere 5](#_Toc44023788)

[1.1. ACTUALITATEA SI OPORTUNITATEA TEMEI 5](#_Toc44023789)

[1.2. Scopul Proiectului 6](#_Toc44023790)

[1.3. Diagrama proiectului 7](#_Toc44023791)

[1.4. Structura proiectului 7](#_Toc44023792)

[2. Design 7](#_Toc44023793)

[2.1. ADOBE EXPERIENCE DESIGN 8](#_Toc44023794)

[2.2. CULORI 8](#_Toc44023795)

[2.3. BRANDING 9](#_Toc44023796)

[2.4. USER EXPERIENCE & USER INTERFACE 9](#_Toc44023797)

[2.4.1. User interface 9](#_Toc44023798)

[2.4.2. User experience 10](#_Toc44023799)

[2.5. STRUCTURA VIZUALA A APLICATIEI 11](#_Toc44023800)

[3. Implementarea Front-End 12](#_Toc44023801)

[3.1. FRAMEWORK-UL REACT.JS 12](#_Toc44023802)

[3.1.1. Despre React.JS 12](#_Toc44023803)

[3.1.2. Despre D.O.M. si Virtual D.O.M. 13](#_Toc44023804)

[3.1.3. De ce folosim Virtual D.O.M. 13](#_Toc44023805)

[3.1.4. Despre JSX 15](#_Toc44023806)

[3.2. Structura aplicatiei Front-End 15](#_Toc44023807)

[3.3. Componenta Aplicatiei 15](#_Toc44023808)

[3.4. Componenta User - Login 19](#_Toc44023809)

[3.5. Componenta User - Resigter 21](#_Toc44023810)

[3.6. Componenta User – Resigter Confirmation 22](#_Toc44023811)

[3.7. Componenta User – Dashboard 23](#_Toc44023812)

[3.7.1. Componentele imporate in obiect 23](#_Toc44023813)

[3.7.2. Functii auxiliare si preluarea datelor 24](#_Toc44023814)

[3.7.3. Randarea elementelor in pagina 28](#_Toc44023815)

[3.7.4. Performanta, vizualizare si integrare 28](#_Toc44023816)

[3.8. Stilizari ale aplicatiei 30](#_Toc44023817)

[3.8.1. Strucutura B.E.M. 30](#_Toc44023818)

[3.8.2. Schema pirmidala a stilurilor 31](#_Toc44023819)

[3.8.3. Variabilele generate prin pre-procesatorului SCSS 31](#_Toc44023820)

[3.9. Testari de unitate 33](#_Toc44023821)

[4. Implementarea Back-End 34](#_Toc44023822)

[4.1. Module Specifice Server-ului 34](#_Toc44023823)

[4.2. Preluarea Datelor Pentru Indicii Inregistrati 34](#_Toc44023824)

[4.3. Preluarea Datelor Pentru Utilizatori 34](#_Toc44023825)

[4.4. Sistemul De Salvare Al Log-urilor 34](#_Toc44023826)

[4.5. Routearea Serverului 34](#_Toc44023827)

[4.6. Certificarea SSL 34](#_Toc44023828)

[5. DevOps 34](#_Toc44023829)

[5.1. Integrare Continua 35](#_Toc44023830)

[5.2. Livrare Contiuna 35](#_Toc44023831)

[Abstract 35](#_Toc44023832)

# Introducere

|  |
| --- |
| **Actualitatea si oportunitatea temei**  **Fonturile**  **Editarea primei pagini**  **Structurarea lucrării**  **Subcapitol**  **Alt subcapitol**  **Gestionarea figurilor, formulelor, tabelelor**  **Recomandări generale**  **Comenzi rapide**  **Citarea referințelor** |

## ACTUALITATEA SI OPORTUNITATEA TEMEI

Traim in era informatiei, ceea ce se materielizeaza prin schimbari zilnice. Aceasta schimbare a avut ca rezultat producerea mai multor interconectari ca în trecutul apropiat. Utilizatorii laici, cât și experții din domeniu sunt intr-o graba permanenta, tehnologia tinand pasul cu aceasta, tehnologia fiind într-o schimbare rapidă și constanta. Odata cu aceasta evoluție, in ultimele decenii, poluarea a avut o creștere exponentiala, ce și-a făcut loc in vietile noastre, reprezentant un factor de risc atat pentru generatia curenta, cat și un factor alarmant pentru generatiile urmatoare. In prezent un numar tot mai mare de ONG-uri, au adus la cunoștiința autoritatilor de la nivele inalte aceasta problema, cu care ne confruntam tot mai des in zilele noastre. Sunt de parere ca tehnologia desi intr-o oarecare masura are o natura destructiva, din perspectiva perisarii mediului inconjurator, aceasta ne poate ajuta în viața de zi cu zi, sau chiar în unele situații se poate afla în ipostaza de a salva vieți, sau de a ne facilita viața în situații limita.

In ultima decada viata noastră a suferit un numar major de schimbari, datorita controlului pe car îl deținem asupra tehnologiei. Cu ajutorul senzorilor si aplicatiilor ce sunt destinate facititarii și ridicarii calitatii viații. Datorita acestor schimbari, cât și obisnuintei de avea un stil de viața, ce nu este concentrat pe rezolvarea unor sarcini, ce în trecut puteau pune probleme, atât din punct de vedere al materiei umane implicate, cât și a timpul epuizat, dispozitivele si aplicatiile inteligente au devenit o parte importanta din viata noastra.

Datorită aplicatiilor cu specific putem sa observam si sa ne documentam, într-o maniera mai facila decât în trecut, asupra dificultatilor întâlnite, în viata zi de zi. Aceste aplicatii ce au la baza functia de a analiza date si statistici, de cele mai multe ori sunt integrate sub forma aplicatilori IoT (Internet of things). La baza principiului lor de functionare stau senzori ce capteaza informatia si software-ului folosit pentru gestionarea si prelucrarea datelor captate, cât si totodata a afisarii acestora.

Mai nou toate aplicatiile utilitare cât și aplicațile de statistica au prins un impact grafic foarte intuitiv respecand principii de baza în strânsa legătura cu datele și continutul. Multe dintre ele acum au un design responsive[1] acestea fiind disponibile atât pe laptop cât și pe alte dispozitive mobile cum ar fi telefon / tableta.

Monitorizarea constanta este un principiu al umanitatii, așa cum noi avem grija de numărul de calorii pe care le mâncam așa putem avea grija și de calitatea aerului pe care îl respiram și în care trăim. Prin aceasta putem să ne asiguram o statistica și o previziune a viitorului în ceea ce privește poluarea și desigur cât și controlarea persoanala în a evita plouarea mediului inconjurator sau a reduce-o.

## Scopul Proiectului

Realizarea unei aplicatii web pentru monitorizarea poluarii prin tehnologii noi. Crearea si folosirea de baze de date pentru big data search[2] folosind ElasticSeach si Amazon Web Services, reproducerea datelor pe un server RESTful API[3] construit pe tehonologia Node.JS si pe framework-ul Express pentru preluarea / modificarea datelor dar si postarea lor in baza noastra de date. Crearea de utlizatori ai aplicatiei si crearea dispozitivelor digitale si analogice pentru ca acestia sa inregistreze valori din perimetul localitatii lor. Vederea in ansamblu a datelor prin tehnologia React.JS care ne ofera posibilitatea de a avea si o gestiune personala pe baza de useri. Integrarea de algoritmi specifici pentru preluarea datelor prin limbajul de programare Javascript. Utilizarea de server side rendering pentru partea de front-end si intregrarea de server RESTful pentru patrea de back-end. Configurarea intregii aplicatii pe un virtual private server[4]. Ajustarea design-ului prin intermediul programului utilitar Adobe Experience Design[5]. Crearea metodelor de interogare a serveru-ului.

## Diagrama proiectului

AICI TREBUIE SA PUN O POZA.

Un utilizator deține dispozitivul de inregistrare a calitatii aerului. Acest dispozitiv transmite date prin REST API-ul creat în partea de back-end, server-ul la rândul sau transmite date către ElasticSearch pe index-ul specific utilizatorului. Acestea odată preluate la accesarea partii de front-end, aplicația de client face cerere la server pentru a aduce date și a concatena toate datele obtinute de la alți utilizatori, acestea sunt afisate sub o forma tableara, în grafice specifice fiecarui tip de date inregistrare și vederea de ansamblu pe harta pusa la dispoziție sub forma de map markers [5] și heat maps [6].

## Structura proiectului

AICI TREBUIE SA PUN O POZA.

- Design: Adobe Experience Design

- Baze de date: ElasticSearch & AWS (Amazon Web Services)

- Back-end: Node.JS & Express

- Front-end: React.JS & Webpack

- DevOps: depunere aplicație în productie

# Design

|  |
| --- |
| **Actualitatea si oportunitatea temei**  **Fonturile**  **Editarea primei pagini**  **Structurarea lucrării**  **Subcapitol**  **Alt subcapitol**  **Gestionarea figurilor, formulelor, tabelelor**  **Recomandări generale**  **Comenzi rapide**  **Citarea referințelor** |

## ADOBE EXPERIENCE DESIGN

Adobe Experience Design prescurtat Adobe XD este o aplicație utilitara ce permite utilizatorilor sa creeze interfete grafice pentru aplicații de web și mobile. Acest softaware premite suport pentru fisierele vectoriale fiind foarte util la redimensionarea scalara a iconitelor sau elementelor grafice prezente în interfata. Prin Adobe XD se pot crea prototipuri pentru a avea o sensatie cât mai naturala a aplicație sau website-ului care sunt deja în curs de dezvoltare și nefinalizate. Se pot redimensiona activ și imagini sau elemente care conpun mock-upul[7] și se pot materializa în prototipuri pentru diverse dispozitive. Aplicația creaza culori primare și comeplentare pentru a ajuta designerul în realizarea unei interfete consistente din punct de vede grafic fără a a avea treceri bruste intre culorile de baza și culorile cu contrast dur care pot provoca utilizatorului o experiența grafica proasta. Paletele complementare de culori sunt create pentru a avea un sentiment de echilibru și a extinde gama de culori primare. Gama larga pe care o ofera aceasta paleta este extrem de versatila și interesantă pentru ochiul nostru, fiind pe deantregul o plăcere de a vizualiza aplicația.

## CULORI



Lime green: Aceasta este o culoare verde strălucitoare, asemănătoare cu cea a pielii fructelor citrice. Aceasta exprima vitalitate, incredere si este strans legata cu natura, deoarece aplicatia prezentata vrea sa exprima grija de natura, de cadrul natural al Brasov-ului. Alegerea culorii a fost legata de acest oras fiind un oras de munte incojurat de paduri si dealuri.

White Flat: Reprezinta pozivitatea in acest cadru sumbru al poluarii. In acest gri neincetat, speram ca prin aplicatia “Harta Poluare Brasov” va ajuta oamenii / populatia sa inceteze poluare recreand un cadru armonios.

## BRANDING



Branding-ul aplicatiei, mai exact logo-ul creat este format din titlul aplicatiei facut pe doua randuri iar in mijloc o frunza si o harta. Frunza exprima girja pentru natura si mediul inconjurator, iar harta se refera la functionalitatea de baza a aplicatiei dar si la incluziunea si valoarea mediului inconjurator suprins in tehnologie.

## USER EXPERIENCE & USER INTERFACE

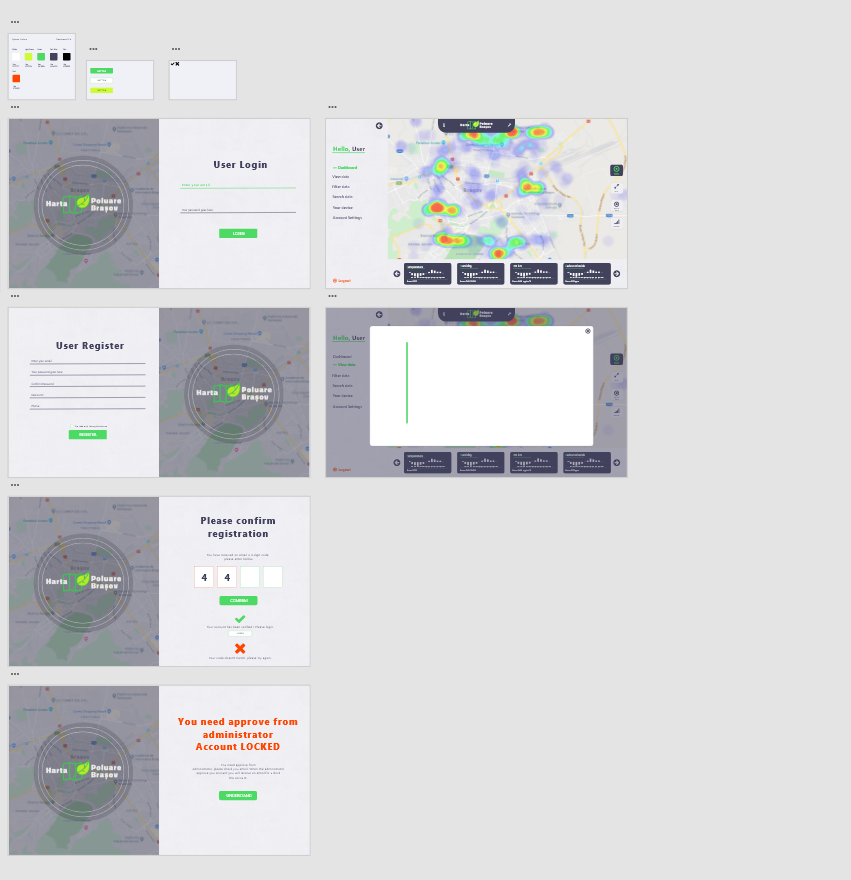
### User interface

Interfata utilizatorului este mijlocul prin care o persoana controleaza o aplicație software sau un dispozitiv hardware. Aceasta se prescurteaza cu UI. O intrefata cat mai simpla ca aceasta faciliteaza si o experienta “usor de folosit”, permitand utlizatorului sa controleze si sa afiseze cat mai usor datele cuprinse de senzori prin partea de back-end a aplicatiei intr-un mod cat mai natural si intuitiv. Ca si designer de aplicatie am dorit sa ofer o interfata consistenta din punct de vedere butoane si culori, obisnuind utilizatorul sa aiba un comportament normal atunci cand aceseaza partea de client a aplicatiei adica front-end-ul.

* Ca si elemente grafice am folosit 3 tipuri de butoane cu actiuni similare atunci cand utilizatorul trece cu mouse-ul peste sau cand acesta face click pe un buton, fiind prin acest mod consistente vizuale si sa respecte principile de baza a UI.
* Tabelele si graficele folosite sunt pe un ton inchis cariua ii sugereaza utilizatorului o profunzime atunci cand verifica sau vizualizeaza datele extrase din server.
* Pop-urile aplicatiei (“modal-urile”) din aplicatia web au fost contruite prin aceiasi structura vizuala, pastrand o experienta constanta si inducand utilizatorului senzatia de consistenta.
* Input-urile folosite sunt unice pe intreaga platforma avand in permanenta un label intuitiv pentru cerinta acestuia si o bara de completare verde / rosie in functie de completarea facuta de utilizator

### User experience

Design-ul din punctul de vedere a experientei utilizatorului, abreviat in engleza UX Design, reprezinta toate experiențele și interacțiunile pe care utilizatorii le-au avut cu fiecare aplicație pe care au folosit-o în viața lor au ajutat la stabilirea așteptărilor lor cu privire la modul în care ar trebui să funcționeze interfețele. Pentru aceasta aplicatie s-au folosit WildFrame-uri in aplicatia Adobe XD care imita functionalitatea evenimentelor petrecute in aplicatie. Acestea sunt structurate prin trageri de “evenimente” de la un obiect la altul, iar in mediul de testare oferit de aplicatia Adobe XD utilizatorul poate acesa aceste evenimente pentru a vedea un rezultat final, spre exemplu deschiderea unui Pop-up la acesarea unui buton.



## STRUCTURA VIZUALA A APLICATIEI

Aplicatia contine 4 interfere principale:

* Login
* Register
* User confirmation
* Dashboard
  + Componenta pentru tabele si grafice
  + Componenta pentru hartile Google
  + Componenta din partea stanga pentru meniu-ul utilizatorului
    - Butoane si evenimente pentru utilizator
  + Componeta pentru informatiile aplicatiei aflata sus
  + Componenta pentru butoanele de filtrare pe harta

Structura design-ului este construita pentru o aplicatie de tip single-page application[8].

# Implementarea Front-End

|  |
| --- |
| **Framework-ul React.JS**  **Despre React.JS**  **Despre D.O.M. si Virtual D.O.M.**  **De ce folosim Virtual D.O.M.**  **Despre JSX**  **Structura aplicatiei Front-End**  **Componenta Aplicatiei**  **Componenta User - Login**  **Componenta User - Register**  **Componenta User – Confirm Registration**  **Componenta User – Confirm Registration**  **Componenta User – Dashboard**  **Stilizari ale aplicatiei** |

## FRAMEWORK-UL REACT.JS

### Despre React.JS

Acest mic framework lansat in anul inceputul anului 2013 este o bibliotecă JavaScript pentru construirea de interfețe de utilizator, fiind eficienta si flexibila pentru construirea de interfete. Privind modelul M.V.C. (Model View Controller) aceasta librarie este “V” dintre cele doua. Aceasta este open-source si la nivel de implementare se bazeaza pe componente si clase acestea la randul lor fiind responsabile pentru stratul de vizualizare al aplicatiei. React.JS este creata si intretinuta de compania Facebook, ceea ce ofera o confidenta mare in acest framework fiind unul stabil.

Din punct de vedere proces de dezvoltare pe aceasta platforma, ea proiecteaza vizualizari simple pentru fiecare stare din aplicatie, iar tot aceasta va actualiza eficient si va reda doar componenta corecta / activa atunci cand utilizatorul cere o schimbare de date / componente din aplicatie. Aceste schimbari sunt denumite in engleza “states”.

Vizualizarea declerativa face ca codul aplicatiei sa fie mult mai previzibil si usor de depanat la nevoie, deoarece totul este stratificat in aceste componente.

O aplicatie React.JS este formata din mai multe componente acestea fiind responsabile pentru redarea unei mici bucati, reutilizabile de cod HTML. Acestea sunt amplasate din componenta in componenta pentru a crea o aplicatie complexa, stabila avand la baza structuri mici sau mari.

React.JS a fost lansat pentru dezvoltarea de aplicatii single-page applications. Am ales acesta librarie pentru acest concept de baza implementandu-l in Harta Poluare Brasov.

### Despre D.O.M. si Virtual D.O.M.

* Realitatea D.O.M
  + În primul rând, D.O.M. înseamnă „Model obiect document”. D.O.M.-ul în cuvinte simple reprezintă UI-ul aplicației. De fiecare dată când există o modificare a stării UI a aplicației, D.O.M. se actualizează pentru a reprezenta această modificare. Acum capturile manipulează frecvent D.O.M. afectează performanțele, ceea ce îl face lent.
* De ce D.O.M. este lent ?
  + D.O.M. este reprezentat ca o structură de date arbore. Din această cauză, modificările și actualizările la DOM sunt rapide. Însă, după modificare, elementul actualizat și copiii să fie redirecționati pentru a actualiza interfața de utilizare a aplicației. Redirecționarea sau re-pictura UI este ceea ce îl face lent. Prin urmare, cu cât aveți mai multe componente UI, cu atât actualizările DOM pot fi mai ineficiente.
* Virtual D.O.M.
  + Conceptul de eficientizare a acestei probleme parcurse mai sus vine prin libraria React.JS, aceasta avand ca si abstractie un D.O.M. Virtual acesta fiind un middleware intre cel real si cel virtual prin incarcare doar a starilor modificate in aplicatie, urmand doar o re-picturare a Modelului Obiect, in cazul React.JS, componenta care isi schimba starea.

### De ce folosim Virtual D.O.M.

De ce trebuie folosit Virtual D.O.M. din prima inițializare a proiectului ? Deoarece aplicația noastră web nu este statica. React.JS are o stare care poate fi reprezentata de UI (User Interface) prin modificari de evenimente. La ce evenimente se face referinta ?

* Evenimente de utilizator
  + Acestea se întâmpla atunci când utilizatorul scrie, actiuneaza click-ul, actioneaza scroll sau resize. Acestea sunt principalele evenimente suportate în aplicația Harta Poluare Brașov
* Evenimente de server
  + Acestea se actioneaza când aplicația primește date sau erori din partea server-ului de date, mai exact partea de Back-end.

Ce se întâmpla atunci când folosim aceste evenimente descrise mai sus ? În general noi actualizam datele din aplicație care sunt necesare și datele reprezentate de starile din modelul de date. Pentru acestea, noi avem o stare care se schimba din modelul de date și ne dorim sa reflectam schimbarea în UI. Pentru completarea acestei acțiuni avem nevoie de metoda de two-way data binding [ref] care se ne translateze din modelul de date către modelul UI schimbarile dorite și vice-versa sau metoda de KVO (Key-Value Observing) [ref], iar aceasta este o metoda care folosește modelul de observare al starilor. În prima faza, trebuie să se asigure ca obiectul pe care îl observa este asignat pentru aceasta funcție, iar prin schimbarile de cheie obiectul poate primi anumite stari. Acest proces este unul greoi provocand intreferente asupra interfetei blocand functionalitate, deci utilizatorul va sesiza interferențe pe aceasta parte.

(Poze din site-ul de KVO)

Deși avem aceste metode mai bune sau mai rele, libraria React.JS ofera o soluție de actualitate privind aceste schimbari de D.O.M. prin soluția implementata de Virtual D.O.M. Acesta este un proces ce ofera o viteza foarte rapidă pe baza schimbarilor de staturi. Aceasta este o abstractie care trateza folosind JavaScript D.O.M.-ul real ca fiind unul reactiv.

Funcționare în pași simplii:

* De fiecare data când starea aplicației se schimba, D.O.M.-ul virtual își schimba la rândul sau starea și randeaza în interfata valoarea dorita.
* React.JS calculează diferențele dintre unul sau mai multe D.O.M-uri virtuale, unul fiind cel care salvează starea trecută și restul schimbandu-si starea. Iar diferența dintre acestea este practic actual rezultat pe care vrem sa îl injectam în D.O.M-ul real.
* React.JS își face update doar dacă este nevoie în D.O.M.-ul real.

### Despre JSX

Ce este JSX ?

* JSX este o extensie pentru JavaScript care extinde în prima faza structura, metodele și obiectele JavaScript și limbajul de mark-up HTML și XML. Pentru noi ne ajuta la generarea de componente care au la baza limbaj JavaScript și elemente HTML. Prin aceasta este posibila crearea de structura declerativa pentru a definii D.O.M.-ul nostru .
* Expresiile JSX sunt tot expresii JavaScript care odată compilate devin un Obiect JavaScript.
* Aceste elemente de JSX vor primii în aplicația noastră toate evenimentele descrise mai sus la capitolul 3.1.3

## Structura aplicatiei Front-End

Structura proiectului a fost sintetizata în modelul de baza React.JS prin metoda A.C.F.S. :

* **ASSETS**
* **COMPONENTS**
* **FUNCTIONS**
* **STYLES**
* **funcții javascript și configurari**

În folder-ul „src” prezent în folderul aplicației regasim aceste sub-foldere.

* În folderul de „assets” se regasesc fisierele statice randate de website
* În folderul de „components” se regasesc componentele care fac parte din interfata grafica
* În folderul de „functions” se regasesc functiile de baza pentru schimbarea interfetei folosind conceptul de KVO
* În folderul de „styles” se regasesc toate silizarile site-ului folosing post-procesul SCSS
* Celelate fisere auxiliare sunt fisiere JavaScript și fisiere de UnitTesting pentru aplicație.

## Componenta Aplicatiei

În folderul princial al aplicației „src” regasim fișierul „App.js” prin care se face conexiunea cu restul aplicației. În acest fisier se prezinta în partea de sus toate componentele auxiliare folosite pentru rularea aplicației. Componentele de baza pentru acesta sunt componente de sine statatoare fiind un obiecte destinate parsarii rutarii, importari de module pentru server-side rendering dar și importari de componente ce definesc interfata UI.

*import* React, { Component } *from* 'react';

*import* {BrowserRouter *as* Router , Route, Switch, Redirect} *from* 'react-router-dom';

*import* axios *from* "axios";

*import* UserLogin *from* "./components/login/UserLogin";

*import* UserRegister *from* "./components/login/UserRegister";

*import* UserConfirmation *from* "./components/login/UserRegisterConfirmation";

*import* HomepageNoLogin *from* "./components/Homepage";

*import* UserDashboard *from* "./components/DashboardUser";

Codul 1 Cod sursă pentru exemplificare importare componente

Clasa „App” este o structure abstracta de predefineste workflow-ul aplicației. Aceasta spre exemplu predefinete aplicației stilurile reposnabile de design ce se afla în folderul „STYLES” din folder-ul principal. Acestui obiect i se atribuie un constructor care ne permite parsarea de date și stari, ceea ce o face ca fiind o clasa abstracta care poate fi apelata din orice context al aplicației.

*import ‘./styles/styles.min.css’;*

*class* App *extends* Component

*constructor*(props) {

*super*(props);

*this.*state = {

            loggedIn: false

        }

    }

Codul 2 Cod sursă pentru exemplificare definire clasa in React.JS

Principala stare pe care o defineste este starea de inregistrare a utilizatorului.

„componentDidMount” este o metoda prin care obiectul este initalizat, cu alte cuvinte, la initializarea obiectului nostru aceasta metoda este apelata pentru a genera o anumita stare. Pentru aplicația noastră starea pe care o genereaza este prin apelarea server-ului Back-End pentru a aduce date despre configurarea și inregistrarea utilizatorului. Dacă server-ul ne răspunde cu un mesaj de acceptare al utilizatorului, înseamnă ca utilizatorul a primit un session cookie de inregistrare. Metoda descrisa va prelua informația oferita și o va verifica prin metoda descrisa iar după aceea ii va asigna starii noastre o valoare booleana care certifica ca utilizatorul nostru este valabil sau nu pentru vizualizarea aplicației noastre.

componentDidMount() {

*var instance = axios.create({*

*withCredentials: true*

*})*;

        instance*.*get("http://localhost:3001/users/who-am-i")*.*then(response *=>* {

*if* (!response*.data.*user) {

*if* (response*.*data) {

*if*(Array*.*isArray(response*.*data)) {

*this.*setState({

                            loggedIn: true,

                            user: response*.*data[0]*.*user,

                        });

                    }

                }

*else* {

*this.*setState({ loggedIn: false });

                }

            }

*else* {

*this.*setState({ loggedIn: false });

            }

        });

    }

Codul 3 Cod sursă pentru exemplificare definire instanta Axios si setare stare

În cazul în care utilizatorul nu este certificat de server prin session cookie starea noastră se prelucreaza într-o stare falsa iar aceasta va împiedica utilizatorul de a accesa restul de functionalitati din aplicația mama. Dacă conditia de negatie este îndeplinită state-ul îsi face upadate în Virtual D.O.M. apeland randarea de componenta „App”, iar utilizatorul aplicației va fi redirectionat spre componentele de Logare sau Inregistrate pe care le voi explica în urmatoarele capitole.

În cazul în care utilizatorul este certificat de către server prin session cookie, starea noastră se va valida ca fiind una pozitiva și va reapela randarea de obiect a componentei „App” iar utilizatorul va ri redirectionat către principala componenta cu routa „/dashboard” sau „/”.

În aceste componente sa va duce și informația utlizatorului ca fiind una valida prin construirea de parametru numit în React.JS „props”. Acest obiect parsat mai departe se numește „user” și este un obiect de sine stator care va putea fi apelat în fiecare subcomponenta a componentei „UserDashboard”. Voi trece prin aceasta implementare în urmatoarele capitole.

    render() {

*return* (

            <Router>

                <div>

                    <Switch>

                        {

*this.state.*loggedIn ? (

                                <>

                                    <Redirect *to*="/" />

                                    <Route *exact* *path*="/" *component*={UserDashboard} />

                                    <Route *path*="/dashboard" *component*={UserDashboard} *user*={*this.state.*user}/>

                                </>

                            ) : (

                                <>

                                    <Route *exact* *path*="/" *component*={HomepageNoLogin} />

                                    <Route *path*="/login" *component*={UserLogin} />

                                    <Route *path*="/register" *component*={UserRegister} />

                                    <Route *path*="/confirm-register" *component*={UserConfirmation} />

                                    <Route *component*={HomepageNoLogin} />

                                </>

                            )

                        }

                    </Switch>

                </div>

            </Router>

        );

    }

Codul 4 Cod sursă pentru exemplificare React Router si verificarea starii de login

Aceste componente sunt inima aplicației Front-End oferita clientului și sunt un model de programare puternic declarativ al React.JS-ului.

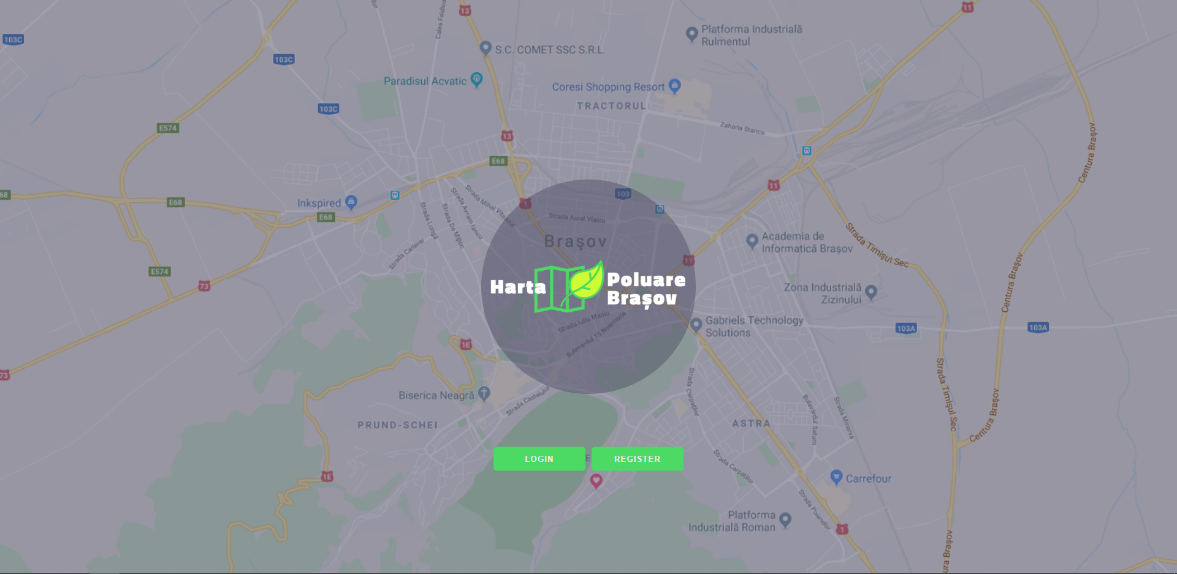
Principala semnatica pe care o folosește componenta principala sta la baza librariei din React.JS numita React Router.

React Router este o colecție de componente de navigație care se compun declerativ cu aplicația noastră. Avem nevoie de acete routari declerative pentru a seta state-uri și pentru a crea un sistem de optimizare al motoarelor de căutare cât mai prietenos cu aplicația noastră. Aceasta poate dispune și de funcții de „hash router” care pentru aplicatiile server-side sunt o adevărata bucurie. Aplicatiile server-side rendering nu dispund de o asemenea implementare, ele fiind randate prin cod brut și nu prin un limbaj de mark-up cum este HTML. Codul HTML este randat odată cu rularea script-urilor. Aceasta înseamnă ca sistemele de căutare nu vor putea indexa rutele noastre. Libraria React Router se ocupa de aceasta, generand un cache, în prealabil, pentru aplicație.

Prin libraria de React Router randez componentele ca fiind un switch de baza, odată ce utilizatorul este înregistrat sau logat pe aplicația noastră, router-ul își face switch către componenta specifica.

Libraria descrisa mai sus care este în interiorul compoenentei de randare principale se numește „axios”.

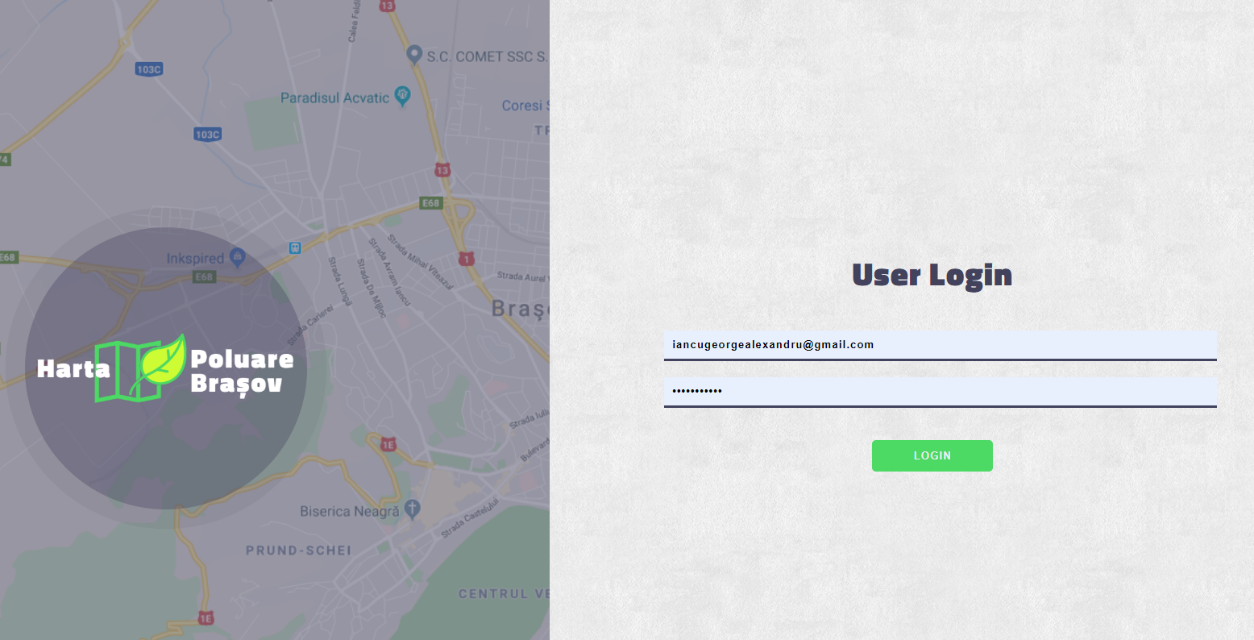
„Axios” este o librărie de mica greutate care efectueaza cereri HTTP pentru client. Aceasta librărie poate fi asemuita cu „Fetch API” din JavaScript ambele având ca baza serviciul „XMLHttpRequest”. Am ales aceasta librărie pentru compatibilitatea crescuta în domeniul browserelor, iar prin aceasta aducem un plus de valoare șistabilitate aplicației noastre. Aceasta poate fi instalata prin managerul de pachete Node.JS în aplicația noastră. Ea este prezentata în folder-ul „node\_modules” și este asignata aplicației prin managerul de procese oferit de React.JS.



## Componenta User - Login

De aceasta data, componenta prezentata face parte din obiectul „mama” adică App Component descris mai sus. Aceasta face parte în folder-ul „components/login/UserLogin.js”.

Notatia cu litera mare la începutul acestui obiect face referinta la o componenta care poate fi importata de o alta sau chemata în codul nostru ea insasi. Vom observa pe parcurs ca avem componente notate cu „\_” și altele notate cu litera mare la baza lor.



Componenta aceasta participa la nivelul de funcționare al aplicației:

USER -> LOGIN -> DASHBOARD USER

Pentru ca un utilizator sa poate acesa aplicația noastră, el trebuie mai întâi să fie înregistrat și logat în server-ul și baza noastră de date.

Aceasta componenta participa la nivelul de verificare al utilizatorului dacă este înregistrat sau va trebui înregistrat

Obiectul de referita face o apelare prin modulul „axios” descris la capitolul 3.3. care la rândul sau interogheaza server-ul nostru și ne aduce date privind utilizatorul care încearcă să se conecteze.

Aplicația în acest moment își seteaza o stare prin care logarea este activa sau nu.

Utilizatorul își introduce în formularul de input email-ul și parola care sunt trimise la server, iar apoi aceasta așteaptă un răspuns de la el.

Input-urile sunt asculate de evenimente „onChange” prin care se actualizeaza starea valorii care este stocata în obiectul principal gestionat de constructor-ul componentei. Acestea își fac bind în obiectul JavaScript, pentru a reuși sa avem conceptul de two-way data binding, prin metodele „onChangeEmail” și „onChangePassword”. Din aceste metode descrise se seteaza starea pentru fiecare input în parte respectiv „email” și „parola”.

*constructor*() {

*super*();

*this.*state = {

            email: "",

            password: ""

        };

*this.*onChangeEmail = *this.onChangeEmail.*bind(*this*);

*this.*onChangePassword = *this.onChangePassword.*bind(*this*);

    }

    onChangeEmail = (e) *=>* {

*this.*setState({ email: e*.target.*value });

    }

    onChangePassword = (e) *=>* {

*this.*setState({ password: e*.target.*value });

    }

Codul 5 Cod sursă pentru exemplificare concept KVO pentru schimbarea starii input-urilor

La baza având un element de tip <form> aplicația noastră reactioneaa cu server-ul prin metoda makeLogin() care este o funcție decalarata în interiorul aceste compoenente și poate fi apelata doar prin verificare acestor credentiale și doar din aceasta componenta. Metoda face o cerinta POST către server și așteaptă răspunsul acestuia, dacă răspunsul este de valoare 200 aceasta va face inregistrarea user-ului absolută, totodata schimbat și starea din componenta principala, care va fi redirectionat către aplicația principala. Dacă metoda primește un alt răspuns fata de 200 aceasta va returna un obiect gol, prin care autentificarea utilizatorului este imposibila și acesta trebuie sa își recompleteze formularul de conectare.

makeLogin() {

*var instance = axios.create({*

*withCredentials: true*

*})*;

        instance

*.*post("http://localhost:3001/users/login", {

                email: *this.state.*email,

                password: *this.state.*password

            })

*.*then(*function*(response) {

*if* (response*.*status === 200) {

                } *else* *if* (response*.*status === 401) {

                    alert("Unable to login, please try again!");

                }

            })

*.*catch(*function*(error) {

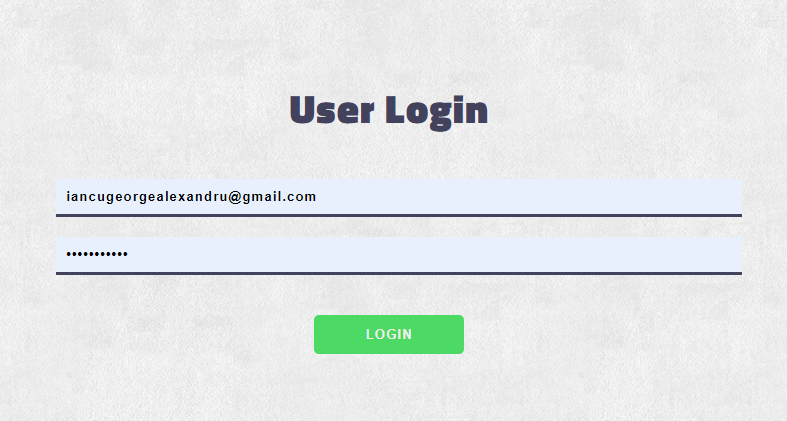
*if* (error) {

                    alert("Unable to login, please try again!");

                }

            });

    }

Codul 6 Cod sursă pentru exemplificare React Router si verificarea starii de login

## Componenta User - Resigter

Aceasta componenta prezenta în aplicația „Harta Poluare Brasov” creaza un utilizator în baza de date interogrand server-ul printr-o metdoa POST. Modulul folosit pentru aceasta interogare este „axios” care face legătura și cerinta către server. Acesta primește care și răspuns starea de 200 sau starea de 404. Starea de 200 este prezenta atunci când utilizatorul reușește prin completarea sa, sa prezinte toate informațiile pentru cont dar și informații privind acceptarea „Termenilor și Conditiilor”. Starea de 404 primita de la server reprezintă o autentificare greșită, sau inacesibilitatea server-ului la care se face cerere de date.

Ca și evenimente principale avem cele trei input-uri pe care utilizatorul trebuie sa le completeze și de asemenea checkbox-ul obligatoriu pentru acceptarea termenilor și conditilor.

Acestea verificări de date sunt actionate odată cu evenimentului de click pe butonul de inregistrare prin care se apeleaza functia „OnRegister()”.

Odată apelata, aceasta face referinta la datele descrise mai sus și interogeaza server-ul din partea de Back-End.

În aceasta componenta se descriu și alte evenimente pentru interfata U.I. prin care se realizeaza o interfata plăcută pentru utilizator indiferent de dispozitul de pe care acceseaza aplicația, el fiiind telefon / tableta sau desktop. Acestea sunt denumite ca și funcții și apartin din metodele folder-ului „FUNCTIONS” pe care le voi descrie în continuare. Functia se regaseste în „iancu.SplashScreen.js” care este un modul de sine statator și se importa în aceasta componenta sub denumirea de UI. Din acest fisier sunt extrase evenimentele pentru modificarea dimensiunii interfetei. Ca și obiect ele sunt descrise prin functiile „ModifySizeCol”. Ele sunt apelate de functia de construcție și deconstructie a componentei în care ne aflam. Prin acestea la redimensionarea interfetei se creaza un aspect plăcut pentru dispozitivul folosit.

*export const ModifySizeCol = () => {*

*let deviceHeight = window.innerHeight;*

*let deviceWidth = window.innerWidth;*

*let cols = document.getElementsByClassName("js-controler-height");*

*let child = document.getElementsByClassName("js-controler-height-child")[0];*

*let modifierSize;*

*if (deviceWidth <= 768) {*

*let modifierHolder = child ? deviceHeight / 2 + (child.offsetHeight - deviceHeight / 2 ) :  deviceHeight / 2;*

*modifierSize = modifierHolder + "px";*

*}*

*else {*

*modifierSize = deviceHeight + "px";*

*}*

*if(cols.length > 1) {*

*for (var i=0; i<cols.length; i++) {*

*cols[i].style.minHeight = modifierSize;*

*}*

*}*

*else {*

*cols[0].style.minHeight = deviceHeight + "px";*

*}*

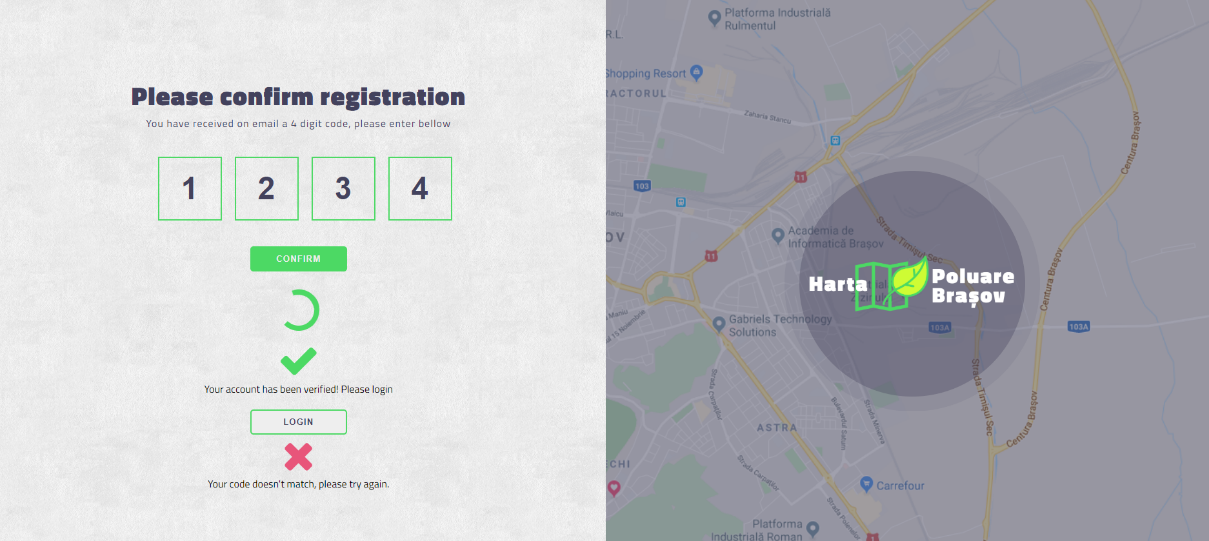
*}*

Codul 7 Cod sursă pentru exemplificare modificare functionalitate mobile devices

## Componenta User – Resigter Confirmation

Aceasta componenta se afla in starea de devzoltare, ulterior aceasta va face posibiliba verificarea utilizatorului printr-un cod unic primit pe email de 4 cifre intregi. Aceaste ulterior se intoduc in interfata componentei iar la actionarea butonului de confirmare, aplicatia va intregoga server-ul pentru a verifica veridicitatea utilizatorului ce se afla in baza de date. In acest moment aceasta componenta se afla in stadiu de creare al design-ului, aceasta ne fiind prezenta in React Router-ul aplicatiei. Se poate accesa doar din modul de dezvoltare.

In acest moment aceasta arata in felul urmator:



## Componenta User – Dashboard

Ajungand cu parcurgerea lecturii pana aici, descoperim componenta de baza care reprezinta functionalitatea aplicatiei din partea de utilizator. Ca si scurt intro vreau sa prezint functiile de baza, iar apoi intram in detalii suplimentare pentru fiecare in parte:

* Header – informatii despre aplicatie si creatori, plus logo-ul aplicatiei
* SideLeft – componenta care prezinta functionalitati de butoane pentru alte elemente, mai specific modal-uri in aplicatie
* MapContainer – componenta care ne incarca pe harta datele si pozitile inregistrarilor preluate din Back-End
* YourDeviceModal – componenta de tip modal ce prezinta informatii suplimentare pentru utilizatorul aplicatiei
* ViewDataModal – componenta principala pentru vizualizarea datelor in aplicatie
* BottomCharts – componenta de vizualizare rapida care se afla la finalul paginii afisate

### Componentele imporate in obiect

Primele doua componente descrise mai sus pe care le imporatam in acest obiect <Header /> si <SideLeft /> sunt componente de sine statoare care nu pot primi parameterii aparte pentru functionarea lor. Acesta insemana ca le putem aduce in orice fereasta fara avea constrangeri din punct de vedere cod si functionalitate. Pentru restul de componente avem anumiti paramentrii care sunt ficsi din punct de vedere format. Ii voi explica intr-un tabel.

|  |  |
| --- | --- |
| **Componenta** | **Parametrii** |
| MapContainer | heatMapData: Obiect de Array-uri  toggleMarkers: Boolean |
| YourDeviceModal | User: Obiect |
| ViewDataModal | dateStart: Obiect de tip Date  dateEnd: Obiect de tip Date  dataNo2: Obiect de Array-uri  labelNo2: String  limitWidth: Boolean  dataO3: Obiect de Array-uri  averageNo2: Integer  averageO3: Integer  dateStartMQ: String  dateEndMQ: String  dataMQ135: Obiect de Array-uri  labelMQ135: String  averageMQ135: Integer  air\_temperature: Obiect de Array-uri  air\_humidity: Obiect de Array-uri  air\_dustDensity: Obiect de Array-uri  air\_MQ125: Obiect de Array-uri  air\_averageTemperature: Integer  air\_averageHumidity: Integer  air\_averageDustDensity: Integer  air\_averageMQ135: Integer  airDateStart: Obiect de tip Date  airDateEnd: Obiect de tip Date  air\_label\_temperature: String  air\_label\_humidity: String  air\_label\_dustdensity: String  air\_label\_MQ135: String |
| BottomCharts | Aceasta componenta mosteneste componenta de ViewDataModal, avand aceiasi parametrii. |

Toti parametrii descrisi mai sus sunt necesari pentru buna functionare a aplicatiei si pentru randarea in intregime a datelor primite de la server. In continuare o sa vedem si anumite modele si functii pentru parcurgerea cu succes a acestor parametrii.

### Functii auxiliare si preluarea datelor

Functia principala dupa cum ne-am obisnut este “componentDidMount” ce reprezinta functia care se apleaza odata cu incercarea de punere in pagina a interfetei curente. Aici sunt declarate functiile de preluare a datelor din server-ul Back-End. La inceputul functiei avem doua variabile care ne indica ziua actuala si ziua precedenta. In aceste variable se returneaza valoarea din metoda publica getDates(). Aceasta functie verifica data in care suntem pentru ziua actuala dupa aceia preia aceasta informatie si suprascrie si valoarea datei calendaristice precedente. Aceasta returneaza un format de tip “DD-MM-YYYY”

    getDates() {

*let d = new Date()*;

*let myDay = d.getDate() < 10 ? "0" + d.getDate() : d.getDate()*;

*let myDayMonth =*

*d.getMonth() < 10 ? "0" + (d.getMonth() + 1) : d.getMonth() + 1*;

*let myDayYear = d.getFullYear()*;

*let lastDay = new Date(Date.now() - 864e5).getDate()*;

*let lastDayMonth =*

*new Date(Date.now() - 864e5).getMonth() + 1 < 10*

            ? "0" + (new Date(Date*.*now() - 864e5)*.*getMonth() + 1)

            : new Date(Date*.*now() - 864e5)*.*getMonth() + 1;

*let lastDayYear = new Date(Date.now() - 864e5).getFullYear()*;

        lastDay = lastDay < 10 ? "0" + lastDay : lastDay;

*let dateStart = lastDayYear + "-" + lastDayMonth + "-" + lastDay*;

*let dateEnd = myDayYear + "-" + myDayMonth + "-" + myDay*;

*this.*setState({

            dateStart,

            dateEnd

        });

*return* {

            dateStart,

            dateEnd

        }

    }

Codul 8 Cod sursă pentru exemplificare aducere de date calendaristice

Odata avand aceste date calendaristice putem interoga server-ul nostru Back-End pentru a ne aduce valori dintr-un serviciu de date open-source cu privire la valoare de No2 si O3 din aer. Interogarea se face cu metoda din JavaScript numita fetch(). Aceasta este o metoda GET care primeste ca raspuns un obiect de tip JSON. Aceasta este o metoda asincron care nu primeste datele in timpul cereri, pentru aceasta avem nevoie de o conditie de Then – Catch, cu alte cuvinte, odata primit raspunsul de la server, prelucreaza datele sau daca intampini o eroare prinde eroarea si lasa aplicatia sa ruleze in continuare.

*let dateStart = this.getDates().dateStart*;

*let dateEnd = this.getDates().dateEnd*;

        fetch(

            "http://localhost:3001/data/opensource/openaq/100/"+dateStart+"/"+dateEnd

        )

*.*then((response) *=>* response*.*json())

*.*then((data) *=>* {

Codul 9 Cod sursă pentru exemplificare aducere date prin metoda fetch()

Urmeaza o cod asemanator pentru fiecare cerere privind datele cerute de la server pentru device-urile create de noi si cele care ruleaza. De mentionat aici ca server-ul aduce 1500 de date fiind limitat din punct de vedere preformanta, pentru ca pagina sa se incarce cat mai rapid posibil, pentru a avea un User Experience cat mai placut.

Obiectele noastre primesc in principal doua valori pentru axa X si axa Y, care sunt necesare in crearea obiectului de grafice, pe obiectul X avem un obiect de tip Date generat prin TimeStamp si pe obiectul Y avem valoarea exacta a datelor inregistrate. Aceste obiecte sunt concatenate intr-un Array prin metoda push().

airMQ135*.*push({

                        x: new Date(item*.\_source.*timestamp),

                        y: item*.\_source.*MQ135

                    });

Codul 10 Cod sursă pentru crearea obiectului de array-uri a datelor

După cum spuneam, în librăria React.JS toate datele sunt salvate sub forma de stări, atunci când toate datele sunt preluate și prelucrare aceasta se salvează în starea principala a aplicației. Pentru obiectele de tip HeatsMap prin ES6 facem o concatenare de obiecte dintre coordonatele georgrafice pe care le-a înregistrat un device și coordonatele georgrafice pe care le aducem din RestAPI-ul third-party.

*this.*setState({

                    air: {

                        temperature: airTemperature,

                        humidity: airHumidity,

                        dustDensity: airDustDensity,

                        MQ135: airMQ135,

                        averageTemperature: averageTemperature / data*.data.*length,

                        averageHumidity: averageHumidity / data*.data.*length,

                        averageDustDensity: averageDustDensity / data*.data.*length,

                        averageMQ135: averageMQ135 / data*.data.*length,

                        airDateStart,

                        airDateEnd,

                        positions

                    },

                    heatMapTest: {

                        positions: [

                            ...positions,

                            ...*this.state.*positionsArray

                        ]

                    }

                }, () *=>* {

                    console*.*log(*this.*state);

                });

Codul 13 Cod sursă pentru verificarea starii de validare a datelor

Pentru nu avea probleme odată cu încărcarea elemtelor în pagina, verficam înainte de a chema obiectele toate date să existe, aceasta verificare se face prin metdoda de if clouser care ne returnează o valoare Booleana pentru verificarea noastră. Cu alte cuvinte daca rezultatul verificării nu satisface condiția nu mai aduce componenta în pagina, deoarce e posibil să aibă erori. Expemplu de funcționare JSX:

{*this.state.*no2Data && *this.state.*o3Data && *this.state.*air && !*this.state.*errorOnAir ? <ViewDataModal …params />}

Codul 14 Cod sursă pentru verificarea starii de validare a datelor

Pentru evenimentele pe care utilizatorul le face, cum ar fi schimbarea de interfata din pagina principala intr-o pagina de tip modal avem in folder-ul “FUNCTIONS” doua metode care se ocupa cu gestionarea vizibilitatii dintre modal sau pagina. Ca si eveniment, atunci cand utilizatorul actioneaza functia de click pe un buton specific de aducere al modal-ului starea aplicatiei se schimba iar Virtual D.O.M. compara rezultatele dintre starturile sale si se apleaza functia de deschidere respective inchidere al elementului. Acestea sunt exportatate catre aplicatie ca si constate fiind accesibile prin importare in orice componenta sau obiect. Pentru adugarea vizibilitatii se adauga o clasa CSS ce spune stilului aplicatiei ca acesta sectiune este vizibila si este in pagina.

*export const openModal = (element, data) => {*

*var modal = document.querySelector("[data-modal='"+data+"']");*

*modal.classList.add("modal-overlay--open");*

*}*

*export const closeModal = (element, data) => {*

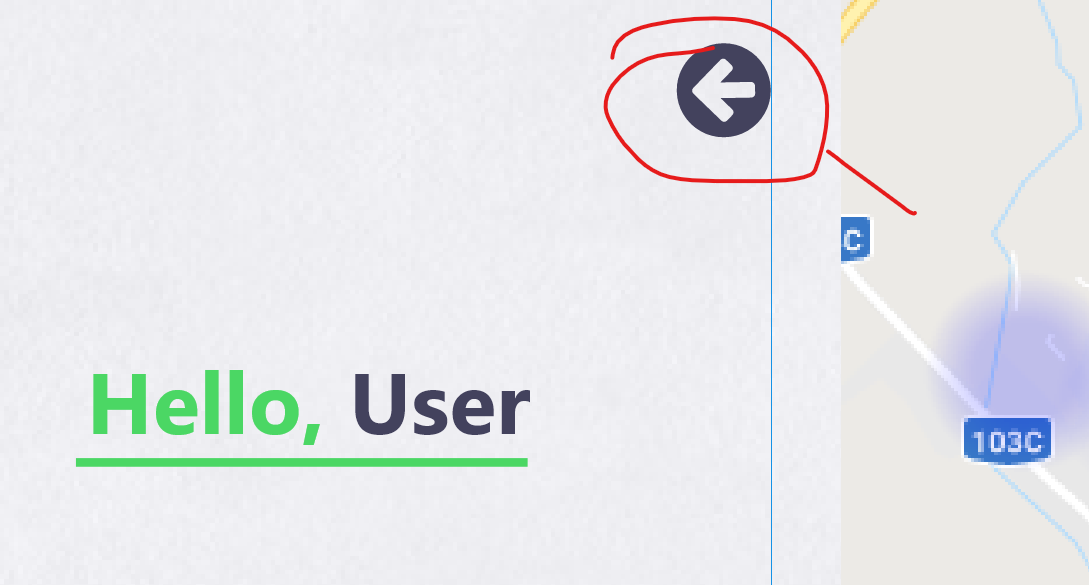
*var modal = document.querySelector("[data-modal='"+data+"']");*

*modal.classList.remove("modal-overlay--open");*

*}*

Codul 15 Cod sursă pentru exemplificare functionalitate modal-uri / pop-uri

O alta functionalitate de acest gen se intampla, folosind modelul KVO, atunci cand utilizatorul incearca restrangerea meniului din stanga, prezent pe pagina principala a aplicatiei “Harta Poluare Brasov”. Functionalitate este prezenta prin doua metode, si acestea fiind constante si implicite in fiecare obiect la cerere, ele fiind denumite toggleSlide() si makeActiveMenu(). Prima functie denumita prezinta un clouser care are un tip de eveniment de adaugare a clasei vizibile si respective retragerea clasei pentru vizibilitate. Aceasta sunt comparate printr-un string de “AddClass” si “RemoveClass”. Acest clouser function se aplica pentru icon-ul din SideBar.

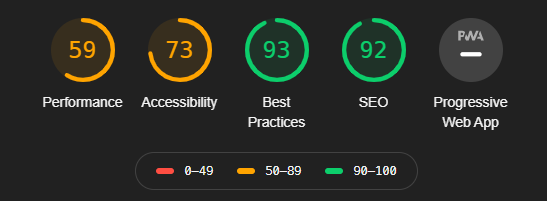


### Randarea elementelor in pagina

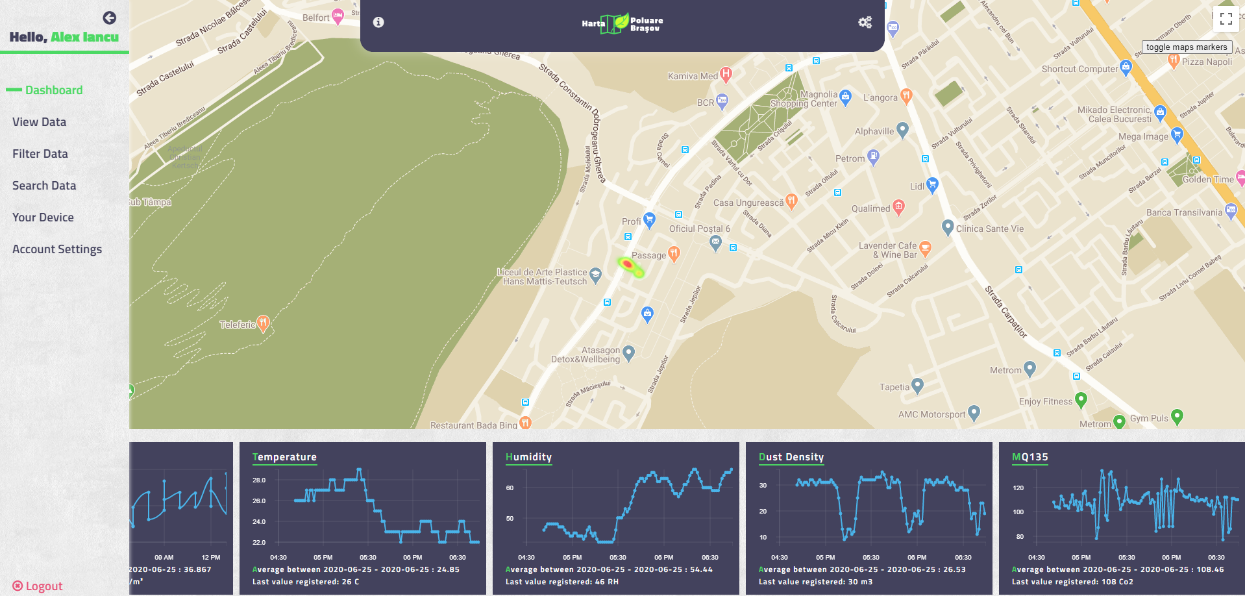
Randarea elementelor în pagina se realizează printr-un sincron “forțat” prin care trebuie verificate toate stările care sunt create în modul asincron al aplicației. După îndeplinirea tuturor condițiilor Boolene elementele auxiliare, ele fiind componente, funcții sau elemente de mark-ul HTML sunt predate către Vitrual D.O.M care vă tine evidenta schimabrilor de stări pentru fiecare în parte.

### Performanta, vizualizare si integrare

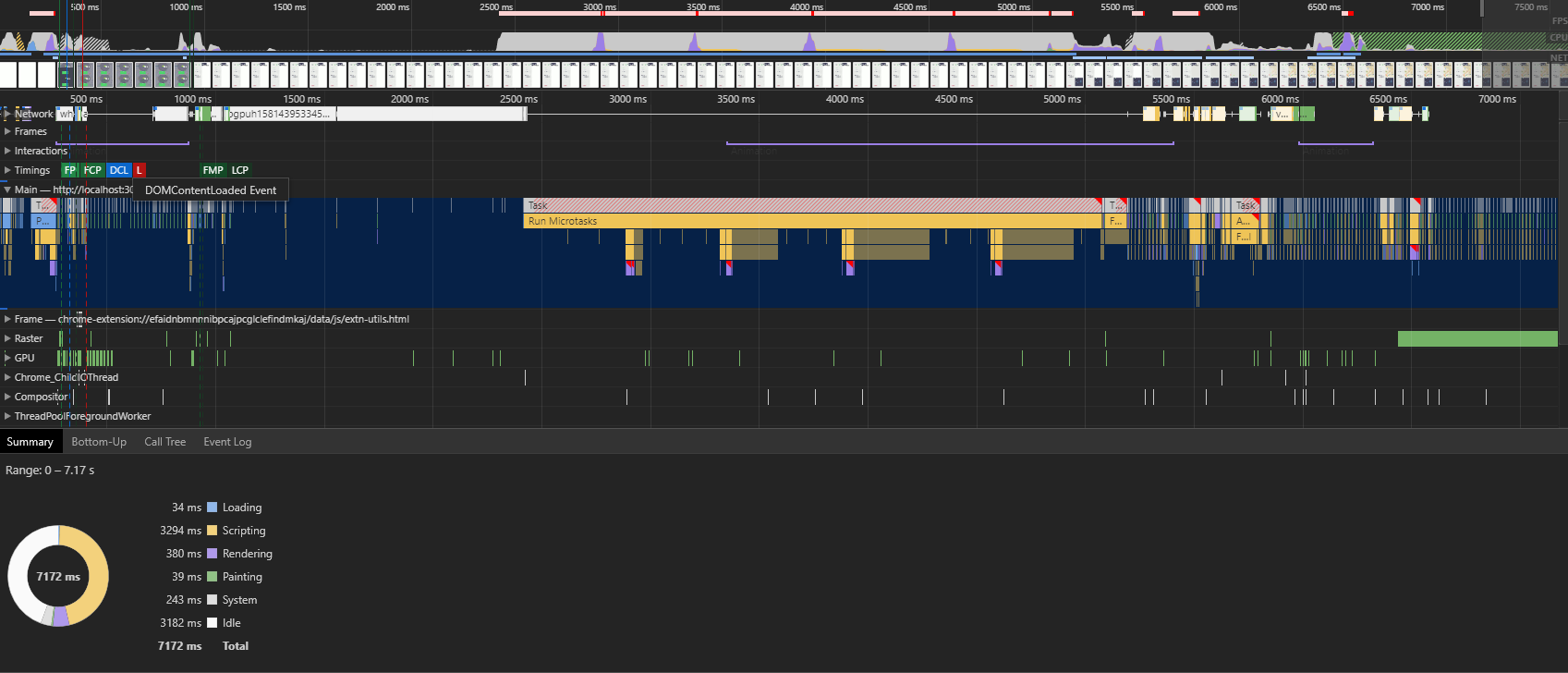
Din punct de vedere static, performanta pe care o are un utilizator in pagina noastra este descrisa prin puncte de tip LightHouse. Acesta este este un instrument automat și open-source pentru îmbunătățirea calității paginilor web. Puteți rula pe orice pagină web, publică sau care necesită autentificare. Are audituri pentru performanță, accesibilitate, aplicații web progresive, SEO și multe altele. Inainte de descrierea acestei lucrari, am rulat un test de perfomata folosind acest plugin, iar din rezultatele sale aplicatia se prezinta:



Grafica functionala:



Pentru integrarea aplicatiei in continuare voi respecta indicia pe care plug-in-ul LightHouse i-a recomandat. Ca si obiective propuse perfomanta ar trebui sa creasca pana la 80 de puncte folosind server-side rendering, iar accesibilitatea, care se refera la contrastul si integrarea culorilor in aplicatie, dar si potentialul dezvoltarii aplicatiei pentru persoane cu dizabilitati trebuie sa creasca la 90 de puncte. Cate din aspectele pe care LightHouse le-a definit:



În imaginea prezentata se observa calea de cascada pe care aplicația o prelucrează la introducerea și accesarea ei în browser. Se pare că primul conținut încărcat este destul de valos din punct de vedere timp al încărcării, acesta poate după chiar și o secunda, ceea ce este vizibil pentru utilizator. Primul conținut pe care îl încarca aplicația desigur este script-ul compilat de către React.JS. În viitor voi crea un sistem de funcții statice care sunt servite direct fară compilarea aplicației, cum ar fi funcțiile pentru schimbarea interfatei UI.

## Stilizari ale aplicatiei

Desigur ca prin CSS te poti distra si poti face obiecte sa arate asa cum iti doresti in pagina web, dar odata ce aplicatia este crestuta in complexitate avem nevoie de un manager de generare al acestui CSS. Aplicatia prezentata foloseste pre-procesatorul de stiluri SCSS, acesta fiind un sistem ce ne permite nesting-ul intre clasele declarate prin format CSS dar si folosirea de variabile pentru dimensiunea obiectelor sau culorile acestora.

### Strucutura B.E.M.

Este cunoscut faptul că ghidul de stil potrivit poate crește semnificativ viteza de dezvoltare, depanarea și implementarea de noi funcții în codul moștenire. Din păcate, majoritatea bazelor de cod CSS sunt uneori dezvoltate fără nicio structură sau convenții de denumire. Acest lucru duce la o codebase CSS de neînlocuit pe termen lung.

Abordarea BEM asigură că toată lumea care participă la dezvoltarea unui site web lucrează cu o singură bază de cod și vorbește aceeași limbă. Folosirea denumirilor corespunzătoare vă va pregăti pentru modificările de design ale site-ului.

Block - Încapsulează o entitate autonomă care este semnificativă de la sine.

Element – Reprezinta părți ale unui bloc și acestea nu au o semnificație autonomă. Orice element este legat semantic de blocul său.

Modifier - Steaguri pe blocuri sau elemente. Folosiți-le pentru a schimba aspectul, comportamentul sau starea.

Exemplu de structura BEM:

<div *className*="form-user">

                            <div *className*="form-user\_\_title">

                                User Login

                            </div>

                            <div *className*="form-user\_\_input-holder">

                                <div *className*="form-user\_\_input">

                                    <input *type*="email" *autoComplete*="off" *placeholder*="Enter your email" *onChange*={*this.*onChangeEmail} *value*={*this.state.*email} />

                                </div>

                                <div *className*="form-user\_\_input">

                                    <input *type*="password" *autoComplete*="off" *placeholder*="Your password goes here" *onChange*={*this.*onChangePassword} *value*={*this.state.*password} />

                                </div>

                                <div *className*="form-user\_\_input  form-user\_\_input--button">

                                    <button *className*="button-primary" *onClick*={() *=>* *this.*makeLogin()}>LOGIN</button>

                                </div>

                            </div>

                        </div>

Codul 17 Cod sursă pentru exemplificarea structurii BEM

### Schema pirmidala a stilurilor

### Variabilele generate prin pre-procesatorului SCSS

Folosind terminologia CSS cream in prima faza folosind functiile SCSS grid-systemul pentru un design responsive si inovativ.

*@mixin* create-mq(*$breakpoint*, *$min-or-max*) {

*@if*(*$breakpoint* == 0) {

*@content*;

    } *@else*{

*@media* *screen* and (#{$min-or-max}-width: *$breakpoint* \*1px) {

*@content*;

        }

    }

}

*@mixin* create-col-classes(*$modifier*, *$grid-cols*, *$breakpoint*) {

*@include* create-mq(*$breakpoint*, 'min') {

        &*\_\_col#{$modifier}-offset-0* {

            margin-left: 0;

        }

*@for* *$i* *from* 1 *through* *$grid-cols* {

            &*\_\_col#{$modifier}-#{$i}* {

                flex-basis: (100 / (*$grid-cols* / *$i*) ) \* 1%;

            }

            &*\_\_col#{$modifier}-offset-#{$i}* {

                margin-left: (100 / (*$grid-cols* / *$i*) ) \* 1%;

            }

        }

    }

}

Codul 17 Cod sursă pentru generarea de grid-system folosind SCSS

Variabilele folosite pentru acest sistem de generare a elementelor responsive sunt:

*$grid\_\_bp-sm*: 576;

*$grid\_\_bp-md*: 768;

*$grid\_\_bp-lg*: 992;

*$grid\_\_bp-xl*: 1200;

*$grid\_\_cols*: 12;

*$map-grid-props*: (

        '': 0,

        '-sm': *$grid\_\_bp-sm*,

        '-md': *$grid\_\_bp-md*,

        '-lg': *$grid\_\_bp-lg*,

        '-xl': *$grid\_\_bp-xl*

);

Codul 17 Cod sursă pentru variabile SCSS

Variabilele denumite prin denumirea „grid” sunt practic dimensiunile ecranului in care se afiseaza aplicatia noastra, urmand ca acesta sa fie mapate in array-ul SCSS $map-grid-props si folosite mai departe in functiile de generare descrise mai sus. Aceste silitzari sunt realizate prin proprietatile „flex” ale CSS-ului.

Deasemena in variabilele globale avem si definirea de padding-uri globale pentru a fi consistenti in toata aplicatia, font-family folosit in toate elementele si alte stilizati pentru componentele descrise mai sus.

De ce s-au folosit aceste variabile ? Deoarece putem avea mai multe palete de culoare si stilizare la un update viitor.

## Testari de unitate

Aceasta aplicatie fiind una de anvergura, trebuie mai intai testata inainte de publicare, deci vom aveam nevoie de testari automate pentru testarea interfetei utilizatorului, pentru a trimite in productia noastra elemente sau evenimente gresite din punct de vedere functionalitate.

De acesta testare se va ocupa modulul de testare React.JS bazat pe principii cat mai acesibile utilizatorului.

Functionalitate se afla in folder-ul parinte al aplicatiei si se poate accesa prin rularea comenzii “npm run test” ce va activa la randul ei fisierul de testare “App.test.js”

*import* React *from* 'react';

*import* { render } *from* '@testing-library/react';

*import* App *from* './App';

test('renders learn react link', () *=>* {

*const { getByText } = render(<App />)*;

*const linkElement = getByText(/learn react/i)*;

  expect(linkElement)*.*toBeInTheDocument();

});

Codul 18 Cod sursă pentru exemplificarea testarii aplicatiei

Exemplu descris mai sus are urmatoarea asteptare :

* Atunci cand aplicatia este functionala, se asteapta sa se randeze iar ulterior aceasta sa contina text-ul « learn react » care va trebui sa existe in document.

Daca aceasta nu este prezenta in document-ul nostru, adica pagina web, vom primi o eroare care ne spune ca avem o testare ca a facut rejectare, iar codul nostru din Front-End trebuie rescris.

# Implementarea Back-End

|  |
| --- |
| **Actualitatea si oportunitatea temei**  **Fonturile**  **Editarea primei pagini**  **Structurarea lucrării**  **Subcapitol**  **Alt subcapitol**  **Gestionarea figurilor, formulelor, tabelelor**  **Recomandări generale**  **Comenzi rapide**  **Citarea referințelor** |

## Module Specifice Server-ului

## Preluarea Datelor Pentru Indicii Inregistrati

## Preluarea Datelor Pentru Utilizatori

## Sistemul De Salvare Al Log-urilor

## Routearea Serverului

## Certificarea SSL

# DevOps

|  |
| --- |
| **Actualitatea si oportunitatea temei**  **Fonturile**  **Editarea primei pagini**  **Structurarea lucrării**  **Subcapitol**  **Alt subcapitol**  **Gestionarea figurilor, formulelor, tabelelor**  **Recomandări generale**  **Comenzi rapide**  **Citarea referințelor** |

## Integrare Continua

## Livrare Contiuna

# Abstract

(short description, in english, of main aspects treated in the project)