Cuprins:

[GENERALITĂȚI 3](#_Toc44451116)

[CAPITOLUL 1 - Introducere 4](#_Toc44451117)

[1.1 Tema proiectului 4](#_Toc44451118)

[CAPITOLUL 2 – Instrumente de dezvoltare 6](#_Toc44451119)

[2.1 – Internetul 6](#_Toc44451120)

[2.2 – Pagini Web, Site-uri Web și Servere Web 7](#_Toc44451121)

[2.3 – Pagini Web Statice și Dinamice 9](#_Toc44451122)

[2.3.1 – HTML 9](#_Toc44451123)

[2.3.2. – CSS 9](#_Toc44451124)

[2.4 – JavaScript 10](#_Toc44451125)

[2.5 – ReactJS 11](#_Toc44451126)

[2.6 – BootStrap 12](#_Toc44451127)

[2.7 – Baze de date 13](#_Toc44451128)

[2.8 – SQL 13](#_Toc44451129)

[2.8.1 – Instrucțiuni SQL de definire a datelor 14](#_Toc44451130)

[2.8.2 – Instrucțiuni SQL de manipulare a datelor 14](#_Toc44451131)

[2.9 – MySQL 15](#_Toc44451132)

[2.10 – XAMPP 16](#_Toc44451133)

[2.11 – Visual Studio Code 16](#_Toc44451134)

[CAPITOLUL 3 – Specificațiile aplicației 16](#_Toc44451135)

[3.1 – Pagina de start 17](#_Toc44451136)

[3.2 – Pagina *About* 19](#_Toc44451137)

[3.3 – Pagina *Resume* 19](#_Toc44451138)

[3.4 – Pagina *Contact* 20](#_Toc44451139)

[3.4 – Pagina *Portfolio* 21](#_Toc44451140)

[3.5 – Pagina Project Details 22](#_Toc44451141)

[3.6 – Pagina *Log-In* 24](#_Toc44451142)

[3.7 – Pagina *Add Project* 25](#_Toc44451143)

[3.8 – Pagina *Not Found* 26](#_Toc44451144)

[CAPITOLUL 4 – Implementarea aplicației 27](#_Toc44451145)

[4.1 – Crearea bazei de date 27](#_Toc44451146)

[4.3 – Inițializarea proiectului NodeJS, instalarea dependințelor 29](#_Toc44451147)

[4.3 – Inițializarea proiectului ReactJS, instalarea dependințelor 32](#_Toc44451148)

[CAPITOLUL 5 – Testarea aplicației 41](#_Toc44451149)

[CAPITOLUL 6 – Concluzii 41](#_Toc44451150)

[6.1 – Concluzii 41](#_Toc44451151)

[6.2 – Posibilități de dezvoltare 41](#_Toc44451152)

[BIBLIOGRAFIE 42](#_Toc44451153)

[Resurse online multimedia 42](#_Toc44451154)

# GENERALITĂȚI

**Capitolul 1 – Introducere** reprezintă partea introductivă a lucrării, prezentarea domeniului din care face parte proiectul, a temei propriu-zise cât și structurarea aplicației.

**Capitolul 2 – Instrumente de dezvoltare** reprezintă o scurtă introducere a principalelor noțiuni legate de Internet. Sunt explicate noțiuni ca Internet, protocoale, aplicații pentru internet, DNS, intranet și extranet, servere web și aplicații web, pagini web statice și dinamice, limbaje de markup, scripting și styling, precum HTML, JavaScript sau CSS. De asemeni sunt prezentate noțiunile de bază ale bazelor de date relaționale. Este prezentat modul de utilizare a MySQL, conectarea și deconectarea de la server, crearea și selectarea bazelor de date, crearea tabelelor și introducerea interogărilor.

**Capitolul 3 – Specificațiile aplicației** prezintă lucrarea. Prezentarea generală a aplicației, interfața cu utilizatorul și modalitatea de stocare a informațiilor și a datelor.

**Capitolul 4 – Implementarea și utilizarea aplicației** reprezintă descrierea aplicației. Implementarea acestui site a fost realizată folosind framework-ul ReactJS a limbajului de programare JavaScript datorită flexibilității și modularității acestuia și a faptului că este de tip open-source și poate fi folosit gratuit, fără restricții. De asemenea, a fost ales sistemul de gestiune a bazelor de date relaționale MySQL datorită faptului că și acesta poate fi folosit gratuit și este unul dintre cele mai folosite produse pentru accesarea și administrarea bazelor de date relaționale.

Interfața grafică a aplicației este construită pe baza unui template, pe care toate paginile site-ului îl folosesc pentru a păstra o imagine unitară a întregului site.

Pentru stilizarea modernă a diferitelor componente HTML a fost folosită librăria BootStrap, versiunea specială pentru ReactJS, iar pentru definirea aspectului final al aplicației s-a folosit un fișier CSS, acesta putând fi modificat, sau înlocuit în totalitate, iar întreg site-ul își va schimba automat înfățișarea.

Prin combinarea acestor trei produse, ReactJS, MySQL și BootStrap, aplicația realizată este una foarte flexibilă, putându-se adăuga, ulterior, noi funcționalități.

**Capitolul 5 – Concluzii** prezintă opinia mea personală despre aplicația realizată cât și posibilitățile ulterioare de dezvoltare a aplicației.

# CAPITOLUL 1 - Introducere

Se poate spune, pe drept cuvânt, că trăim într-o societate informatizată. În zilele noastre, întâlnim calculatoare peste tot, de la micul magazin din colț, care-și ține evidențele cu ajutorul unui calculator și până la ghișeul la care plătim abonamentele de telefonie sau taxele și impozitele, la examenele pentru permisul de conducere. Peste tot sunt calculatoare, legate de cele mai multe ori între ele, formând astfel rețele de calculatoare. Toate acestea se datorează faptului că ne dăm seama din ce în ce mai mult că sistemele de calcul, calculatoarele, ne ușurează atât munca cât și viața, prin apariția lor.

Să ne gândim puțin cât de mult s-a schimbat lumea de când au apărut calculatoarele. Înaintea apariției calculatoarelor, orice tip de evidență era înregistrată pe hârtie, scrise de mână sau, în cel mai bun caz, dactilografiate. Marele dezavantaj al acestei metode era că acele hârtii se deteriorau în timp, sau pur și simplu se pierdeau în arhive imense. Astfel, apariția calculatorului reprezenta o mare avansare în domeniu, dar tot odată și în alte domenii.

Oamenii nu mai erau nevoiți să-și țină evidențele pe hârtii, deoarece pur și simplu introduceau datele în calculator și le salvau pe discul magnetic al acestuia. Și așa, datele erau într-un loc sigur, securizat, unde nu se deteriorau, nici nu se pierdeau, iar accesul la ele îl aveau doar persoanele care erau desemnate pentru utilizare și stricta securitate a discului magnetic, sau, puteau fi puse la dispoziția altor persoane/instituții, prin intermediul Internetului.

## Tema proiectului

Omenirea, prin evoluția informaticii, care cu fiecare zi ce trece îmbunătățește viața omului, unde cei tineri devin bătrâni, tot așa este și cu calculatoarele, care atunci când au apărut erau în număr mic, dar acum, oriunde am merge, mereu o să vedem că oamenii lucrează pe calculatoare, tablete sau telefoane mobile, chiar dacă sunt modele mai vechi. În ziua de astăzi nu există instituție de stat sau companie privată care să nu fie dotată cu calculatoare. Sa ne gândim numai la bănci, service-uri auto sau hoteluri, toate folosesc calculatoare pentru a-și ține evidențele, să transfere date sau pur și simplu să caute și să acceseze informații de pe Internet.

Astfel, aplicația ce face subiectul prezentei lucrări, se dorește a fi colecția de proiecte care fac proba competențelor personale, păstrând și prezentând dovezi pertinente ale cunoștințelor și aptitudinilor dobândite în domeniul programării și dezvoltării aplicațiilor informatice pe ramura de FrontEnd. Astfel, prin intermediul internetului și a unui domeniu informatic, se pune la dispoziția oricărei persoane/entități interesate, Portofoliul – colecția de proiecte, fiecare dintre acestea putând fi rulat și testat individual.

# CAPITOLUL 2 – Instrumente de dezvoltare

În această lucrare va fi descris modul de implementare a unei aplicații WEB folosind unul din framework-urile limbajului de programare JavaScript, și anume ReactJS. În cadrul acestei aplicații se vor folosi fișiere de tip *JS*, *HTML* și *CSS* iar informațiile vor fi citite/salvate dintr-o baza de date de tip *MySQL*. Legătura cu baza de date se va realiza prin intermediul unor fișiere de tip *PHP*.

Aplicația va fi dezvoltată într-un mediu de lucru local, de tip LocalServer, iar la final va fi compilată varianta de producție și copiată pe un server real, putând fi accesată de orice persoana care cunoaște adresa URL și folosește un calculator, tabletă sau telefon mobil, ce are o conexiune la Internet.

## 2.1 – Internetul

Prin Internet înțelegem, de obicei, ansamblul rețelelor de calculatoare interconectate din întreaga lume. De fapt, unii susțin că Internetul nu este neapărat o rețea, ci mai degrabă un ansamblu de reguli – protocoale de comunicație – care permit transferul de date între diferite rețele de pe glob, implicit la informațiile stocate în calculatoarele acestora.

Totuși, Internetul este o gigantică rețea de calculatoare, mai precis o rețea de rețele de calculatoare, așa cum se poate vedea în figura-1. Unele dintre aceste calculatoare oferind o multitudine de servicii, în timp ce altele oferă o diversitate de informații pentru oamenii de pe întregul glob.

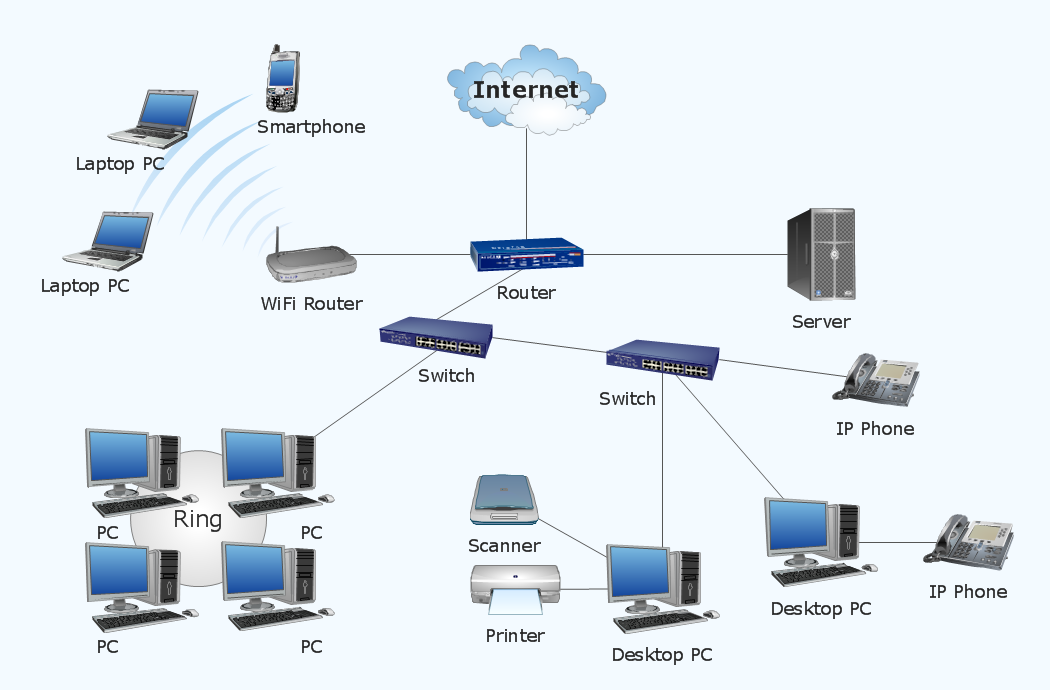


Figura 2.1 – Reprezentarea unei rețele de dispozitive, conectate la Internet

Accesul la Internet este modul de conectare în rețea, cât, mai ales, accesul la informația ce se vehiculează prin calculatoarele legate în rețea, posibilitatea de comunicație cu specialiști din toată lumea.

## 2.2 – Pagini Web, Site-uri Web și Servere Web

Construcția WWW (*World Wide Web*) este bazată pe protocolul numit HTTP/S (*HyperText Transfer Protocol/Secure*). Acesta este un protocol mic și rapid care se potrivește foarte bine sistemelor informatice multimedia.

Internetul reprezintă o mulțime globală de calculatoare interconectate, similar unei rețele locale, dar la o scară mult mai largă. Elementul primar al Internet-ului este WWW-ul, sau Web-ul, care este un suport pentru text, grafică, animație și sunet. Documentele destinate Web-ului sunt cunoscute sub numele de pagini Web.

Paginile Web, în sine, pot fi documente foarte complexe, interesante și atractive. Acestea sunt memorate pe HDD-ul (*Hard Disk Drive*) unui sever specializat, sunt gestionate printr-un software special, sunt regăsite și afișate prin intermediul navigatoarelor Web (browsere).

Browser-ul Web afișează paginile Web prin interpretarea unor marcatori, definiți cu ajutorul unui limbaj special numit HTML (*HyperText Markup Language*), utilizați pentru a codifica pagina de Web cu informația de afișat. Marcatorii au diferite semnificații. De exemplu aceștia semnifică modul în care vor fi așezate în pagină și afișate apoi pe ecran, diversele componente ale paginii, sau stabilesc legături între documente/fișiere.

Un Site Web este definit ca o colecție coerentă de informații, prezentată sub forma unor pagini Web, fișiere multimedia, documente, formulare, etc .., între care există legături bine definite.

Într-un Site Web bine proiectat, toate celelalte pagini punctează către pagina principală, prin Hypelink-uri, chiar dacă browser-ele au butoane de navigare speciale pentru acest lucru.

Atunci când se realizează un Site Web, aceste fișiere sunt păstrate, în mod uzual, într-un director, sau o colecție de directoare, în HDD-ul local iar acesta este cunoscut sub numele de site local (*Local Site*).

Prin publicarea unui site local (*Upload*), directorul principal, împreună cu conținutul acestuia, este transferat la serverul Web care conține un Software prin care site-ul este transmis către browser-ele Web ale calculatoarelor conectate la Internet și care solicită accesarea acelui site. Odată publicat, site-ul local se transformă în site Web, iar interacțiunea utilizatorului cu el are loc similar modului descris în figura 2.

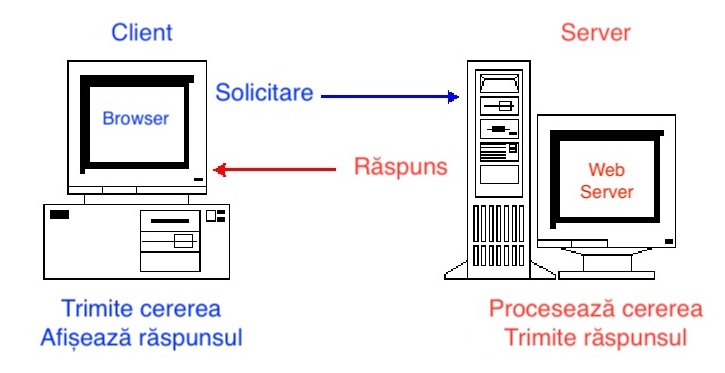


Figura 2.2 - Interacțiunea dintre Browser si Web Server

Un Server Web, este un program (*software*) care furnizează pagini Web la cerere. Când un utilizator, de la o adresă IP specifică, solicită un anumit fișier, serverul Web încearcă să obțină acel fișier și să-l trimită înapoi utilizatorului.

Fișierul solicitat poate fi codul sursă HTML al unei pagini Web, o imagine .GIF, un document .PDF sau un fișier multimedia audio tip .MP3 sau video tip .AVI. Browser-ul Web este cel care determină ceea ce trebuie cerut, nu serverul Web.

Conexiunile la serverul Web se stabilesc pe măsură ce sunt solicitate resurse; se transmite o cerere a unei pagini Web de la un server Web, o conexiune HTTP este stabilită prin intermediul Internet-ului între gazda solicitantă și gazda pe care rulează serverul Web. Pagina solicitată este transmisă prin acea conexiune, iar aceasta este întreruptă de îndată ce răspunsul este primit de către inițiator, în final pagina fiind interpretată și afișată solicitantului.

## 2.3 – Pagini Web Statice și Dinamice

### 2.3.1 – HTML

Paginile Web sunt fișiere cu simplu text folosind limbajul HTML. Acesta este implementat ca o mulțime de elemente (*Tag-uri*).

Tag-urile HTML sunt folosite de către autorii paginilor Web pentru a marca paginile de text, iar browser-ele le folosesc pentru a randa și afișa informațiile pentru vizualizarea de către utilizator. Marea majoritate a paginilor Web conțin hyperlink-uri către alte pagini Web.

Fiind un limbaj de mark-up, HMTL-ul permite crearea layout-ului paginilor și a formularelor, dar nimic mai mult, adică o pagina Web Statică.

### 2.3.2. – CSS

Majoritatea browser-elor noi permit folosirea și a altor tehnologii, dintre care cea mai cunoscută este CSS (*Cascading Styles Sheet*). Acest limbaj permite descrierea (*stilizarea*) unui document HTML, adică felul în care un anumit element trebuie să fie poziționat în pagină, culoarea textului, mărimea fontului, etc ....

De asemeni, prin CSS se pot crea anumite efecte de umbrire, mișcare și chiar animații ale unor elemente, ceea ce poate conferi puțin dinamism unei pagini Web, ce inițial a fost statică.

## 2.4 – JavaScript

Pentru a se construi interfețe intuitive, și sofisticate în același timp, dar mai ales dinamice, este necesar a se folosi și un limbaj de *scripting* la nivel de client. *Scripting-ul* permite scrierea de cod (mici programe) care rulează în cadrul browser-ului, direct pe calculatorul clientului.

Cel mai cunoscut limbaj de scripting pe parte de client este, fără nicio îndoială, JavaScript, limbaj ce este suportat, mai mult sau mai puțin, de aproape orice browser existent. Folosind JavaScript se pot realiza: animarea textului și a imaginilor, validarea formularelor, crearea de meniuri drop-down și a controalelor de navigare, se pot efectua procesări numerice de bază sau modificări asupra textelor. Pe scurt, JavaScript poate introduce multiple funcționalități în paginile Web.

JavaScript este un limbaj de programare orientat obiect bazat pe conceptul prototipurilor.

Scripting-ul permite programatorilor să detecteze și să proceseze *evenimentele*. Spre exemplu, o pagină care se încarcă, un formular trimis, mișcarea pointer-ului mouse-ului asupra unei imagini, toate acestea sunt evenimente, iar script-urile pot fi executate automat de browser atunci când acestea au loc.

Script-urile pot fi incluse în codul HTML, dar cel mai bine și corect, este ca acestea să fie stocate în fișiere externe și legate (*importate*) în interiorul codului HTML prin intermediul tag-ului <script></script>.

Spre deosebire de alte limbaje de script-ing, precum PHP (*PHP: Hypertext PreProcessor*) ce este executat pe serverul Web, JavaScript este executat direct în calculatorul clientului, în browser-ul acestuia.

Dar, JavaScript se poate folosi și pentru a rula scripturi în-afara browser-ului, cu ajutorul unui mediu de lucru, de tip Open-Source, numit NodeJS. Astfel a luat naștere o nouă paradigmă, anume ”JavaScript peste tot”, adică folosirea unui singur limbaj de programare pentru crearea scripturilor ce rulează atât în browser-ul clientului cât și pe Web server.

NodeJS, împreună cu Express permite crearea paginilor Web dinamice direct de pe Web server.

## 2.5 – ReactJS

ReactJS (React.js sau React) este o bibliotecă (*un framework*) JavaScript, creată special pentru construirea interfețelor utilizator. ReactJS este open-source și este menținut de Facebook, Instagram și o comunitate de mii de programatori și corporații.

ReactJS este eficient, flexibil și ne permite să construim componente încapsulate care se auto-gestionează, ca mai apoi să le putem combina pentru a face interfețe complexe.

Librăria este în prezent folosită în website-urile unor celebre companii precum Facebook, Netflix, Airbnb, DropBox sau Yahoo! Mail. Ecosistemul său bogat – incluzând instrumente precum Flux, Redux sau Node.js în backend și în multe librarii open-source de taskuri, cât și suportul său important din partea corporațiilor și a comunității în continuă mișcare, arată faptul că ReactJS este o tehnologie modernă pe care te poți baza în dezvoltarea de interfețe utilizator de ultimă generație.

De ce să alegem ReactJS ?

ReactJS ne permite să creăm repede o interfața utilizator scalabilă și ușor de utilizat pentru aplicațiile web. Este una dintre librăriile open-source cele mai populare în rândul programatorilor dar și al mediului de afaceri, mulțumită avantajelor sale în dezvoltarea aplicațiilor web:

* Permite dezvoltarea de aplicații la scară largă în scurt timp și la un nivel superior de calitate;
* Permite scrierea de cod curat, modular și reutilizabil;
* Aplicațiile web construite în ReactJS sunt flexibile, se pliază pe cerințele SEO (*Search Engine Optimization – Optimizarea pentru motoarele de căutare*), sunt ușor de scalat și menținut;
* Este ușor să se treacă la ReactNative și să se creeze aplicații mobile cu un aspect nativ;
* Este ideal pentru proiecte ce pot fi sparte în componente separate și mai apoi reutilizabile.

În mod normal, un site web va folosi fișierele HTML pentru a reactualiza DOM-ul (*Document Object Model*), proces ce face ca ”lucruri să se întâmple” fără a fi nevoie ca utilizatorul să facă reîncărcarea manuală la pagina web. Acest lucru este acceptabil pentru aplicații mici, dar pe măsură ce aplicația crește, odată cu aceasta crește și DOM-ul (reprezentarea structurii logice a documentului HTML sau XML precum este evidențiat și in Figura 3.) și de aici intervin timpii mari de încărcare, procesare și afișare a paginilor către utilizatori.

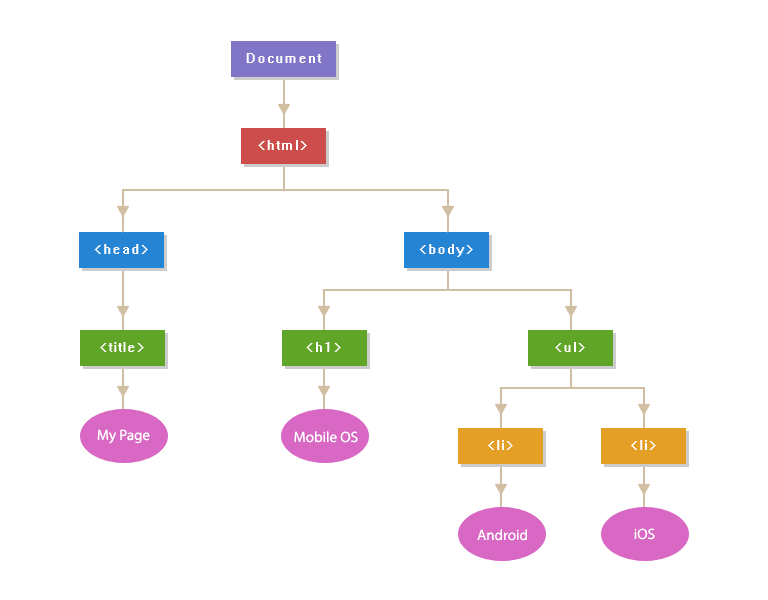


Figura 2.3 – Reprezentarea DOM a unui document HTML

În schimb, dacă programatorul va folosi librăria ReactJS, atunci aplicația web va crea un DOM virtual, care este o copie în memorie a DOM-ului, iar ReactJS se va folosi de acest DOM virtual pentru a urmări ce componentă sau componente a DOM-ului actual trebuie reîncărcate la apariția unui eveniment, spre exemplu când utilizatorul apasă pe un buton, sau duce pointerul mouse-ului deasupra unei imagini.

Acest mod de reîncărcare selectivă a elementelor utilizează mai puține resurse pentru calcul și se transpune direct în timpi de încărcare mai mici a paginii web.

## 2.6 – BootStrap

Bootstrap este un framework CSS gratuit și open-source, intuitiv, puternic, direcționat către dezvoltarea web front-end receptivă și mobilă. Conține elemente și șabloane de design reutilizabile, bazate pe HTML, CSS și JavaScript pentru tipografie, formulare, butoane, meniuri și alte componente de interfață.

## 2.7 – Baze de date

Bazele de date au devenit o componentă esențială a vieții de fiecare zi din societatea actuală. În cursul oricărei zile, fiecare dintre noi desfășurăm activități ce implică interacțiunea cu o bază de date, ca de exemplu depunerea sau extragerea unor sume de bani din bancă, rezervarea biletelor de avion sau a unei camere de hotel.

În sensul cel mai larg, *o bază de date* (*DataBase* sau *DB*) este o colecție de date corelate din punct de vedere logic, care reflectă un anumit aspect al lumii reale și este destinată unui anumit grup de utilizatori. O bază de date poate fi creată și menținută manual (de exemplu fișele de evidență a cărților dintr-o bibliotecă, așa cum erau folosite cu ani în urmă) sau computerizat, ceea ce se va realiza și în prezentul proiect.

O bază de date trebuie să asigure:

* *Abstractizarea datelor*, baza de date fiind un model al realității;
* *Integrarea datelor*, se referă la corectitudinea datelor încărcate și manipulate astfel încât să respecte restricțiile de integritate;
* *Securitatea datelor*, adică limitarea accesului la baza de date;
* *Partajarea datelor*, datele putând fi accesate de mai mulți utilizatori, eventual în același timp;
* *Independența datelor*, organizarea datelor să fie transparentă pentru utilizatori, modificările în baza de date sa nu afecteze funcționarea aplicațiilor.

## 2.8 – SQL

SQL (*Structured Query Language – Limbaj Structurat de Interogare*) este un limbaj de programare specific lucrului cu bazele de date. În timp, acesta a devenit un standard în domeniu (standardizat ANSI-ISO), fiind cel mai popular limbaj utilizat pentru crearea, modificarea, regăsirea și manipularea datelor de către Sistemele de Gestiune a Bazelor de Date relaționale.

Caracteristicile generale ale limbajului SQL sunt:

* acesta conține atât componenta de descriere a datelor, DDL (*Data Description Language – Limbaj de Descriere a Datelor*), cât și componenta de manipulare a datelor, DML (*Data Manipulation Language – Limbaj de Manipulare a Datelor*);

### 2.8.1 – Instrucțiuni SQL de definire a datelor

**Instrucțiunea de creare** a unui tabel (CREATE TABLE) definește atributele (coloanele tabelului), domeniile atributelor şi diferite constrângeri pe care datele înregistrate (valori ale atributelor) trebuie să le respecte pentru asigurarea integrităţii (corectitudinii) bazei de date. Sintaxa generală a acestei instrucţiuni este:

CREATE TABLE nume\_tabel (

coloana\_1 domeniu\_1,

coloana\_2 domeniu\_2,

...................,

coloana\_n domeniu\_n);

**Instrucțiunea de modificare** a unui tabel (ALTER TABLE) permite adăugarea sau ștergerea unor atribute, modificarea domeniilor unor atribute, precum şi adăugarea, modificarea sau ştergerea unor constrângeri ale tabelului.

Pentru adăugarea atributului DataAngajarii în tabelul ANGAJATI, se scrie în felul următor:

ALTER TABLE ANGAJATI ADD DataAngajării date;

Pentru ștergerea atributului Functie din tabelul ANGAJAT, se poate face cu comanda:

ALTER TABLE ANGAJAT DROP Functie;

**Instrucțiunea de ștergere** a unui tabel este:

DROP TABLE nume\_tabel;

### 2.8.2 – Instrucțiuni SQL de manipulare a datelor

**Instrucțiunea de selectare** SELECT reprezintă blocul de interogare de bază și ea selectează informațiile dorite din tabelele bazei de date. Aceasta este o instrucțiune foarte puternică și are următoarea sintaxă generala:

SELECT lista\_coloane FROM lista\_tabele [WHERE condiție];

Se remarcă astfel 3 secțiuni (clauze) importante ale construcției de interogare:

* + clauza SELECT: indică lista atributelor (coloanelor) unor tabele sau al expresiilor care vor fi selectate și afișate. Coloanele din listă trebuie să aparțină uneia din tabelele specificate în clauza FROM;
  + clauza FROM: indică tabelele asupra cărora se face interogarea;
  + clauza WHERE: este urmată de o condiție, dată ca o expresie booleană.

**Instrucțiunea de introducere** INSERT se folosește, după cum însuși sugerează, pentru introducerea liniilor într-un tabel și are următoarea sintaxă:

INSERT INTO nume\_tabel (coloana\_1, coloana\_2, ...)

VALUES (valoare\_1, valoare\_2, ...);

**Instrucțiunea de actualizare** UPDATE permite actualizarea valorilor coloanelor (atributelor) din una sau mai multe linii ale unui tabel, si are sintaxa de forma:

UPDATE nume\_tabel

SET coloana\_1 = expr\_1, coloana\_2 = expr\_2, ...

[WHERE condiție];

**Instrucțiunea de ștergere** DELETE permite ștergerea uneia sau mai multor linii ale unui tabel, și are următoarea sintaxă:

DELETE FROM nume\_tabel [WHERE condiție];

Din tabel sunt șterse acele linii care îndeplinesc condiția din clauza WHERE. Daca este omisă clauza WHERE, atunci vor fi șterse toate liniile din tabel.

## 2.9 – MySQL

MySQL este cel mai popular Sistem de Gestiune al Bazelor de Date relaționale, de tip Open-Source, la ora actuală, fiind o componentă cheie a stivei LAMP (Linux, Apache, MySQL, PHP).

Pentru a administra bazele de date MySQL se poate folosi modul linie de comandă (*CLI – Command Line Interface*) sau, prin descărcarea de pe Internet a unei interfețe grafice: MySQL Administrator si MySQL Query Browser. Un alt instrument de management al acestor baze de date este și aplicația SQL Manager.

MySQL poate fi rulat pe mai toate Sistemele de Operare: GNU/Linux, MacOSX, Windows, Solaris, NetBSD, SunOS, etc...

Printre caracteristicile de bază, se remarcă:

* funcționarea pe diferite platforme;
* complet multi-threaded folosind thread-uri de kernel, adică poate lucra cu ușurință pe mai multe procesoare, dacă acestea sunt disponibile;
* folosește tabele temporare stocate în memorie;
* serverul este disponibil ca program separat ce poate fi folosit într-un mediu de rețea de tip client/server. De asemenea, este disponibil și ca bibliotecă ce poate fi inclusă în aplicații de sine stătătoare.

### 2.10 – XAMPP

Pentru stocarea informațiilor introduse în baza de date se va folosi XAMPP, care este un pachet de aplicații ce constituie infrastructura software necesară găzduirii site-urilor web: server de web (*Apache*), server de date (*MySQL*) și interpretoare pentru scripturi PHP.

XAMPP a fost creat însă pentru a pune la dispoziția programatorilor un instrument eficient de testate a diferitelor aplicații în curs de dezvoltare. Odată instalat pe calculatorul propriu, pachetul de aplicații va face ca acesta să aibă comportamentul unui server, permițând astfel testarea aplicațiilor scrise fără a intra în conflict cu firma care va găzdui în final aplicația software realizată.

### 2.11 – Visual Studio Code

Visual Studio Code (*VS-Code*) este un editor gratuit, de tip open-source, dezvoltat de către Microsoft pentru platformele Windows, MacOSX și Linux.

VS-Code are integrat propriul *Terminal*, un sistem de management al versiunilor ce suportă Git, cât și un robust sistem de auto-completare, de tip *InteliSense*. De asemeni, are disponibile peste 10.000 de extensii ce pot fi descărcate și instalate, ceea ce îl face extrem de personalizabil, astfel că fiecare programator și-l poate adapta propriilor nevoi.

Într-un studiu efectuat în 2019, peste 50% dintre respondenți au afirmat că folosesc VS-Code ca principal editor pentru programare.

# CAPITOLUL 3 – Specificațiile aplicației

Proiectul propus pentru a fi dezvoltat în această lucrare este o aplicație web și voi folosi VS-Code, unde voi inițializa un proiect ReactJS.

În cadrul acestui proiect voi folosi librarii externe precum *React-Bootstrap*, *React-Router*, *MDBReact*, *React-Date-Picker*, ...etc. Aceste librarii se vor instala prin intermediul CLI și au rolul de a facilita navigarea între paginile aplicației, de a conferi proprietăți de adaptabilitate (*Responsive)* paginilor, în funcție de dispozitivul și de rezoluția ecranului de pe care este accesată aplicația, de a integra diverse imagini pre-stilizate.

Baza de date este creată și pusă la dispoziția unui server NodeJS prin intermediul aplicației MySQLWorkbench. La final doresc ca baza de date să fie transferată pe serverul domeniului personal, împreună cu varianta *Production Build* a aplicației.

Încă de la început, proiectului i-a fost creat un *repository* pe platforma GitHub, <https://github.com/horoiu/Portfolio-Licenta>, unde pot fi observați toți pașii din procesul de dezvoltare a componentelor aplicației. În prezent există un număr de peste 50 *commit-uri*.

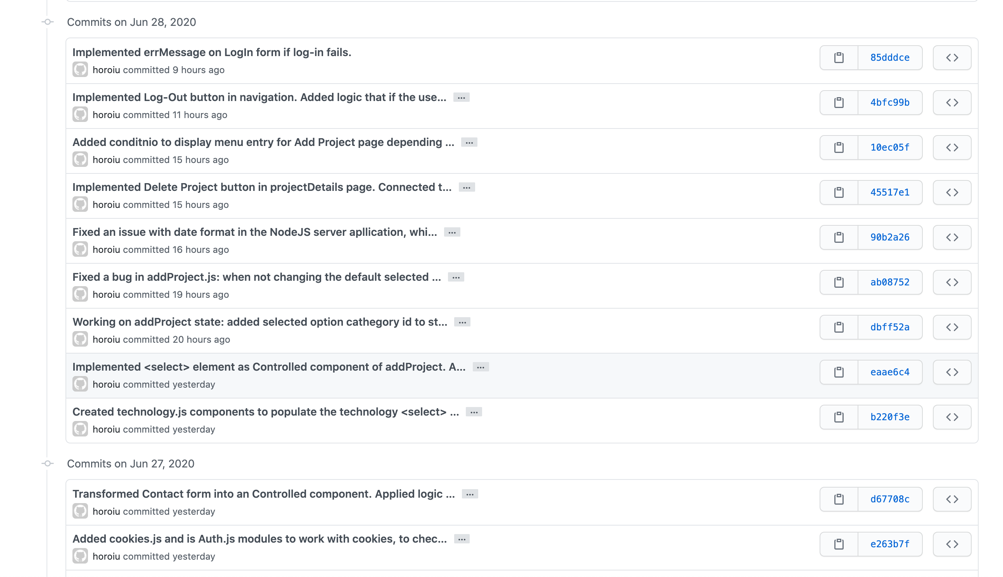


Figura 3.1 – Commit-uri în repository-ul de *GitHub*

## 3.1 – Pagina de start

Pagina de start este una simplă fără prea multe elemente, dar cu un titlu succint. Se observă prezența unui *Header* ce conține meniului de navigație, în partea de sus a paginii, cât și a unui *Footer* în subsolul paginii.

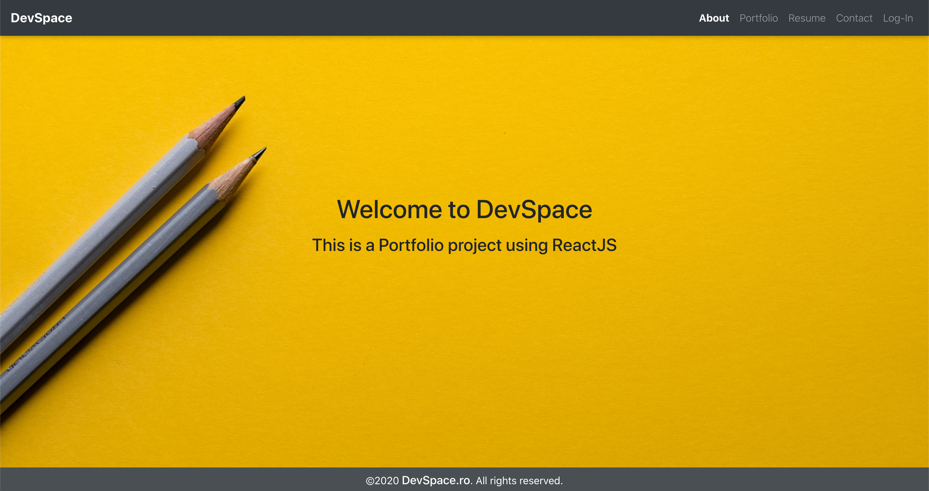


Figura 3.2 – Pagina de start a aplicației

Majoritatea elementelor din acest proiect sunt stilizate cu clase din librăria Bootstrap, sau chiar au fost preluate unele blocuri de elemente din aceeași librărie sau din MDB-React, pentru a conferi aplicației proprietăți *responsive.*

Bara de navigație *(meniul)* are în primă fază, când utilizatorul nu este autentificat, următoarea structură:

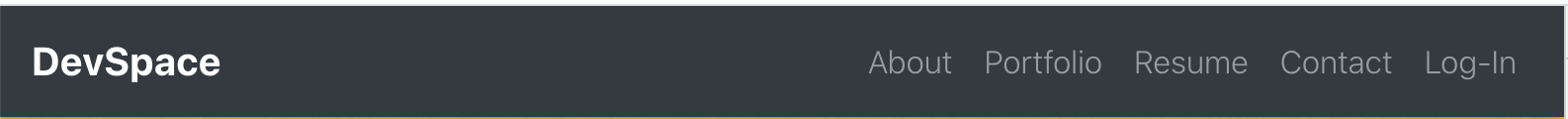


Figura 3.3 – Meniul de navigație, cu utilizatorul neautentificat

urmând ca aceasta să se modifice în mod dinamic, prin intermediul unui *cookie* creat la momentul realizării cu succes a operației de autentificare, și a unei funcții de verificare a autentificării isAuth(). Acestea vor fi explicate mai în detaliu în secțiunea dedicată dezvoltării aplicației.

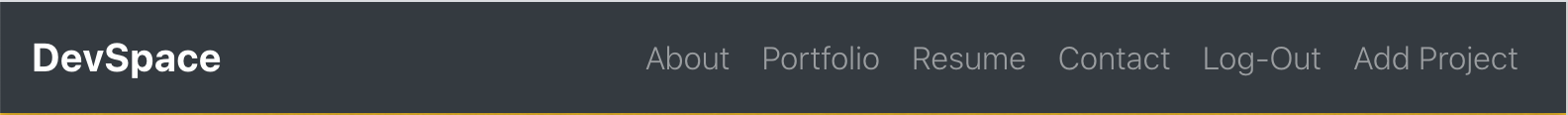


Figura 3.4 – Meniul de navigație, cu utilizatorul autentificat

Meniul este *responsive*, astfel că pe ecrane cu o rezoluție mai mică acesta se transformă într-un *drop-down* *menu*:

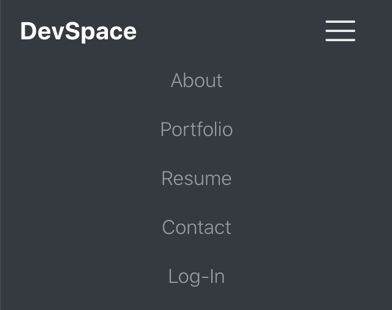
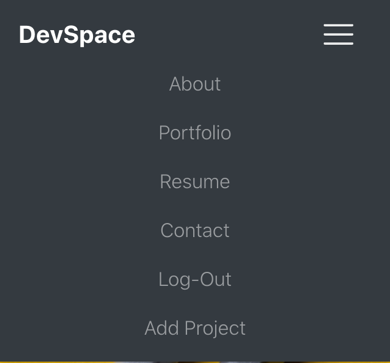
 

Figura 3.5 & 3.6 – Meniul de navigație în modul *drop-down*

## 3.2 – Pagina *About*

Pagina *About* nu este încă pe deplin configurată.

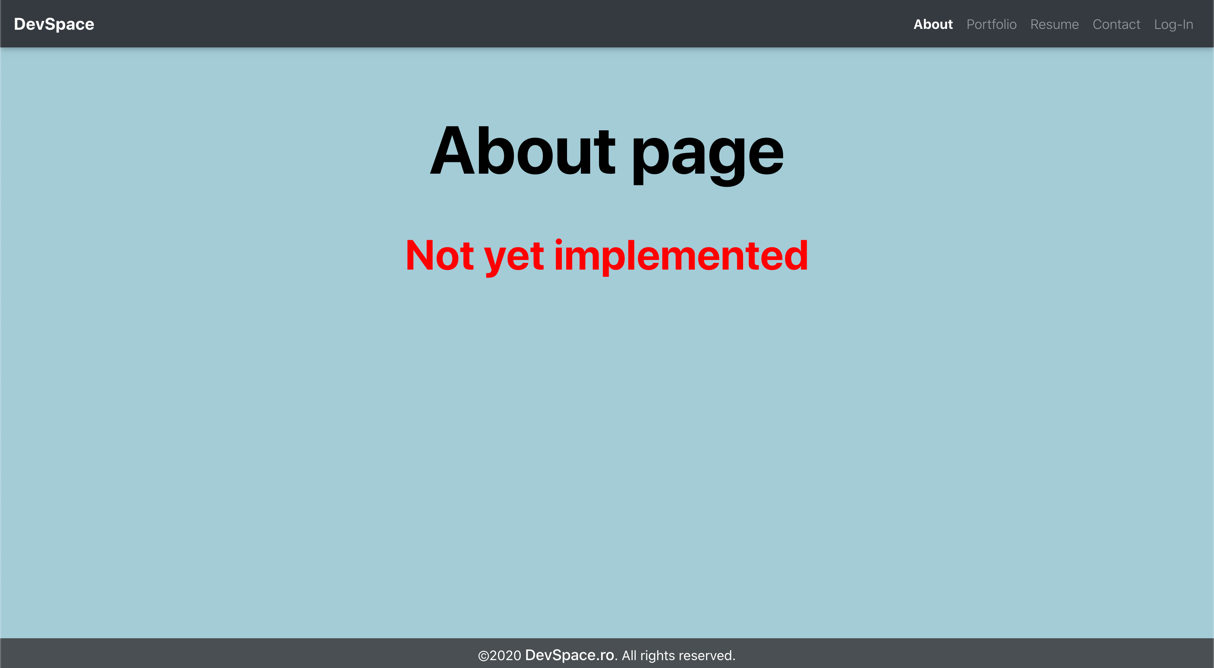


Figura 3.7 – Pagina *About*, momentan ne-implementată

## 3.3 – Pagina *Resume*

În pagina *Resume* este prezentat un document de tip *Curriculum Vitae* în format \*.pdf. Pentru facilitarea integrării documentului în structura paginii, am instalat componenta React-Pdf.

În pagină sunt disponibile 2 butoane, cu săgeți, pentru a facilita navigarea între paginile documentului. Cum în prezenta lucrare documentul \*.pdf are o singură pagină, codul celor două butoane va fi *comentat*, astfel că acestea nu for fi vizibile în varianta finală a aplicației, dar se pot activa pentru un viitor proiect.



Figura 3.8 – Pagina *Resume*

## 3.4 – Pagina *Contact*

Pagina *Contact* este un formular ce se dorește a fi un canal de comunicare pus la dispoziția utilizatorului pentru a putea trimite un mesaj către administratorul domeniului. În aceeași pagină a fost integrată și o hartă *GoogleMaps*, cu o locație prestabilită.

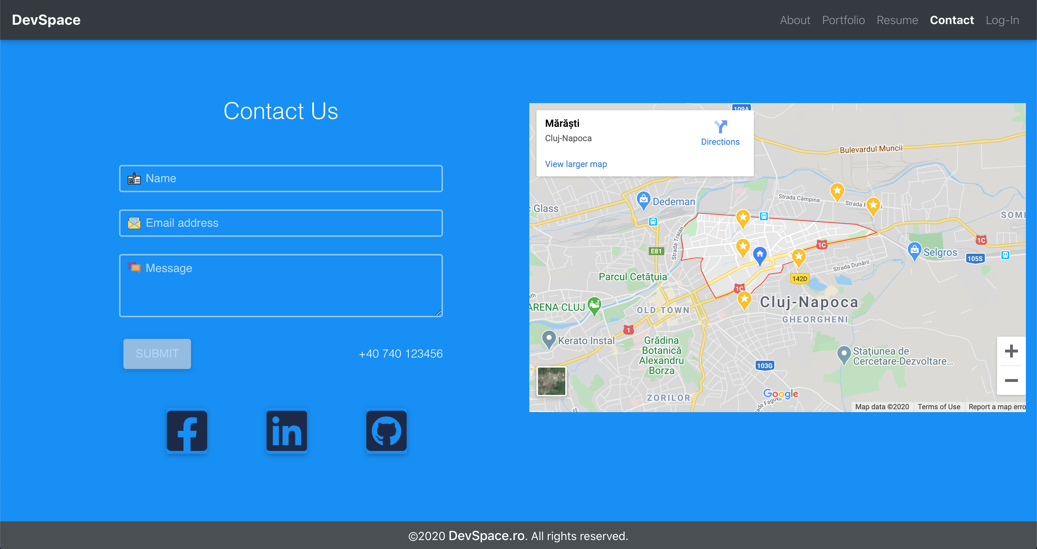


Figura 3.9 – Pagina de *Contact*

Formularul de contact este o componentă React de tip *Controlled Component*, cenu este încă conectat la un serviciu de mesaje, astfel că la apăsarea butonului ***Submit*** nu se întâmplă nimic. În schimb, butonul nu este accesibil dacă nu sunt completate toate câmpurile, iar pe câmpul de email există o validare suplimentară pentru verificarea ca adresa de email introdusă să respecte o structură specifică.

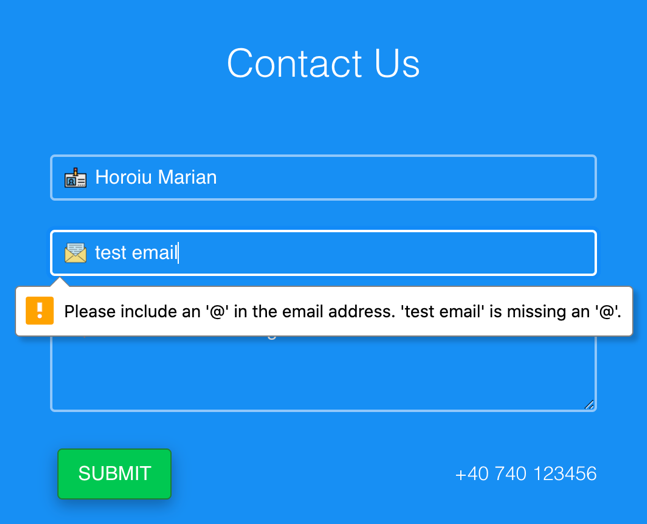


Figura 3.10 – Validare câmp e-mail

Tot în pagina de Contact mai sunt prezente câteva pictograme, încărcate dinamic dintr-un fișier extern ce conține un obiect JSON (*JavaScrip Object Notation*), ce reprezintă legături externe către paginile unor rețele de socializare / platforme specifice domeniului de activitate al fiecărui utilizator.

## 3.4 – Pagina *Portfolio*

Aceasta este pagina principală a aplicației. Aici sunt prezentate, sub formă de carduri, proiectele existente în baza de date și pe care utilizatorul le pune la dispoziția vizitatorilor.

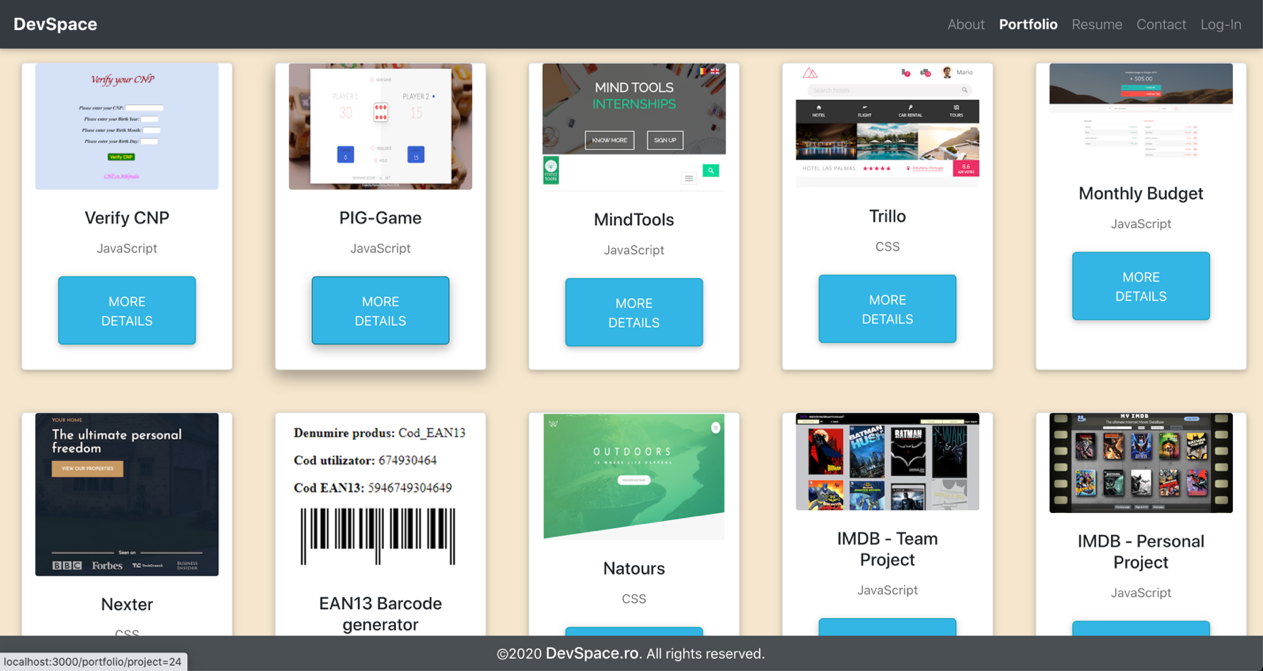


Figura 3.11 – Pagina *Portfolio*

Fiecare card are încorporată o imagine, un titlul, tehnologia de bază folosită la elaborare, cât și un buton ce ne direcționează către o noua pagină în care proiectul este detaliat.

În această pagină nu este nicio diferență evidentă funcție de starea de autentificare a utilizatorului în aplicație.

## 3.5 – Pagina Project Details

Aceasta nu este o pagină de sine stătătoare, în consecință nu are intrare in meniul de navigație. Aceasta este o componentă a paginii *Portfolio* populată în mod dinamic cu datele specifice cardului cu proiectul de pe care s-a apăsat butonul *”More Details”*. După cum se poate observa și în bara de navigație din browser, adresa este specifică fiecărui proiect in parte.

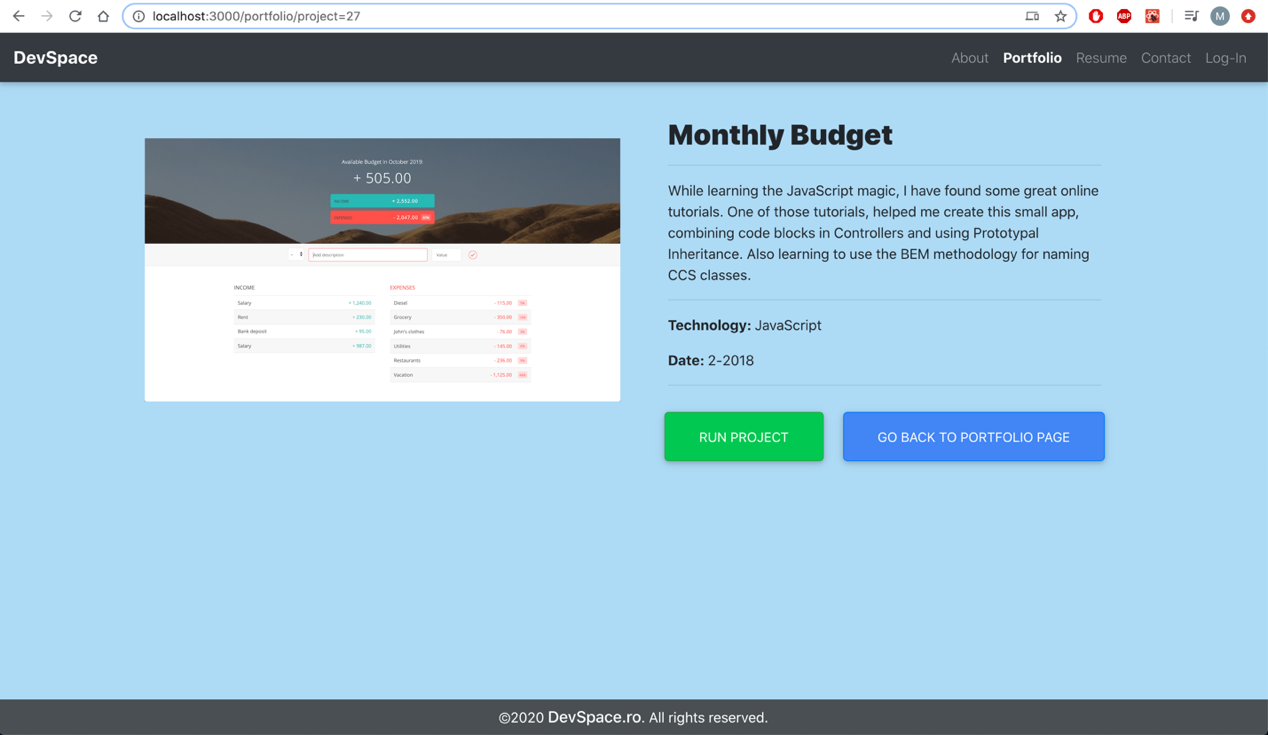


Figura 3.12 – Pagina *Project Details,* specificăproiectului cu ID=27, utilizator ***neautentificat***

Este afișată o noua poză a proiectului, data realizării cât și o scurtă descriere a acestuia.

Din acest punct, aplicația începe să facă diferența dintre utilizatorul autentificat și cel neautentificat. În figura 3.10, se pot observa doar două butoane, cu titluri sugestive: *”Run Project”* și *”Go Back To Portfolio Page”*. Dar odată ce ne ne-am autentificat, devine vizibil un al trei-lea buton, și anume cel de *”Delete Project”*.

Toate cele 3 butoane sunt complet implementate și funcționale.

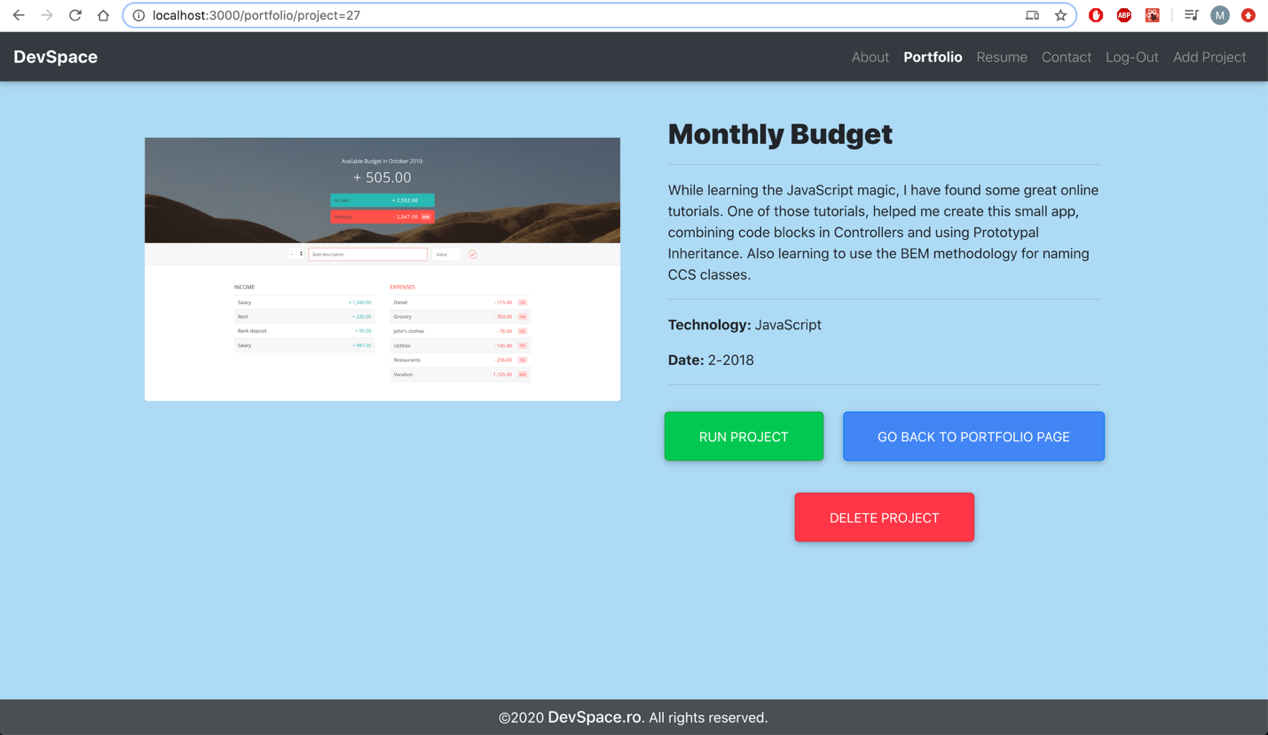


Figura 3.13 – Pagina *Project Details,* specificăproiectului cu ID=27, utilizator ***autentificat***

De asemeni, odată autentificat, se observă modificarea meniului de navigare.

## 3.6 – Pagina *Log-In*

De aici utilizatorul se poate autentifica în aplicație. Se solicită introducerea unui nume și a unei parole, ce vor fi verificate în baza de date printr-un *request HTML* de tip POST. Requestul va fi făcut către serverul de *back-end* creat cu Express în mediul de lucru NodeJS.

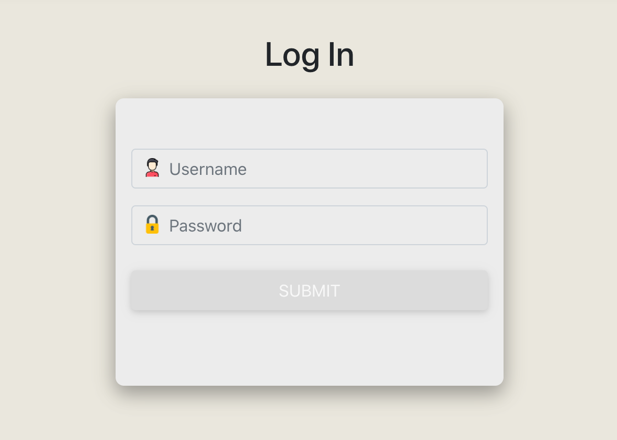
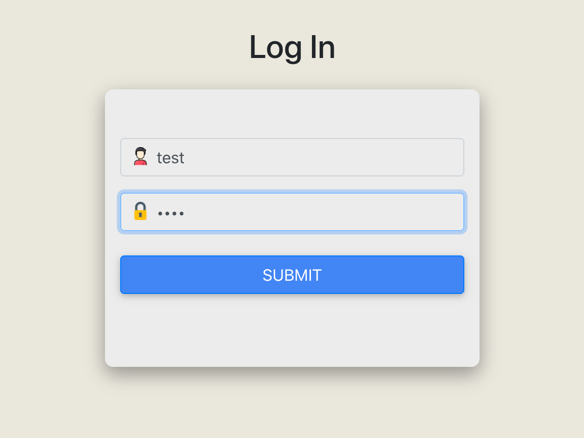
 

Figura 3.14 – *Log-In* Figura 3.15 – *Log-In* cu câmpurile completate

După cum se poate observa din figurile 3.14 și 3.15, butonul de *”Submit”* este activat doar după completarea ambelor câmpuri de *Input*.

Dacă se introduc valori ce nu sunt validate în baza de date, utilizatorul primește un mesaj de informare, iar câmpurile formularului sunt golite.

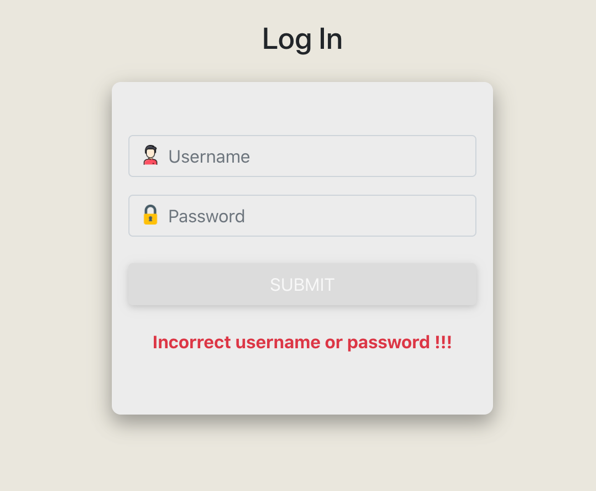


Figura 3.16 – *Log-In* nevalidat

## 3.7 – Pagina *Add Project*

Odată autentificat, utilizatorului i se oferă, în bara de meniu, un nou buton, care îl redirecționează către pagina de *Add Project*. Această pagină conține un formular în care se pot introduce datele pentru crearea unei noi intrări în baza de date, adică salvarea unui nou proiect.

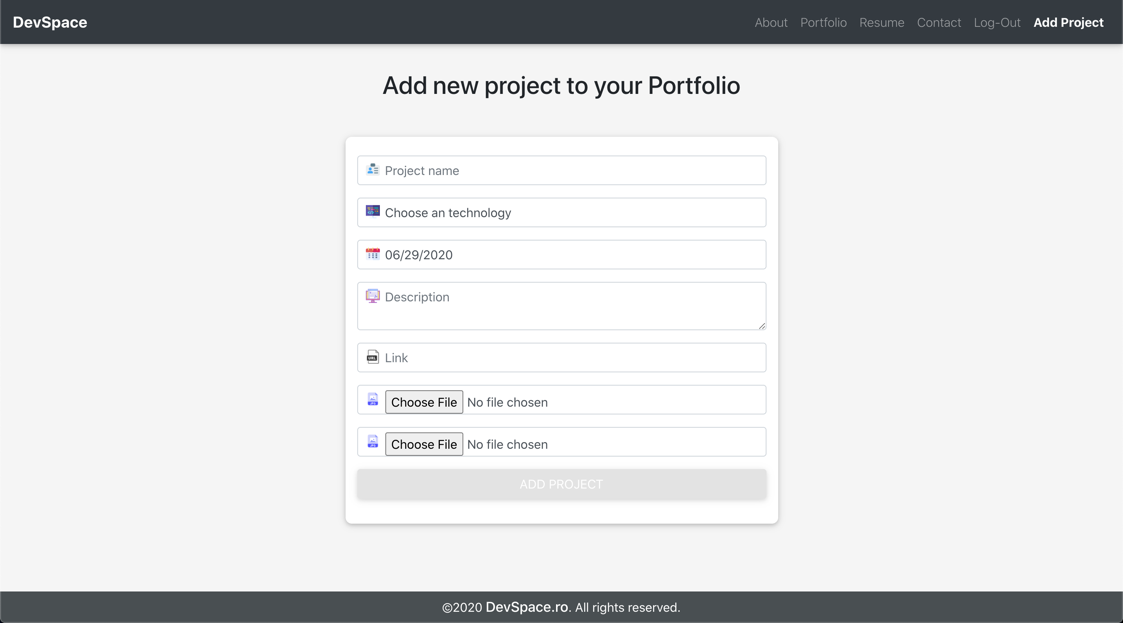


Figura 3.17 – Pagina *Add Project*

Ca și la celelalte formulare, butonul de *”Add Project”* nu va fi activat decât după introducerea valorilor în toate câmpurile.

În cadrul acestui formular vom regăsi următoarele tipuri de câmpuri pentru introducerea datelor: *Input text*; *Select & Option*; *Date picker*; *Input text-area* și  *Input file*.

Câmpul pentru selectarea tehnologiei este unul de tip *Select & Option*, populat în mod dinamic cu valori din baza de date, tabelul *categorie*, această operațiune executându-se la fiecare accesare a paginii, prin efectuarea interogării propriu-zise a bazei de date.

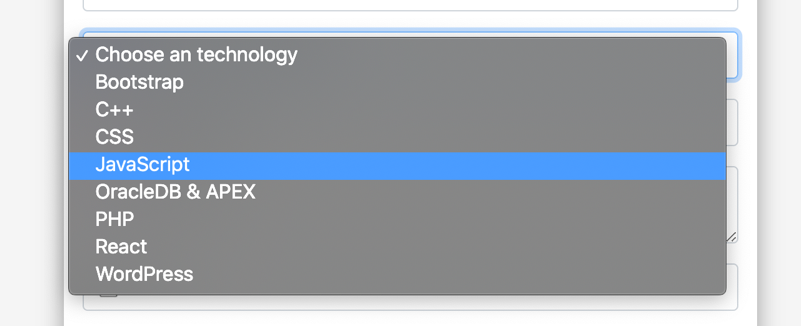


Figura 3.18 – Structura *Select* & *Option*

Valoarea datei se poate selecta prin intermediul câmpului de tip *Date-Picker*. Această opțiune nu este disponibilă în mod nativ în HTML sau React. A fost necesară instalarea unei componente externe.

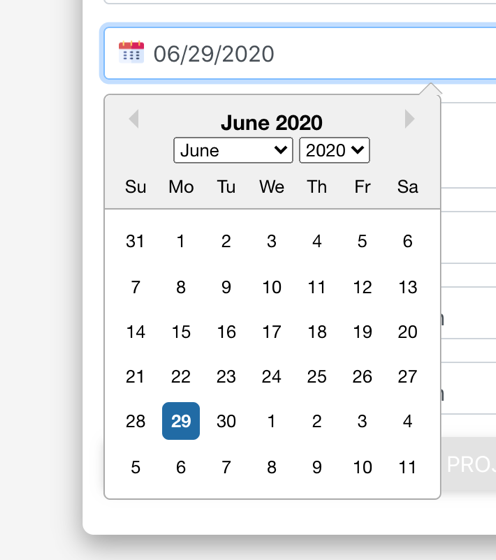


Figura 3.19 – Date-Picker

## 3.8 – Pagina *Not Found*

Dacă utilizatorul introduce manual adresa unei pagini ce nu există în aplicație, sau dacă, dintr-o eroare, este direcționat la adresa unei pagini inexistente, atunci se face automat redirecționarea către pagina *Not Found* unde se oferă posibilitatea navigării către pagina de *Start* a aplicației.

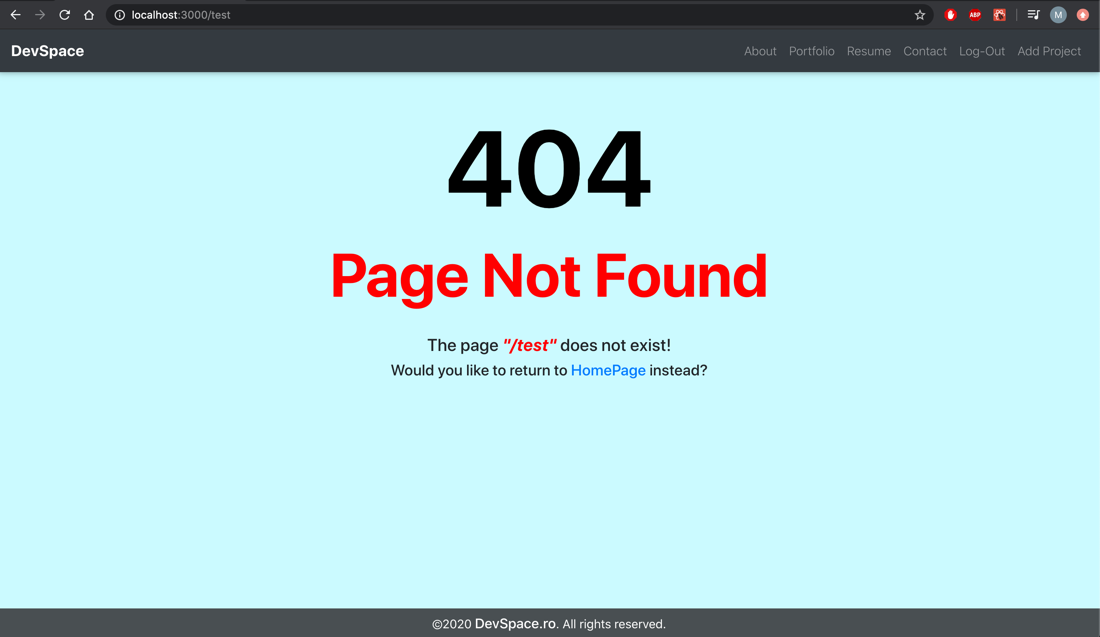


Figura 3.20 – Pagina *Not Found*

# CAPITOLUL 4 – Implementarea aplicației

Dezvoltarea aplicației web se va realiza în sistemul de operare Macintosh OS Catalina v10.15.5, astfel că toate celelalte aplicații și pachete de programe necesare, vor fi specifice acestui sistem de operare.

Editorul de text folosit este Visual Studio Code ver. 1.46.1 iar browser-ul principal utilizat este Google Chrome ver. 83.0.4103.97 (Official Build) (64-bit), întrucât acesta oferă un set foarte bun de unelte pentru dezvoltatorii software.

Baza de date va fi creată și pusă la dispoziția aplicației de către utilitarul MySQL-Workbench, prin intermediul unui server Express dezvoltat în mediul NodeJS.

## 4.1 – Crearea bazei de date

Voi folosi o bază de date denumită *portfolio* ce va conține 3 tabele: *admin, categorie* și *proiect*. Aceasta va fi creată pas cu pas, după cum urmează:

* se creează baza de date:

CREATE SCHEMA `portfolio` DEFAULT CHARACTER SET utf8 ;

* se creează tabelul a*dmin*, ce conține 3 coloane: *id\_admin*, *user* și *password*

CREATE TABLE `portfolio`.`admin` (

`id\_admin` INT NOT NULL AUTO\_INCREMENT,

`user` VARCHAR(40) NOT NULL,

`password` VARCHAR(40) NOT NULL,

PRIMARY KEY (`id\_admin`));

* în mod oarecum asemănător, se vor crea și celelalte 2 tabele.

Diferența va fi că în tabelul *proiect* vom avea o *cheie primară* iar în tabelul *categorie* vom avea o *cheie străină*, între cele două fiind făcută o relație de *constrângere*. Se implementează această legătură, întrucât doresc ca pentru fiecare proiect ce se va adăuga în baza de date, să se poată selecta dintr-o listă de categorii, tehnologia folosită.

În final, baza de date va arăta conform imaginii:

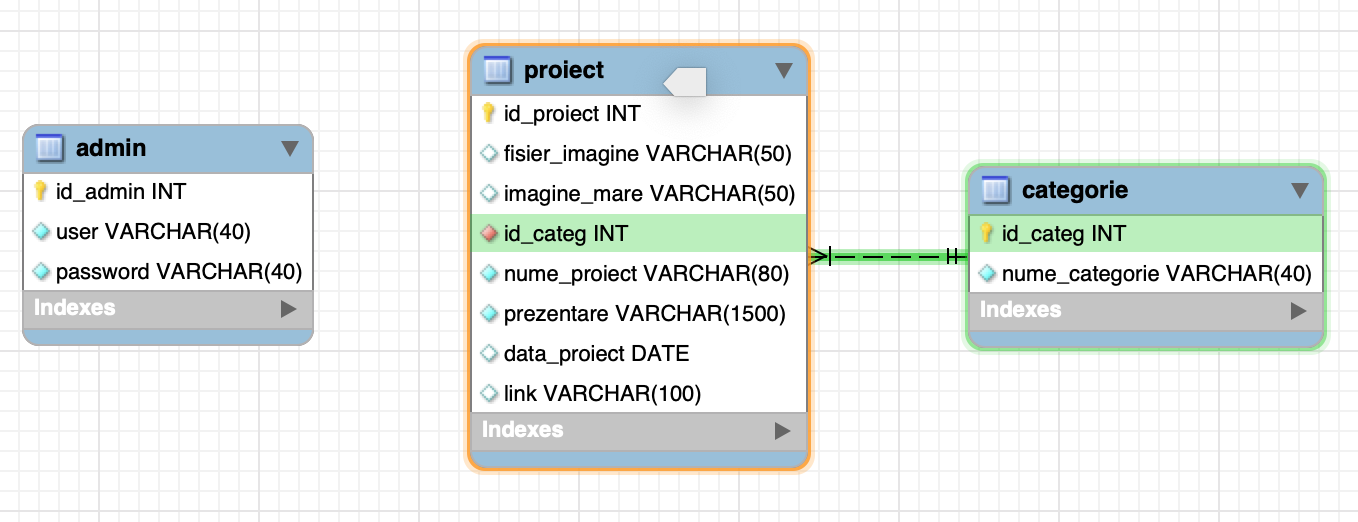


Figura 4.1 – Structura bazei de date văzută in MySQL Workbench

După adăugarea în mod manual a unor date în fiecare tabel, o simplă interogare a tabelei *proiect* ne va transmite datele conform imaginii din Figura 4.2.

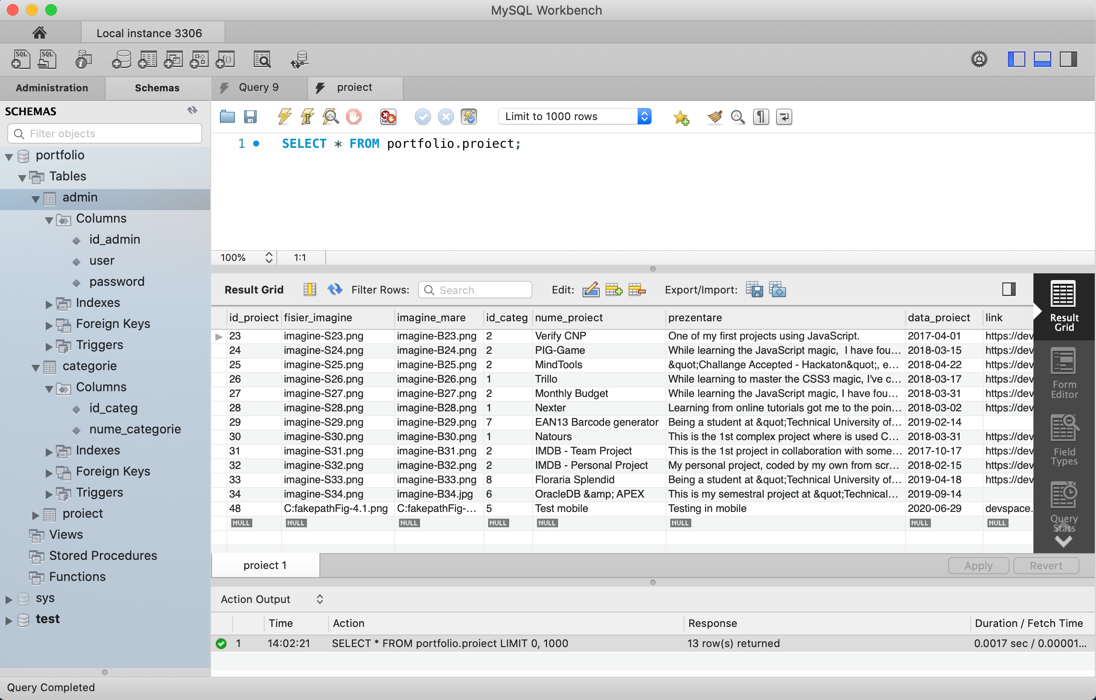


Figura 4.2 – Interfața *MySQLWorkbench*

## 4.3 – Inițializarea proiectului NodeJS, instalarea dependințelor

Se creează pe desktop un folder nou cu numele *Portfolio*. Navigăm în interiorul acestui folder și creăm un nou folder, numit *server-app.* Vom deschide acest folder cu VS-Code, iar din *Terminal* vom lansa succesiv comenzile :

touch package.json – crearea unui fișier package.json;

npm init -y – inițializarea fișierului package.json;

npm install express --save – instalarea librăriei Express, pentru a putea crea un server sau API;

npm install nodemon --save – instalarea dependinței ce va monitoriza modificările din scriptul *index.js* și va face reîncărcarea automată;

npm install mysql --save – instalarea dependinței ce va facilita legătura cu baza de date de tip MySQL;

npm install cors --save – instalarea dependinței ce va permite requesturile de tip ”cross-origin” făcute către server;

npm install body-parser --save – instalarea dependinței ce va facilita efectuarea request-urilor de tip POST;

touch index.js – crearea unui fișier index.js;

Din acest moment, se poate configura scriptul *index.js* ce va crea un API (*Access Point Interface*) prin care se va pune la dispoziția aplicației din ReactJS a informațiilor din baza de date. Vom importa toate librăriile și dependințele instalate și vom crea conexiunea la baza de date:

**const** express = require("express");

**const** bodyParser = require("body-parser");

**const** cors = require("cors");

**const** mysql = require("mysql");

**const** app = express();

**const** connection = mysql.createConnection({

host: "localhost",

user: "root",

password: "password",

database: "portfolio",

});

connection.connect((err) **=>** {

if (err) {

console.log("ERROR while connecting to DB: ", err);

return err;

}

});

// codul pentru definirea API-urilor se va scrie aici

app.listen(4000, () **=>** {

console.log(`NodeJS Server listening on port 4000`);

});

Din acest moment, serverul poate fi pornit din *Terminal* cu comanda: nodemon index.js și vom observa că acesta deja funcționează:

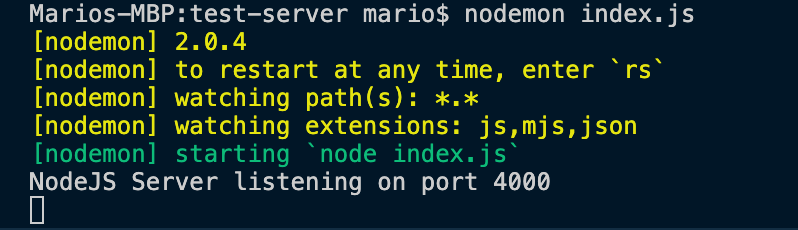


Figura 4.3 – Terminalul VS-Code

Dacă declarăm pentru un request de tip GET pentru ruta *“/”*:

*app*.*get*("/", (req, res) *=>* {

*res*.*send*("Hello from the MainPage of NodeJS server");

});

atunci, la accesarea din browser a url-ului *“localhost:4000”*, vom observa că API-ul funcționează corespunzător.

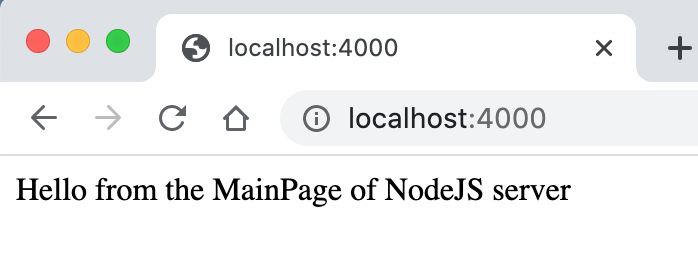


Figura 4.4 – API funcțional în browser

În mod similar, voi declara toate rutele de care voi avea nevoie în acest proiect. Spre exemplu, pentru a obține lista cu tehnologiile disponibile din tabelul *categorie*, voi declara ruta "/technology" pentru un request de tip GET, unde voi integra și comanda de interogare a bazei de date, după cum urmează:

*app*.*get*("/technology", (req, res) *=>* {

*const SELECT\_ALL\_TECHNOLOGIES\_QUERY =*

*`SELECT id\_categ AS id, nume\_categorie AS technology FROM portfolio.categorie`*;

*connection*.*query*(*SELECT\_ALL\_TECHNOLOGIES\_QUERY*, (err, results) *=>* {

*if* (err) {

*return* *res*.*send*(err);

} *else* {

*return* *res*.*json*({

data: results,

});

}

*connection*.*end*();

});

});

atunci, la accesarea din browser a url-ului *“localhost:4000****/technology****”*, vom observa că API-ul ne returnează informațiile solicitate.

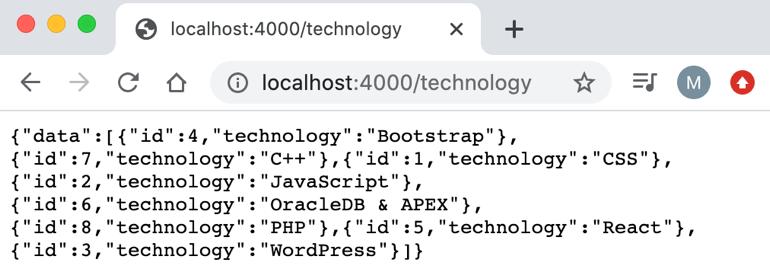


Figura 4.5 – API funcțional

## 4.3 – Inițializarea proiectului ReactJS, instalarea dependințelor

Acum suntem pregătiți pentru începerea lucrului la aplicația ReactJS. Navigăm în folderul *Portfolio*, unde vom crea un nou folder, numit *portfolio-app.* Vom deschide acest folder cu VS-Code, iar din *Terminal* vom lansa comanda:

npx create-react-app portfolio-app

După finalizarea instalării, se navighează în interiorul folderului */portfolio-app* unde se observă structura de foldere și fișiere create în timpul inițializării proiectului.

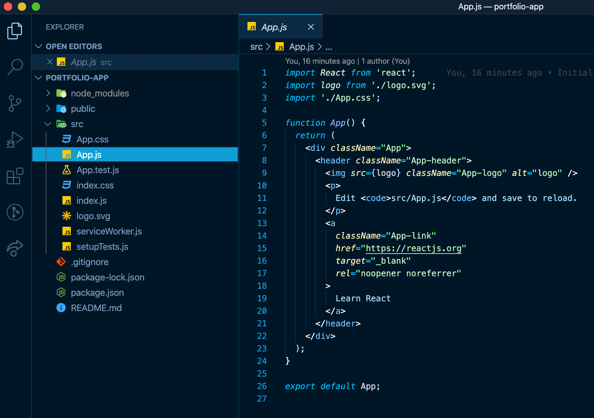


Figura 4.6 – Structura de foldere și fișiere

Se continuă cu instalarea dependințelor ce urmează a fi folosite:

npm install react-router-dom –save

npm install react-router-bootstrap –save

Aceste module facilitează navigarea atât între paginile statice ale aplicației cât și între paginile create în mod dinamic din resursele obținute.

Acum continuăm instalarea dependințelor ce ne vor ajuta la stilizarea și crearea elementelor și componentelor *responsive*:

npm install react-bootstrap bootstrap --save

npm install mdbreact --save

În acest moment se poate lansa în execuție aplicația, prin comanda: npm start

Această comandă realizează compilarea aplicației în *modul dezvoltare* și încărcarea acesteia în browser-ul Chrome, browser-ul implicit pe acest calculator, la adresa url *”localhost:3000”*.

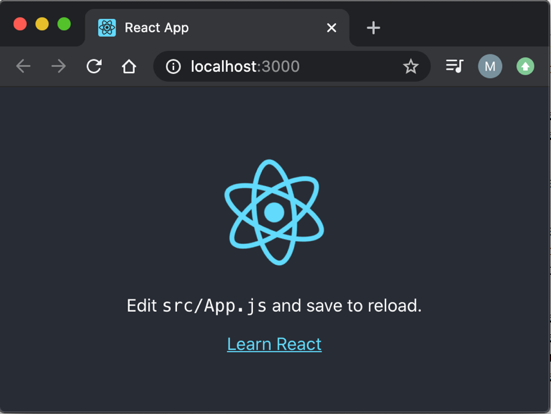


Figura 4.7 – Rularea aplicației inițiale in browser

Din acest moment se poate începe personalizarea aplicației.

Fișierul de pornire este *index.html*, unde regăsim elementul<div *id*="root"></div> . În interiorul acestui element se va întâmpla toată *magia ReactJS*.

Fișierul *index.js* importă componenta *ReactDom* din *React* și, prin intermediul metodei render(), va rula componenta <App />, de asemeni importată la începutul fișierului.

În componenta *App.js* ne vom folosi de primele elemente din librăriile importate, și anume de *BrowserRoutes* ca *Router* din *react-router-dom.* Acest element afișează cele 3 componente în elementul <div *id*="root"></div> din *index.html*.

**function** App() {

return (

<Router>

<div className="App">

<Navigation />

<Routing />

<Footer />

</div>

</Router>

);

}

export default App;

Componenta *Navigation.js* se folosește de elementele importate din librăriile react- bootstrap și react-router-bootstrap. Aici se construiește meniul de navigare în aplicație și se aplică logica funcției isAuth() pentru a determina elementele ce trebuie afișate în meniu.

În funcția isAuth(), exportată de componenta cu același nume, regăsim alte 2 funcții:

* funcția checkCookie()*,* care verifică existența unui *cookie* creat la momentul autentificării unui utilizator în aplicație. Pentru a avea ce verifica, aceasta apelează funcția getCookie();
* funcția getCookie()*,* va citi toate *cookie-urile* din browser și va căuta *cookie-ul* cu numele de *”portfolio-app”*; funcția va returna fie valoarea acestuia fie un *string* gol;

**const** getCookie = (cname) **=>** {

.................

};

**const** checkCookie = () **=>** {

.................

};

export default **function** isAuth() {

return checkCookie();

}

În final, funcția isAuth() va returna o valoarea de tip Boolean, true sau false, ce va fi folosit în condițiile ternare din interiorul aplicației ce stabilesc clasele de stilizare ce se aplică elementelor, în special “diplay: none” și “display: block” , în scopul afișării meniului de navigare creat în componenta *Navigation*.

.............................

<Navbar> ........

<Nav> ........

<LinkContainer to="/contact">

<Nav.Link>Contact</Nav.Link>

</LinkContainer>

<LinkContainer

to="/login"

className={isAuth() ? "d-none" : "d-block"}

>

<Nav.Link>Log-In</Nav.Link>

</LinkContainer>

<LinkContainer

to="/logOut"

className={isAuth() ? "d-block" : "d-none"}

>

<Nav.Link>Log-Out</Nav.Link>

</LinkContainer>

.............................

Componenta *Routing.js* este în strânsă legătură cu *Navigation.js*, întrucât aici sunt definite rutele către care va fi direcționat utilizatorul la apăsarea pe unul din butoanele de navigație, după ce, în prealabil, la începutul fișierului au fost importate toate paginile ce vrem să le fie atribuită o rută.

.............................

import Home from "../pages/home/home";

import About from "../pages/about/about";

.............................

<Switch>

<Route path="/" exact component={Home} />

<Route path="/about" exact component={About} />

<Route path="/portfolio" exact component={Portfolio} />

.............................

În formularul de *LogIn* regăsim clasa cu același nume, ce returnează un formular ce conține două componente funcționale care stochează în *state-ul* clasei valorile câmpurilor, în funcție de interacțiunea utilizatorului.

În contructorul clasei inițializez cu string-uri goale *state-ul* pentru trei variabile: două câmpuri de input și un mesaj de eroare.

*this*.*state* *=* {

user: "",

password: "",

errMessage: "",

};

Avem patru metode ce aparțin acestei clase:

handleChange = (e, attr) **=>** {

**const** newState = { ...this.state };

newState[attr] = e.target.value;

this.setState(newState);

};

* este metoda atașată celor două elemente de input;
* primește doi parametri: evenimentul și atributul elementului care a declanșat evenimentul;
* funcție de valoarea atributului primit, se actualizează valoarea variabilei respective din *state* cu ajutorul metodei this.setState() cu valoarea regăsită în e.target.value;

isValid() {

if (this.state.user === "" || this.state.password === "") {

return false;

}

return true;

}

* verifică valorile din *state* pentru cele două câmpuri de input și returnează un Boolean, true sau false, ce va fi folosit la activarea/dezactivarea butonului de *”Log-In”*;

handleSubmit = (e) **=>** {

e.preventDefault();

fetch("http://localhost:4000/login", {

method: "POST",

headers: {

"Content-Type": "application/json",

Accept: "application/json",

},

*// We convert the React state to JSON and send it as the POST body*

body: JSON.stringify(this.state),

})

.then(**function** (response) {

return response.json();

})

.then((data) **=>** {

**let** response = data.data[0];

if (!response) {

*// show error message*

this.logInReject();

} else {

if (

response.user === this.state.user &&

response.password === this.state.password

) {

*// go set cookie for LOGED-IN*

Cookies.setCookie("portfolio-app", "loged-in", 1);

*// redirect to HomePage after successfully Log-in*

window.location.replace("/");

} else {

*// show error message*

this.logInReject();

}

}

})

.catch((err) **=>** console.log("Error from 'login fetch':", err));

};

* este metoda invocată implicit la apăsarea butonului de *”Log-In”*, buton ce este definit ca type=“submit” pentru formular;
* în interiorul metodei se face request-ul de tip POST către ruta de *”/login”*, pusă la dispoziție de serverul NodeJS;
* funcție de răspunsul ce vine de la server, se face validarea:
  + dacă validarea are loc, este invocată funcția Cookies.setCookie(), ce va crea un *cookie* de login cu valabilitatea de o ora iar utilizatorul este redirecționat către pagina principală, putând să observe și schimbarea intrărilor din bara de navigație;
  + dacă validarea nu are loc, este invocată funcția logInReject();

logInReject() {

this.setState({ user: "" });

this.setState({ password: "" });

this.setState({

errMessage: "Incorrect username or password !!! ",

});

}

* dacă răspunsul de la server nu permite validarea autentificării, atunci se golește *state-ul* componentelor de input și se încarcă mesajul de eroare ce va fi afișat utilizatorului.

Odată autentificat, utilizatorul poate apăsa butonul de *”Log-Out”* din meniu, moment în care componenta *LogOut* apelează funcția Cookie.deleteCookie() din fișierul *cookies.js*, și se șterge *cookie-ul* setat la momentul autentificării. La final se face o redirecționare către pagina de start și implicit o reîncărcare a componentei de navigație.

Pagina *Portfolio* este randată prin intermediul componentei funcționale *Portfolio* din fișierul *Portfolio.js*.

Aici mă voi folosi de *React Hooks*, adică *cârlige*. Unul dintre cârlige este useEffect(), care îi spune Reactului că trebuie să facă *ceva* după fiecare render. Acel *ceva* este o funcție, care în cazul de față face un request asincron către un API pentru a primi lista proiectelor din baza de date.

**function** Portfolio() {

useEffect(() **=>** {

fetchItems();

}, []);

**const** [items, setItems] = useState([]);

**const** fetchItems = **async** () **=>** {

**const** data = await fetch("http://localhost:4000/projects");

**const** items = await data.json();

setItems(items.data);

};

return (

<main className="mt-5 ml-3 mr-2">

<Row className="portfolio mt-5 mb-5">

{items.map((item, index) **=>** (

<Card

key={index}

className="card m-4 col-xl-2 col-lg-3

col-md-3 col-sm-4 hoverable"

style={{ width: "18rem" }}

>

..........................................

</Card>

))}

</Row>

</main>

);

}

export default Portfolio;

useEffect() primește de regulă un singur parametru, și anume funcția *callback* ce trebuie executată. Însă, dacă îi dăm și al doi-lea parametru, și anume o listă goală, îi *spunem* Reactului să facă o singură dată render.

Răspunsul ce vine de la server este o listă iar aceasta este salvată în *state-ul* componentei funcționale cu ajutorul unui alt *cârlig*, și anume useState().

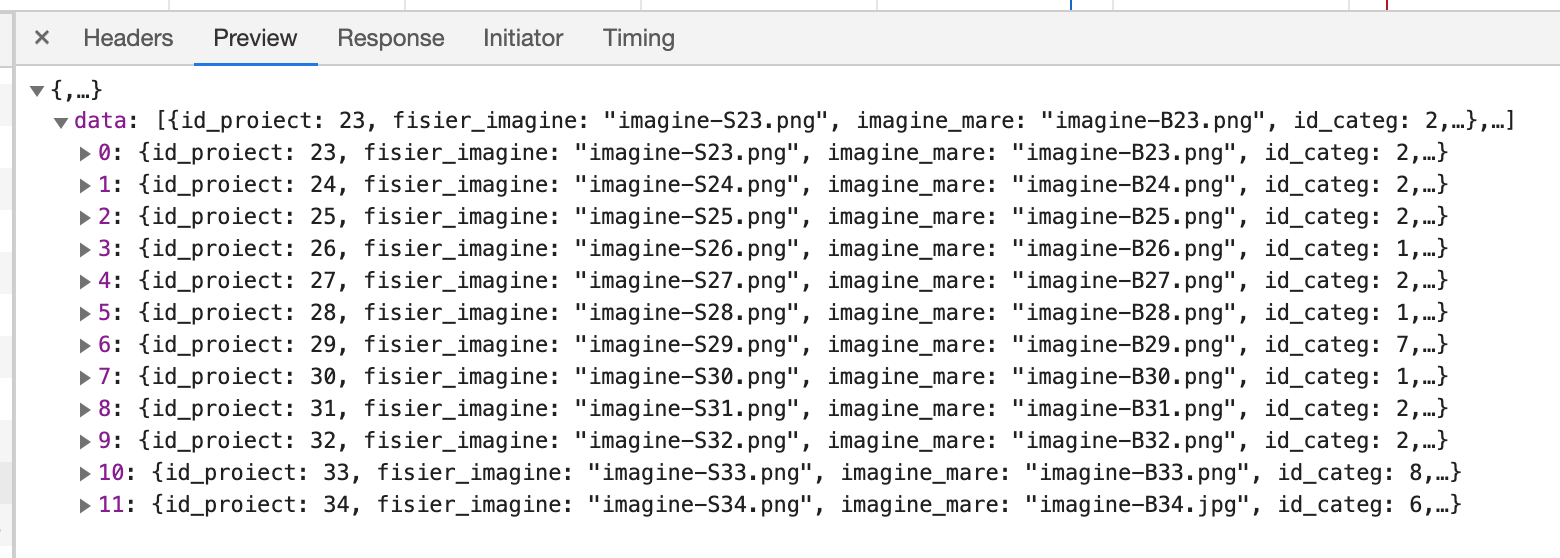


Figura 4.8 – Răspunsul de la server, sub formă de listă de obiecte

De aici totul este simplu, funcția va return-a o componentă <main> în interiorul căreia se construiesc cardurile cu proiectele ce vor fi afișate, folosind metoda map pe lista de proiecte.

Fiecare card are atribuit un element <Link> din react-router-dom, ce va fi afișat sub forma unui buton. Acest element face ca, prin intermediul react-router, să putem construi, în mod dinamic, url-ul unic pentru fiecare proiect, bazat pe *id\_proiect* ce vine din baza de date

<Link

to={`/portfolio/project=${item.id\_proiect}`}

className="btn btn-info btn-lg"

>

More details

</Link>

Odată apăsat butonul *”More details”* de pe un card, aplicația ajunge în componenta *Project* din scriptul *Project.js*. Aici lucrurile sunt oarecum similare cu ceea ce se întâmpla în componenta *Portfolio*. Însă, pentru a putea salva într-o variabilă id-ul proiectului pe cardul căruia s-a apăsat, mă voi folosi de Obiectul *”match”* pus la dispoziție de Obiectul *”this.props”* . *”match”* la rândul lui, ne dă acces la Obiectul *”params”* din care putem salva id-ul dorit, ce se regăsește in url-ul proiectului.

**const** id = parseInt(match.params.id\_proiect);

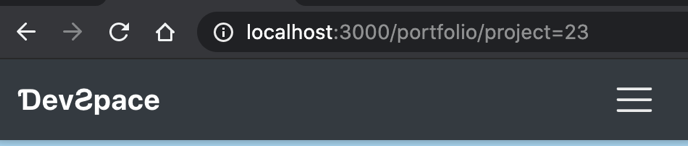


Figura 4.9 – URL-ul proiectului



Figura 4.10 – Afișarea în consolă a Obiectului *„match”*

De îndată ce avem id-ul proiectului, se face un request asincron către baza de date, dupa care filtrăm răspunsul cu lista cu proiecte și salvăm în *state-ul*  componentei obiectul cu proiectul dorit.

În final, componenta funcțională *Project* va transmite pe *props* către *ProjectDetails* proiectul salvat în *state*.

*ProjectDetails* este o clasă în constructorul căreia, prin destructurare, salvez în variabile datele venite prin *props* de la componenta părinte  *Project.*

**const** {imagine\_mare, nume\_proiect, prezentare, data\_proiect,

link, nume\_categorie, id\_proiect } = this.props.data;

Datele astfel salvate sunt folosite pentru construirea elementelor ce vor afișa pe ecran proiectul selectat.

Ceea ce prezintă un interes deosebit, este butonul *”Delete Project”*.

Acesta, după cum am prezentat și la componenta *Navigation*, este strict legat de funcția isAuth(), adică este vizibil și accesibil doar dacă utilizatorul este autentificat.

Butonul are setat pe evenimentul de *onClick* metoda de deleteProject() căreia îi sunt transmiși doi parametri: evenimentul și id-ul proiectului ce se dorește a fi șters.

<Button

className={ isAuth() ? "d-block" : "d-none" }

variant="danger"

size="lg"

onClick={(e) **=>** deleteProject(e, { id\_proiect })}

>

Delete Project

</Button>

Metoda deleteProject() nu prezintă elemente deosebite. Aceasta face un request de tipul DELETE către API-ul livrat de către serverul NodeJS pe ruta *”/delProject”*, la final efectuându-se o reîncărcare a paginii *Portolio*, în care se poate observa ca proiectul nu mai este disponibil.

window.location.replace("/portfolio");

De asemeni, confirmarea că ștergerea a fost efectuată cu succes se poate vedea și în consola browser-ului.

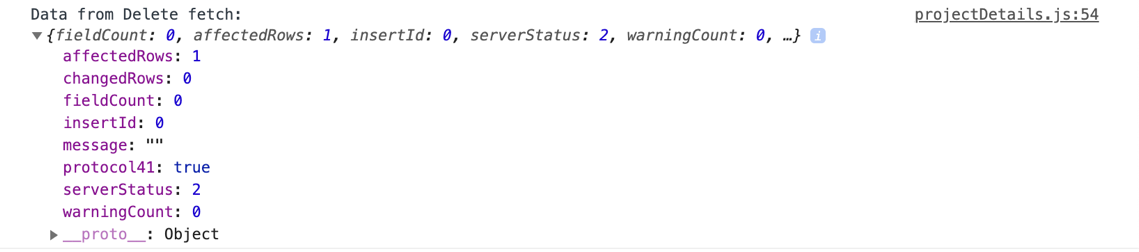


Figura 4.11 – Afișarea în consolă a răspunsului de succes la ștergerea unui proiect

După autentificare, utilizatorul are la dispoziție pagina de *Add Project.* Componenta acestei pagini este clasa *AddProject* ce implementează un formular cu componente funcționale.

Deosebirea față de celelalte formulare din aceasta aplicație, sunt prezența câmpurilor *Select*, *Date-Picker* și *Input file*.

Câmpul *Select* este populat în mod dinamic cu elemente *Option* ce au valori obținute din baza de date prin intermediul unui request asincron de tip GET.

Selectarea datei se poate face cu ajutorul componentei importate din librăria *Date-Picker*.

npm install react-datepicker –-save

*State-ul* clasei este actualizat în mod constant prin intermediul metodei handleChange() pentru toate elementele formularului.

Butonul de *”Add Project”* este deblocat de bine-cunoscuta metoda isValid() doar după completarea tuturor câmpurilor din formular.

Și acest buton este de tipul *submit*, ceea ce face ca la apăsarea acestuia să se declanșeze metoda de handleSubmit() ce primește ca unic parametru evenimentul și face un request de tip PUT către ruta *”/addProject”*. Pe ramura de *success* se va afișa în consolă răspunsul de la serverul NodeJS.

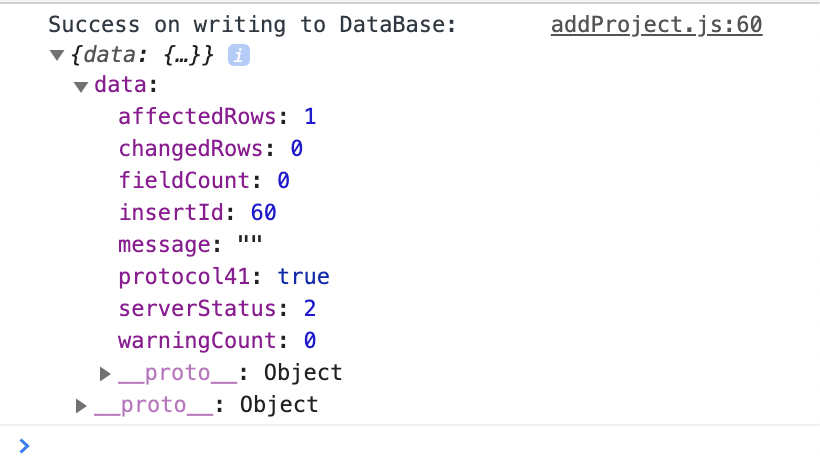


Figura 4.11 – Afișarea în consolă a răspunsului de succes la adăugarea unui proiect

Formularul nu este funcțional 100%, în sensul că, input-urile pentru imaginile proiectului vor transmite doar un *string* cu numele imaginilor, astfel că, proiectul ce va fi salvat în baza de date și apoi afișat în pagina *Portfolio*, nu va avea imaginile disponibile.

Pentru a avea aceste câmpuri într-adevăr funcționale, va fi nevoie de implementarea unei dependințe de tipul *multer* ( <https://codeburst.io/image-uploading-using-react-and-node-to-get-the-images-up-c46ec11a7129> ) pentru a putea salva imaginile pe server.

Pagina *Resume* se folosește de utilitarul React-pdf, ce se poate adăuga proiectului cu comanda din Terminal:

npm install react-pdf --save

Acesta facilitează integrarea și afișarea unui document *PDF* . Întrucât documentul din acest proiect are o singură pagină, nu au fost implementate funcționalitățile de navigare între pagini, sau de afișare a numărului paginii.

Pagina *Contact*

Pagina *About*

În ceea ce privește stilizarea paginilor, voi puncta doar două aspecte:

* am pentru marea majoritate a elementelor HTML am folosit clasele sau chiar blocurile de elemente predefinite ale librăriei *Bootstrap*;
* am implementat în fișierul principal *App.css* sistemul de variabile pentru culorile folosite în acest proiect. Pentru fiecare componentă în parte am creat și folosit propriul fișier de stiluri, cu importarea variabilei corespunzătoare pentru acea componentă. Prin această metodă, este foarte ușor să se schimbe aspectul proiectului.

:root {

--nav-bg-color: #494f52;

--about-bg-color: #a2ccd7;

--resume-bg-color: #77e997;

--contact-bg-color: #188ef4;

--contact-media-icons: #1c2a48;

--portfolio-bg-color: #f5e6cc;

--addProject-bg-color: whitesmoke;

--logIn-bg-color: #ececec;

--logIn-form-color: #eae7dc;

--notFound--bg-col: #cafafe;

--color-1: #fff;

--color-2: yellow;

--color-btn-disabled: lightgrey;

}

.projectDetails {

background-color: var(--project-bg-color);

}

# CAPITOLUL 5 – Testarea aplicației

Întrucât acest proiect se dorește a fi pus la dispoziția publicului larg pe un domeniu de internet, trebuie să funcționeze pe majoritatea browserelor de internet cât și pe o multitudine de dispozitive.

Întrucât dezvoltarea aplicației s-a făcut 95% pe browserul Google Chrome, pot spune că aceasta are o proprietate *responsive* în proporție de 75% atât în acest browser, cât și în Firefox sau Safari.

In ceea ce privește funcționalitatea aplicației, am observat următoarele deficiențe:

* ***Firefox v.77.0.1 (64-bit):***
  + Nu se adaugă întotdeauna proiectul în baza de date și nici nu este afișat mesaj de eroare în consolă;
  + Nu se face ștergerea unui proiect din baza de date, având mesaj de eroare în consolă;



Figura 5.1 – Afișarea în consolă a erorii la ștergerea unui proiect în Firefox

* ***Safari v. 13.1.1 (15609.2.9.1.2):***
  + Nu se adaugă întotdeauna proiectul în baza de date și nici nu este afișat mesaj de eroare în consolă;
  + Nu se face ștergerea unui proiect din baza de date, având mesaj de eroare în consolă;

# CAPITOLUL 6 – Concluzii

## 6.1 – Concluzii

## 6.2 – Posibilități de dezvoltare

Implementarea serviciului de trimitere a mesajelor din formularul Contact

Validarea la Log-In ar trebui făcută pe server

Crearea unui nou API care să returneze un singur proiect, pe baza unui id\_proiect, iar din componenta Project să apeleze acel API, pentru a se evita întoarcerea tuturor proiectelor și apoi căutarea proiectului dorit după id;

Implementarea validării conținutului câmpurilor din formulare împotriva atacurile de tip SQL Injection

Implementarea unei dependințe gen *multer* pentru salvarea imaginilor pe server.

# BIBLIOGRAFIE

1. Conf. Dr. ing. Mihai DAMIAN - Curs UTCN-IAP – *Programarea Interfețelor Utilizator*, Cluj-Napoca, 2020;
2. Prof. ing. Ildiko REVNIC – Curs UTCN-IAP – *Crearea și exploatarea bazelor de date relaționale*, Cluj-Napoca, 2019;
3. Conf. Dr. ing. Mihai DAMIAN & Prof. Sanda TOMA – Curs UTCN-IAP – *Realizarea saiturilor și aplicațiilor pentru WWW*, Cluj-Napoca, 2019;
4. <https://reactjs.org>;
5. <https://wikipedia.org;>
6. Icons made by https://www.flaticon.com

# Resurse online multimedia

1. <https://codeforgeek.com/handle-get-post-request-express-4/>;

# Mulțumiri

##### **Structura lucrării**

* O introducere cuprinzând descrierea problemei care urmează a fi rezolvată (obiective, cerințe, restricții, etc.)
* Posibilități (variante) de rezolvare a problemei (cu referințe la bibliografie)
* Justificarea metodei alese
* Prezentarea detaliată a cerințelor aplicației
* Prezentarea arhitecturii hardware și/sau software a aplicației
* Prezentarea detaliată a modului de rezolvare a temei (cu programe listate și explicate, copii ecran cu rezultate etc.)
* Precizarea testelor efectuate și rezultatele obținute
* Posibilități de continuare și extindere a temei.
* Bibliografie