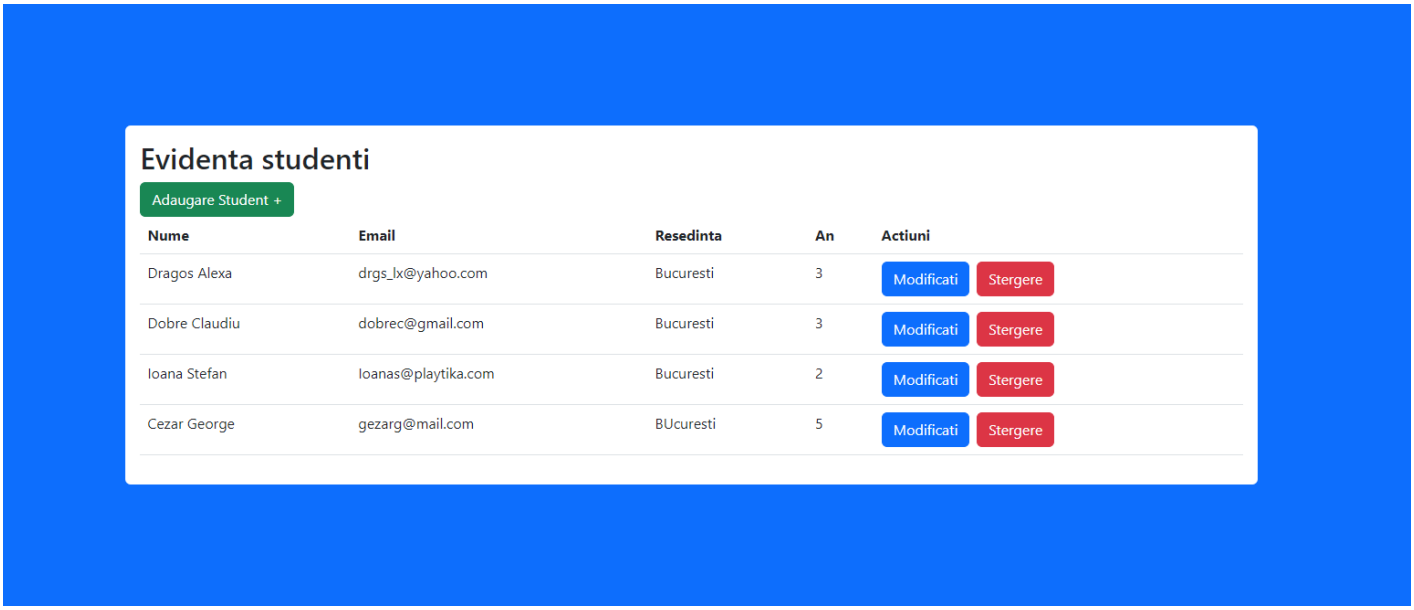
PROIECT INGINERIE SOFTWARE

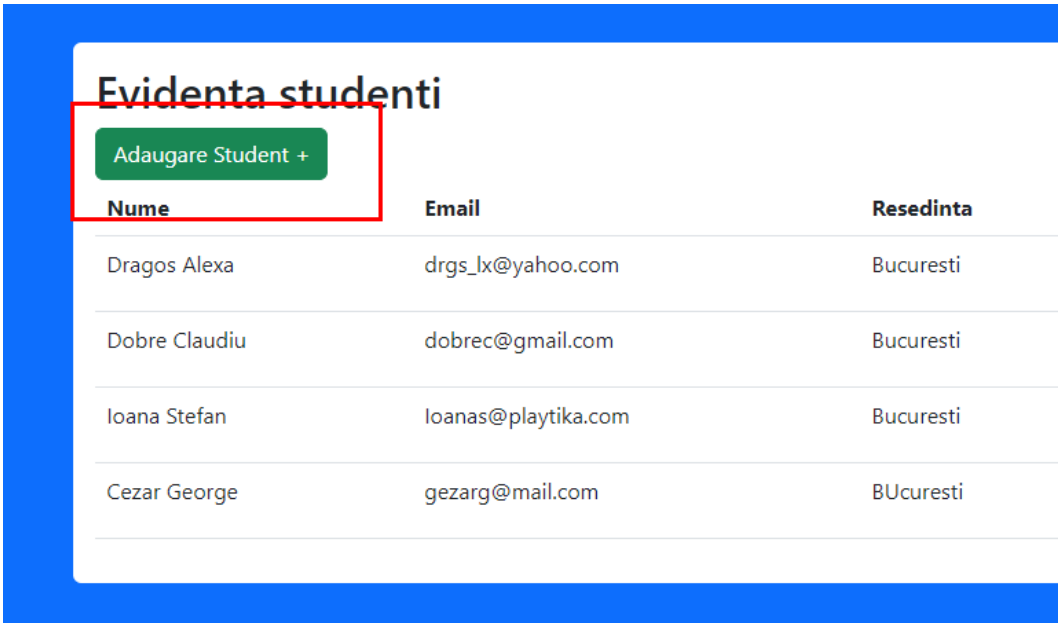
DRAGOS-STEFAN ALEXA

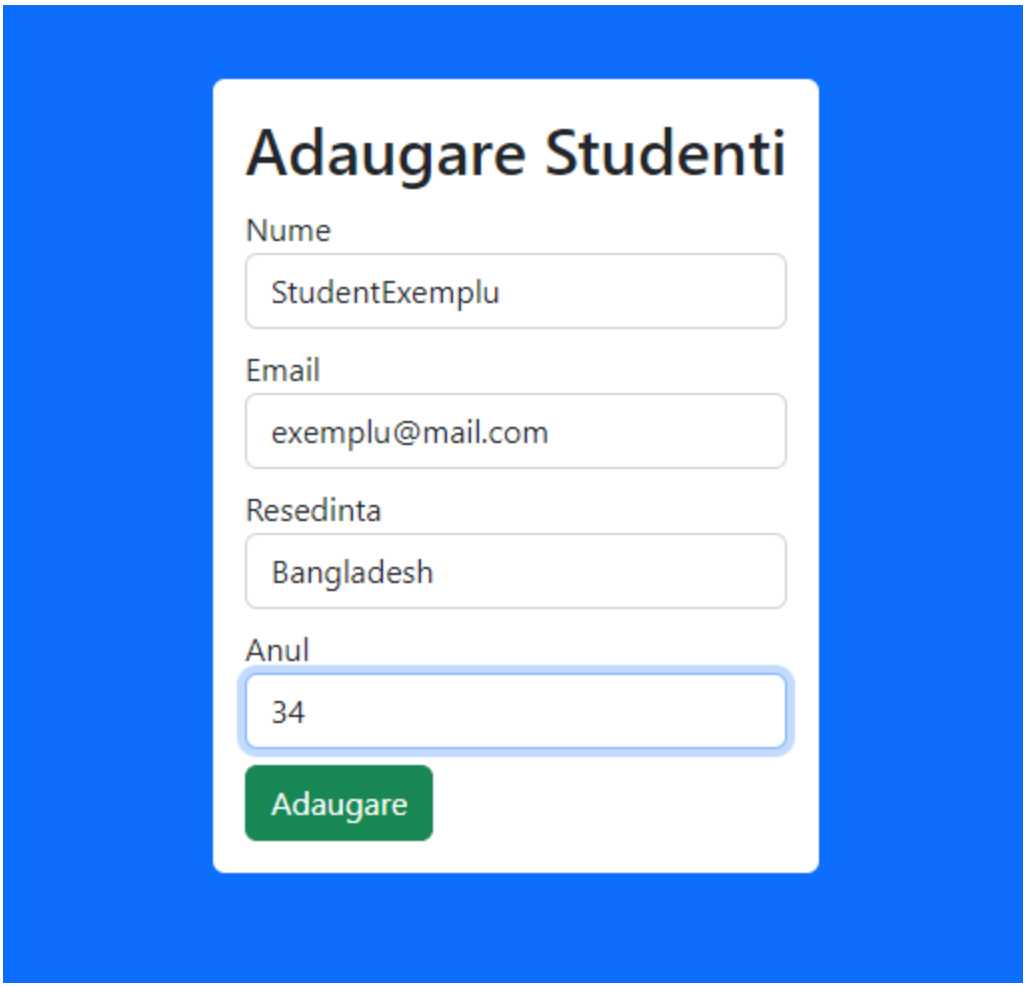
UNIVERSITATEA HYPERION, FACULTATEA DE INFORMATICA, ANUL 3

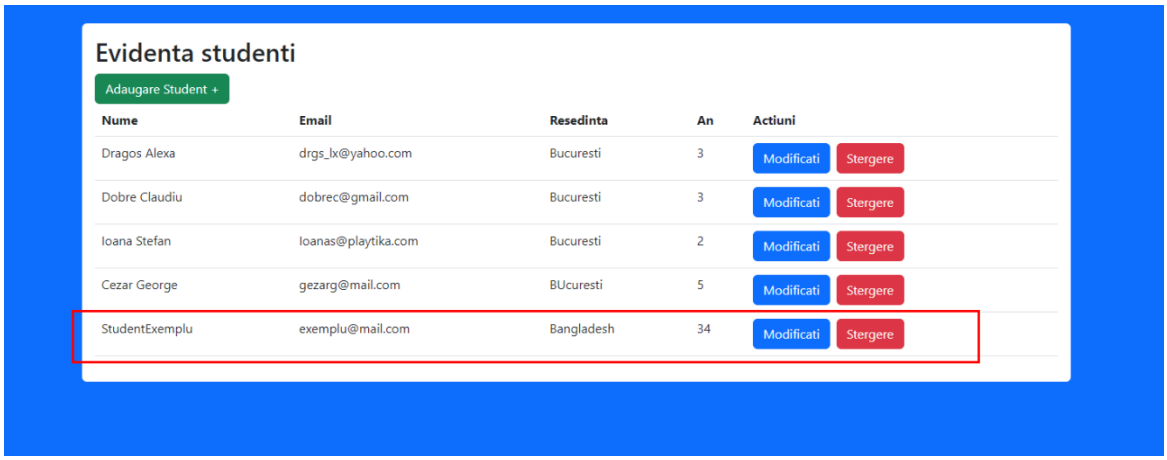
Aplicatia este o baza e date ce va evidentia studentii. Se pot adauga student, modifica datelor acestora, precum si stergerea unui student:



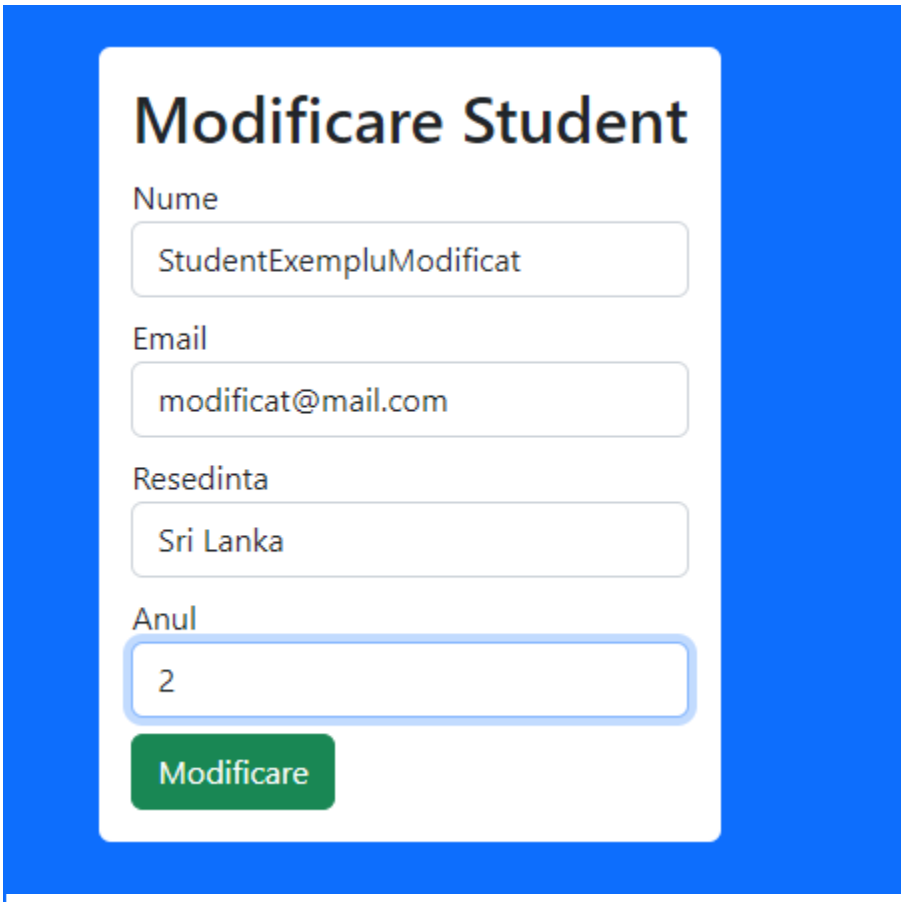
Adaugare



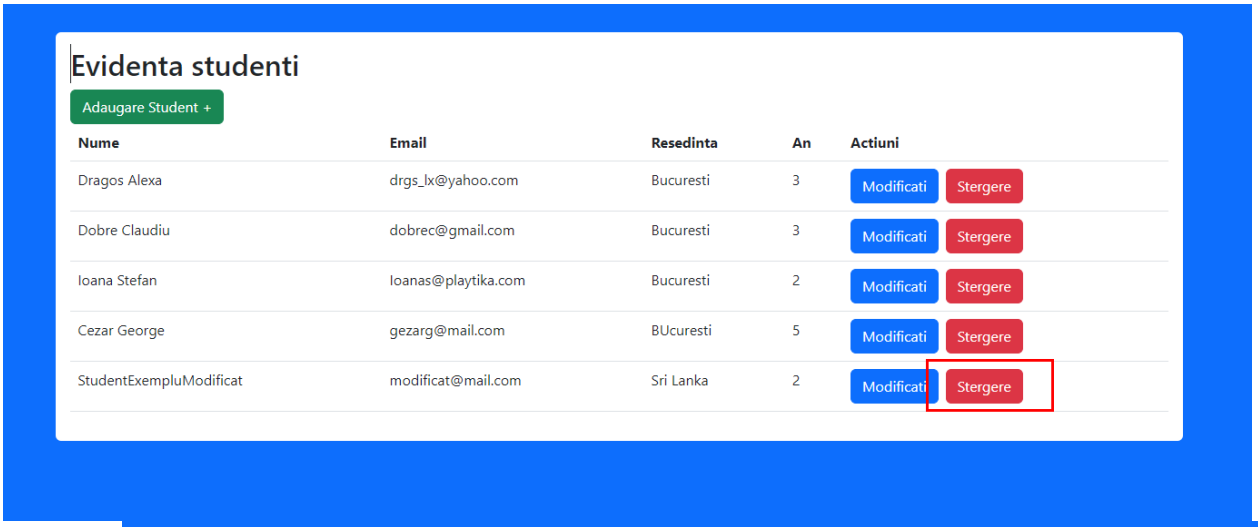

Adaugare Studenti
Nume
StudentExemplu
Email
exemplu@mail.com
Resedinta
Bangladesh
Anul
34
Adaugare

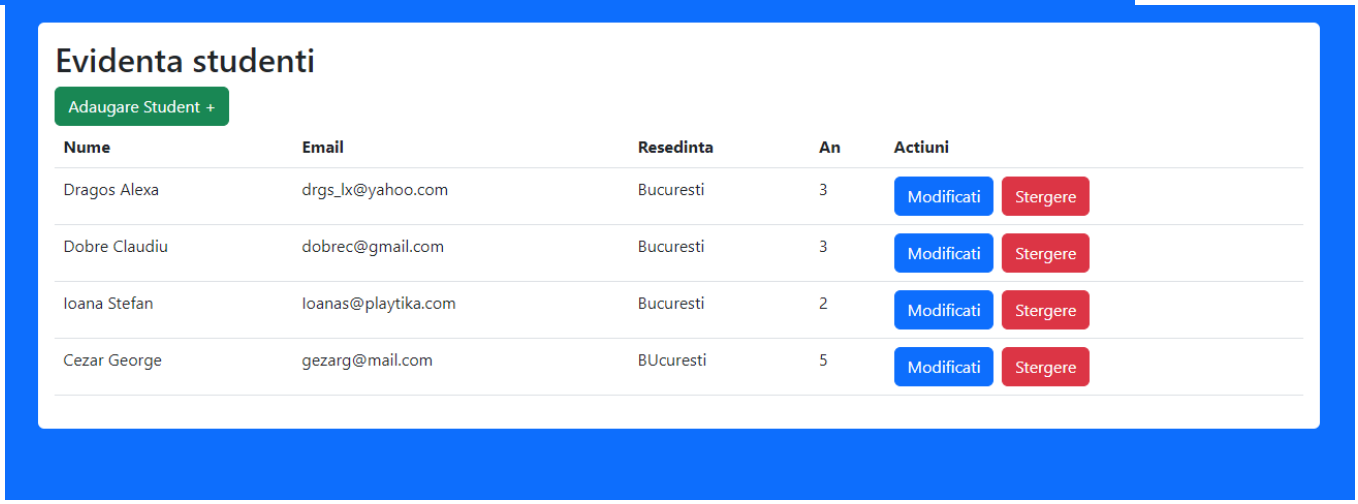

Evidenta studenti
Adaugare Student +
Nume
Email
Resedinta
An
Actiuni
Dragos Alexa
drgs_lx@yahoo.com
Bucuresti
3
Modificati
Stergere
Dobre Claudiu
dobrec@gmail.com
Bucuresti
3
Modificati
Stergere
Ioana Stefan
Ioanas@playtika.com
Bucuresti
2
Modificati
Stergere
Cezar George
gezarg@mail.com
BUcuresti
5
Modificati
Stergere
StudentExemplu
exemplu@mail.com
Bangladesh
34
Modificati
Stergere

Modificare Student



StudentExempluModificat	modificat@mail.com	Sri Lanka	2	



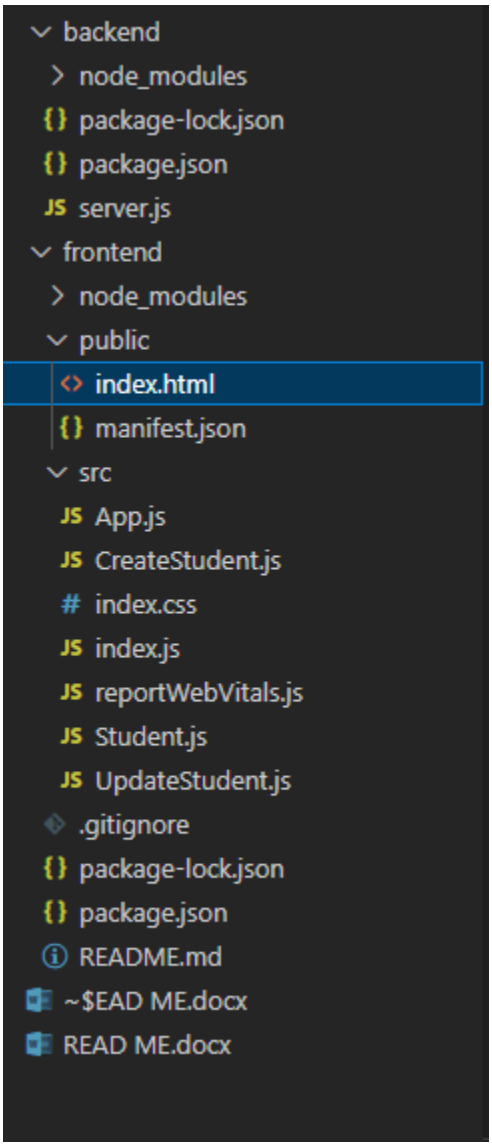


Tehnologii folosite:

- **React Javascript(frontend)**
- **React bootstrap (pentru stilizre)**
- **Axios pentru data fetching**
- **Node JS (backend)**
- **MYSQL – baza de date**
- **PHP MYADMIN – DB Server**

!!ATENTIE – In cod se vor mai folosi comentarii cu simbolul "//" unde se vor explica pe scurt anumite functionalitati!!

Structura fisierelor



backend – node modules sunt modulele principale dupa care functioneaza aplicatia si scripturi precum cele de local server

package.json – este templateul dupa care se vor lua dependintele acestei aplicatii precum – cors, express (node js framework) si mysql

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "mysql": "^2.18.1"
  }
}
```

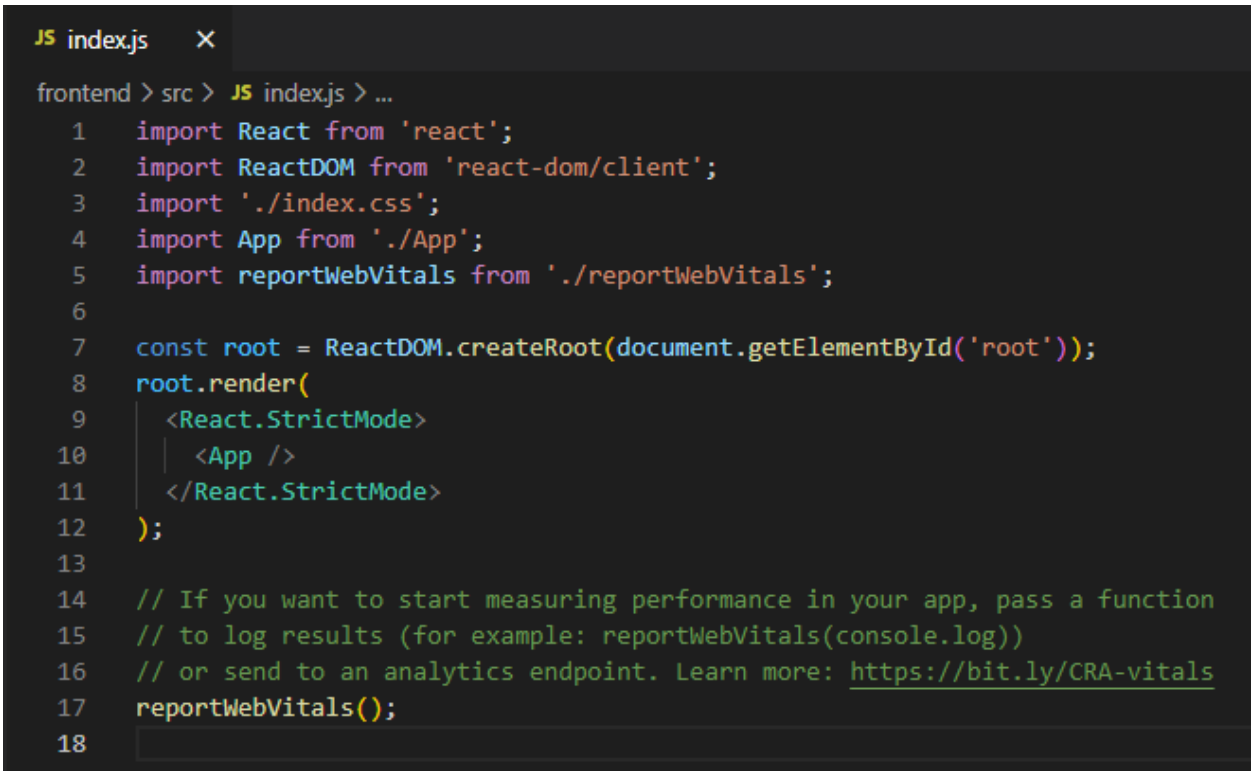
Server .js – fisierul principal de backend (se va explica ulterior)

Frontend – package lock.json functioneaza pentru modulele de frontend

Package.json – dependintele pe partea de frontend:

```
{
  "name": "frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.3.6",
    "bootstrap": "^5.2.3",
    "react": "^18.2.0",
    "react-bootstrap": "^2.7.4",
    "react-dom": "^18.2.0",
    "react-router": "^6.10.0",
    "react-router-dom": "^6.10.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
```

INDEX.JS



Index.js este pagina principala dupa care se va randa aplicatia. Se vor importa modulele **'React'** pentru sintaxa, **'ReactDOM'** pentru manipularea domului, **index.css** pentru stilizarea aplicatiei (in special fonturi)

Root.render va randa fisierul App. React StrictMode este modul strict pentru aplicatie.

ReportWebVitals(); Este o metoda predefinita din modulele de 'react' ce permite testarea permanenta a aplicatiei. Atunci cand nu este ceva declarat correct, de exemplu, va da eroare rularea acesteia si nu se va mai putea afisa. Se va afisa eroarea respectiva.

APP.JS

```
import 'bootstrap/dist/css/bootstrap.min.css'
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Student from './Student'
import CreateStudent from './CreateStudent';
import UpdateStudent from './UpdateStudent';

function App() {
  return (
    <div className="App">
      <BrowserRouter>
        <Routes>
          <Route path='/' element ={<Student />}></Route>
          <Route path='/create' element ={<CreateStudent />}></Route>
          <Route path='/update/:id' element ={<UpdateStudent />}></Route>
        </Routes>
      </BrowserRouter>
    </div>
  );
}

export default App; //Exporteaza componenta pentru a putea fi citit de index.js
```

Aceasta pagina se foloseste doar pentru a stabili rutele URL-urilor si a randa componentele principala pentru frontend: **Student**, **CreateStudent**, **UpdateStudent.** Fiecare componenta are URL-ul propriu pentru a putea fi citita de backend .

De asemenea, partea de frontend va comunica in permanenta cu aceste router: ex: localhost:3000/create ne va permite sa accesam pagina Adaugare student aratata anterior.

```
import 'bootstrap/dist/css/bootstrap.min.css'
import { BrowserRouter, Routes, Route } from 'react-router-dom';
```

aceste liniide cod vor importa fisierul bootsrap pentru stilizarea aplicatiei precum si componente predefinite in modulele de react-router-dome – **BrowserRouter, Routes, Route**

```
import Student from './Student'
import CreateStudent from './CreateStudent';
import UpdateStudent from './UpdateStudent';
```

aceste linii de cod vor importa componentele principale de frontend pentru a putea fi citita de **App.js**

return (…codul) ne va returna fiecare componenta randata.

STUDENT.JS

```
import React, { useEffect, useState } from 'react'; //se vor importa modulele
import axios from 'axios'; //tehnologia axios folosita pentru frontend
import { Link } from 'react-router-dom'; //Linkuri de rutare ale componentelor

    function Student () {
        const [student, setStudent] = useState([])
        useEffect(() => {
            axios.get('http://localhost:8081/')
            .then(res => setStudent(res.data))
            .catch(err => console.log(err));
        }, [])
```

Aceasta functie va seta Stateul principal pentru initalizare “student” si “SetStudent” intr-un array. Mai apoi va prelua din backend URL-ul http:localhost:8081/ si ne va returna un raspuns. (in backend este queryul unde ne va afisa toate datele din baza de date intr-un tabel.

```
 const handleDelete = async (id) => {
            try {
                await axios.delete('http://localhost:8081/student/'+id)
                window.location.reload()
            }catch(err) {
                console.log(err);
            }
        }
```

Aceasta componenta ne va prelua datele din baza de date, va adauga un id (ce va fi preluat tot din baza de date) apoi ne va sterge acel rand din tabel in functie de ID. Este o funtie de tip async ce ne permite sa nu facem mai multe operatiuni pe baza de date pana nu se va executa operatiunea precedenta.

Ambele metode au prin axios metoda **catch** pentru a putea afisa erori in caz ca un proces nu functioenaza corespunzator.

```
return (
      <div className='d-flex vh-100 bg-primary justify-content-center align-
items-center'> //className=…este stilul paginii folosit de bootsrap framework
        <div className='w-50 bg-white rounded p-3'>
         <h2>Evidenta studenti</h2> //Titlul Aplicatiei
            <Link to="/create" className='btn btn-success'>Adaugare Student
+</Link> //Este rutarea catre url-ul localhost:3000/create
            <table className='table'> //datele tabelului
                <thead>
                    <tr>
                    <th>Nume</th>
```

```
                    <th>Email</th>
                    <th>Resedinta</th>
                    <th>An</th>
                    <th>Actiuni</th>
                    </tr>
                </thead>
                <tbody>
                    {
                        student.map((data, i) => (
                            <tr key={i}>
                                <td>{data.Nume}</td>//afiseaza toate valorile
                                <td>{data.Email}</td>//una cate una din DB
                                <td>{data.Resedinta}</td>
                                <td>{data.An}</td>
                                <td>
                                    <Link to={`update/${data.ID}`} className='btn
btn-primary'>Modificati</Link> Este rutarea catre url-ul localhost:3000/update
                                    <button className='btn btn-danger ms-2'
onClick={e => handleDelete(data.ID)}>Stergere </button> //apelarea functiei
                                </td>                //handleDelete ce va lua
                            </tr>              //parametrii data si ID
                        ))
                    }
                </tbody>
            </table>
        </div>
      </div>
    )

    }
```

Acest este folosit in mare pentru crearea tabelului prin cod HTML.
Link permite rutarea catre URL-urile …/update si /create. Fiecare Buton in parte va afisa aceste doua componente. Practic, aceste Link ne trimite la url-urile aferente si ne va afisa pepagina componente aferente. Update se va folosi si de ID pentru a cunoaste ce ID din randul tabelului va fi modificat.

CREATESTUDENT.JS

```
import axios from 'axios' //se importa modulele necesare
import React, {useState} from 'react'
import { useNavigate } from 'react-router-dom'

//Initializarea Stateului prin valorile preluate din backend prin valoare +
setValoare
 const [nume, setName] = useState('')
  const [email, setEmail] = useState('')
  const [resedinta, setResidence] = useState('')
  const [an, setYear] = useState('')
  const navigate = useNavigate();

 function handleSubmit(event){
      event.preventDefault();
      axios.post('http://localhost:8081/create', {nume, email, resedinta, an})
      .then(res =>{
        console.log(res);
        navigate('/');
      }).catch(err => console.log(err)); //returneaza erori dupa caz
    }
```

handleSubmit va lua ca parametru un event. Axios.post va lua din server numele, emailul, resedinta si anul ce sunt prezente in backend ca mai apoi se seteze prin onChange fiecare valoare in baza de date.

Navigate('/) este o metoda din modulul de react ce ne va trimite la pagina initiala prin randarea url-ului de baza '/'.

```
return (
    <div className='d-flex vh-100 bg-primary justify-content-center align-items-
center'>
      <div className='w-40 bg-white rounded p-3'>
        <form onSubmit={handleSubmit}> //apelarea functiei handleSubmit
          <h2>Adaugare Studenti</h2>
            <div className='mb-2'>
                <label htmlFor="">Nume</label>
                <input type="text" placeholder='Introduceti Numele'
className='form-control'
                onChange={e => setName(e.target.value)}
            </div>              //seteaza valoarea campului in baza de date
            <div className='mb-2'>
                <label htmlFor="">Email</label>
```

```
                <input type="text" placeholder='Introduceti Emailul'
classNamе='form-control'
                onChange={e => setEmail(e.target.value)}/>
            </div>              //seteaza valoarea campului in baza de date
            <div className='mb-2'>
                <label htmlFor="">Resedinta</label>
                <input type="text" placeholder='Introduceti Resedinta
Studentului' className='form-control'
                onChange={e => setResidence(e.target.value)}/>
            </div>              //seteaza valoarea campului in baza de date
            <div className='mb-2'>
                <label htmlFor="">Anul</label>
                <input type="text" placeholder='Introduceti Anul de studiu'
classNamе='form-control'
                onChange={e => setYear(e.target.value)}/>
            </div>              //seteaza valoarea campului in baza de date
            <button className='btn btn-success'>Adaugare</button>
        </form>
      </div>
    </div>
  )
}
```

UPDATESTUDENT.JS

```
import axios from 'axios'; //importam modulele
import React, {useState} from 'react'
import { useNavigate, useParams } from 'react-router-dom';

//Initializarea Stateului prin valorile preluate din backend prin valoare +
setValoare
const [nume, setName] = useState('')
  const [email, setEmail] = useState('')
  const [resedinta, setResidence] = useState('')
  const [an, setYear] = useState('')
  const {id} = useParams();
  const navigate = useNavigate();


    function handleSubmit(event){
      event.preventDefault();
      axios.put('http://localhost:8081/update/'+id, {nume, email, resedinta, an})
      .then(res =>{
        console.log(res);
        navigate('/');
      }).catch(err => console.log(err));
    }
```

handleSubmit() in aceasta componenta va lua ca parametru un event precum in cazul antecedent. Axios.put va lua din server numele, emailul, resedinta si anul ce sunt prezente in backend ca mai apoi se seteze prin onChange fiecare valoare in baza de date + id-ul aferent pentru a ne putea asigura ca va modifica doar campul de care avem nevoie si nu aleatoriu.

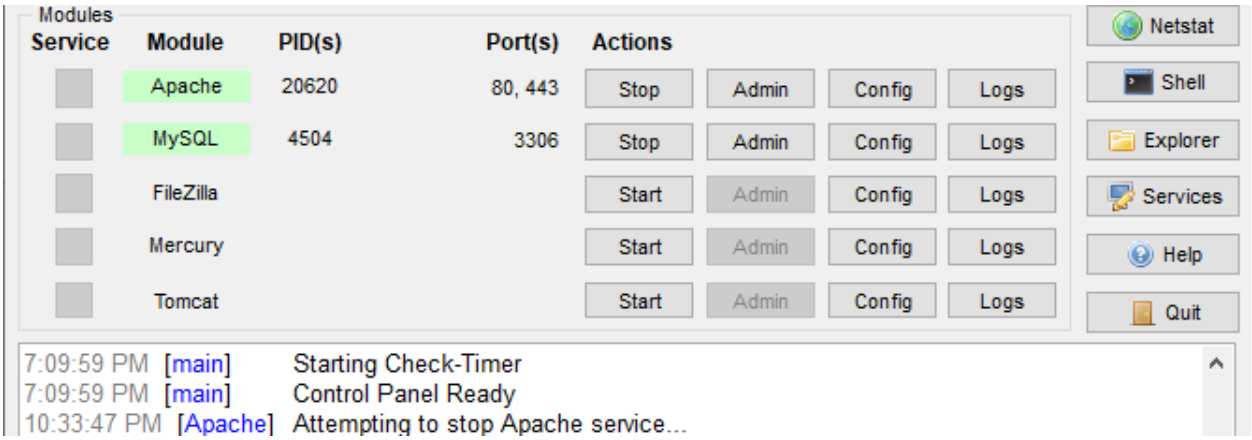
Metoda preventDefault() anuleaza evenimentul daca acesta poate fi anulat, ceea ce înseamna că acțiunea implicităacare aparține evenimentului nu va avea loc. De exemplu, acest lucru poate fi util atunci când: Facand clic pe un buton „Trimite", il impiedicaț sa trimita un formular. Facand clikc pe un link, impiedicati linkul sa urmărească adresa URL.

```
 <div className='d-flex vh-100 bg-primary justify-content-center align-items-
center'>
      <div className='w-40 bg-white rounded p-3'>
        <form onSubmit={handleSubmit}> //apelarea functiei handleSubmit
          <h2>Modificare Student</h2>
            <div className='mb-2'>
                <label htmlFor="">Nume</label>
                <input type="text" placeholder='Introduceti Numele'
className='form-control'
                onChange={e => setName(e.target.value)}/>
```

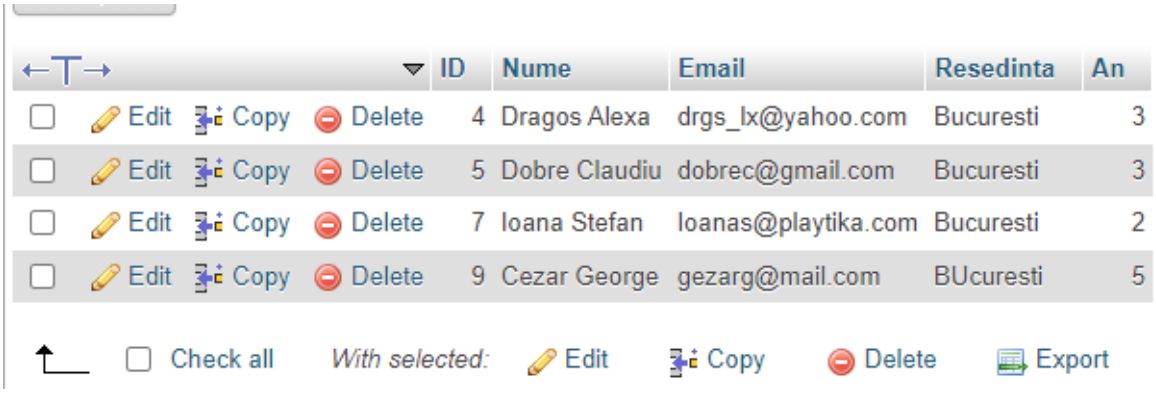
```
            </div>//modifica valoarea campului in baza de date

            <div className='mb-2'>
                <label htmlFor="">Email</label>
                <input type="text" placeholder='Introduceti Emailul'
className='form-control'
                onChange={e => setEmail(e.target.value)}/>
            </div> modifica valoarea campului in baza de date

            <div className='mb-2'>
                <label htmlFor="">Resedinta</label>
                <input type="text" placeholder='Introduceti Resedinta
Studentului' className='form-control'
                onChange={e => setResidence(e.target.value)}/>
            </div> modifica valoarea campului in baza de date
            <div className='mb-2'>
                <label htmlFor="">Anul</label>
                <input type="text" placeholder='Introduceti Anul de studiu'
className='form-control'
                onChange={e => setYear(e.target.value)}/>
            </div>

            <button className='btn btn-success'>Modificare</button>
        </form>
      </div>
    </div>
  )
}
```

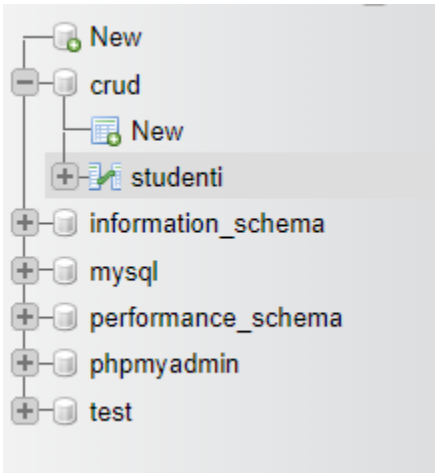
SERVER.JS

Vom folosi XAMP ca server pentru baza de date + web server



Accesand phpMyAdmin dashboard, putem vedea DB serverul si crea baza de date **crud** (numele previne de la create read update dele si este o preferinta) si a tabelului **student**





```
const express = require("express"); //preluam modelele necesare
const cors = require("cors");
const mysql = require("mysql");

const app = express();
app.use(express.json());
app.use(cors());

//Cream conexiunea cu baza de date
const db = mysql.createConnection ({
    host: "localhost",
    user:"root",
    password:"",
    database:"crud"
})

//Se preia url-ul principal si se va aplica queryul ce ne va arata toate datele
din DB. Acesta ne permite sa afisam date, comunicand cu frontendul si fisierul
“Student.js”
app.get("/", (req, res) =>{
    const sql = "SELECT * FROM studenti"; //studenti este numele bazei de date
din PHP My Admin
    db.query(sql, (err, data) => {
        if (err) return res.json ("Error");
        return res.json(data);
    });
});

//Se preia url-ul /create si se va aplica queryul ce ne va introduce date din DB.
Acesta ne permite sa introducem datele afisam date, comunicand cu frontendul si
fisierul “CreateStudent.js”
app.post("/create", (req, res) => {
    const sql ="INSERT INTO studenti (`Nume`, `Email`, `Resedinta`, `An`) VALUES
(?)"; //Cele din paranteza sunt campurile din tabelul "studenti"
    const values = [ //se vor prelua campurile din DB (nume, email, resedinta si
an pentru a se initializa
        req.body.nume,
        req.body.nume,
        req.body.email,
        req.body.resedinta,
        req.body.an
    ]
    db.query(sql, [values], (err, data) => {
        if(err) return res.json("Error"); //eroare daca este cazul
```

```
        return res.json(data); //ne va returna fisierul JSON pentru a putea
comunica cu axios pentru data fetching
    });


//Se preia url-ul /update si se va aplica queryul ce ne modifica datele din DB +
id-ul fiecarui rand. Acesta ne permite sa datele apo sa afisam datle modficate,
comunicand cu frontendul si fisierul “UpdateStudent.js”

app.put("/update/:id", (req, res) => {
//se aplica queryul pentru a modifica aceste campuri
    const sql ="update studenti set `Nume` = ?, `Email` = ?, `Resedinta` = ?,
`An` = ? where ID = ?";

    const values = [ //se vor prelua campurile din DB (nume, email, resedinta si
an pentru a se initializa
        req.body.nume,
        req.body.email,
        req.body.resedinta,
        req.body.an
    ]
    const id = req.params.id;
    db.query(sql, [...values, id], (err, data) => {
        if(err) return res.json("Error");
        return res.json(data); //ne va returna fisierul JSON pentru a putea
comunica cu axios pentru data fetching

    });
});

//Se preia url-ul /student si se va aplica queryul ce ne sterge datele din DB +
id-ul fiecarui rand. Acesta ne permite sa datele apo sa afisam datle modficate,
comunicand cu frontendul si fisierul “Student.js”

app.delete("/student/:id", (req, res) => {
    const sql ="DELETE FROM studenti WHERE ID = ?";
    const id = req.params.id;
    db.query(sql, [id], (err, data) => {
        if(err) return res.json("Error");
        return res.json(data);
    });
});

app.listen(8081, () => {
    console.log("server is up"); //afiseaza prin consola daca serverul e UP
```

```
})
```