

**Health Monitor**

**Inginerie software si aplicatii in comunicatii de date**

**Profesor coordonator:**

**Ciobanu Cătălin**

**Participanți:**

**Gheorghe Ștefan – Responsabil documentație**

**Mînecuță Răzvan - Manager**

**Mureșan Vicențiu - Testare**

**Vlăsceanu Nicolae Ovidiu - Development**

**BRAȘOV, 2023**

Cuprins

[1 Optimizarea supravegherii pacienților care suferă probleme la nivelul inimii sau al plămânilor 4](#_Toc134206236)

[1.1 Sumar 4](#_Toc134206237)

[1.2 Obiective 5](#_Toc134206238)

[2 Componentele folosite pentru realizarea proiectului 6](#_Toc134206239)

[2.1 Modul senzor puls cardiac GY-MAX30100 6](#_Toc134206240)

[2.2 ESP32-WROOM-32 8](#_Toc134206241)

[2.3 Breadboard 11](#_Toc134206242)

[3 Arhitectura proiectului 11](#_Toc134206243)

[3.1 Diagrama electrica 11](#_Toc134206244)

[3.2 Schema electrical 12](#_Toc134206245)

[4 Mini Tutorial pentru folosire 13](#_Toc134206246)

[5 Tehnologii folosite pentru development 13](#_Toc134206247)

[5.1 Arduino IDE 13](#_Toc134206248)

[5.2 React.js 14](#_Toc134206249)

[5.3 Firebase 15](#_Toc134206250)

[5.4 Wifi.h 16](#_Toc134206251)

[6 Repository GitHub 17](#_Toc134206252)

[6.1 Generalitati 17](#_Toc134206253)

[6.2 Avantaje 17](#_Toc134206254)

[7 Testare 18](#_Toc134206255)

[8 Management 19](#_Toc134206256)

[8.1 Importanța Management-ului în realizarea unui proiect 19](#_Toc134206257)

[8.2 Jira Management Software 19](#_Toc134206258)

[9 Lecții invatate 20](#_Toc134206259)

[10 Concluzii 21](#_Toc134206260)

[11 Codul sursa 21](#_Toc134206261)

[11.1 Conectiune Wifi 21](#_Toc134206262)

[11.2 Header.jsx 27](#_Toc134206263)

[11.3 Introducere.jsx 29](#_Toc134206264)

[11.4 Rezultate.jsx 33](#_Toc134206265)

[12 Bibliografie 40](#_Toc134206266)

# Optimizarea supravegherii pacienților care suferă probleme la nivelul inimii sau al plămânilor

## Sumar

Ne aflăm în 2023, proaspăt ieșiți din pandemia COVID-19, începută pe data de 17 Noiembrie 2019, în Wuhan, China, care a omorât 6,829,605 de oameni din jurul lumii. Chiar și până în ziua de astăzi, lumea care a supraviețuit acestei boli a rămas cu probleme pulmonare și de inimă. Astfel, a devenit din ce în ce mai importantă monitorizarea stării de sănătate și menținerea unui stil de viață sănătos. Un proiect care poate ajuta la aceste scopuri este Health Monitor.

Health Monitor este un proiect care utilizează un modul Arduino, un ESP32 și un pulsoximetru pentru a monitoriza nivelul de oxigen din sânge (SpO2) și ritmul cardiac al utilizatorului. Acest proiect poate fi construit pentru a monitoriza starea de sănătate a utilizatorilor și pentru a oferi alerte atunci când valorile de SpO2 sau pulsul scad sub anumite praguri critice.

Pulsoximetrul este un dispozitiv medical non-invaziv care poate fi atașat la degetul utilizatorului și care măsoară nivelul de oxigen din sânge, numit SpO2, precum și ritmul cardiac al acestuia. Modulul Arduino poate fi programat să citească valorile de SpO2 și pulsul utilizatorului prin intermediul senzorului din pulsoximetru și să le trimită către modulul ESP32. Acesta poate fi apoi conectat la un dispozitiv mobil sau la o platformă cloud pentru a fi monitorizate și înregistrate.

În plus, proiectul Health Monitor poate fi extins prin adăugarea altor senzori sau module, cum ar fi senzorii de temperatură sau de umiditate. Acest lucru poate ajuta la monitorizarea altor aspecte ale sănătății, cum ar fi nivelul de hidratare sau temperatura corpului.

Proiectul poate fi util pentru persoanele care doresc să-și monitorizeze starea de sănătate sau pentru cei care au afecțiuni medicale. De asemenea, poate fi folosit pentru a monitoriza sportivii și pentru a preveni apariția unor probleme de sănătate în timpul antrenamentelor.

În concluzie, proiectul Health Monitor poate fi un instrument util pentru monitorizarea stării de sănătate și menținerea unui stil de viață sănătos, mai ales în contextul actual, proaspăt ieșiți din pandemia de COVID-19.

## Obiective

Obiectivele pe care noi le propunem prin realizarea acestui proiect sunt următoarele:

* evidențierea necesității monitorizării pacienților cu probleme pulmonare sau de inimă, sau a sportivilor în timpul antrenamentelor
* evidențierea și crearea unor statistici/grafice pe termen scurt și îndelungat ale fiecărui pacient, date care sunt stocate în cloud
* implementarea unui software a sistemului pentru identificarea fiecărui pacient cu ajutorul datelor personale
* soluție hardware în timp real și software pentru trimiterea datelor despre pulsul pacientului, în cazul în care pulsul scade/crește sub valoarea unui prag critic să fie anunțat atât pacientul cât și doctorul care se ocupă de acesta printr-un SMS
* implementarea unor grafice zilnice pentru fiecare pacient din oră în oră pentru a putea fi analizate de către un doctor specializat.

# Componentele folosite pentru realizarea proiectului

## Modul senzor puls cardiac GY-MAX30100

Senzorul GY-MAX30100 este un senzor de oximetrie de tip pulsoximeter, care poate fi utilizat pentru a măsura nivelul de oxigen din sânge (SpO2) și ritmul cardiac. Acest senzor poate fi integrat în diverse aplicații, cum ar fi dispozitive medicale portabile, brățări fitness sau ceasuri inteligente.

Senzorul GY-MAX30100 utilizează tehnologia reflectometrică pentru a măsura nivelul de oxigen din sânge. Acesta emite o lumină infraroșie și roșie prin intermediul unui LED și măsoară cantitatea de lumină reflectată de la piele. În funcție de nivelul de oxigen din sânge, hemoglobina din sânge absoarbe mai mult sau mai puțină lumină infraroșie, ceea ce poate fi detectat de senzor. Senzorul poate detecta, de asemenea, fluctuațiile în intensitatea luminii reflectate, care sunt generate de pulsul cardiac al utilizatorului.

Senzorul GY-MAX30100 are o interfață I2C pentru comunicarea cu un microcontroller, cum ar fi Arduino. Acesta poate fi setat pentru a emite date de SpO2 și puls printr-un singur registru de control și poate fi configurat pentru a efectua măsurători continue sau pentru a măsura doar la cerere. Senzorul este, de asemenea, dotat cu un filtru digital pentru a elimina zgomotul și interferențele.

Specificații:

* + Tensiune alimentare: 3 – 5 VDC
  + Aplicații: Pulsoximeter (SpO2), Heart rate monitor
  + Sticlă acoperire senzor optic
  + Pini: GND, SCL, VIN, SDA, INT
  + Curent LED și rată refresh programabile pentru optimizarea consumului
  + Consum curent în mod oprit: 0.7 μA Tip
  + Consum monitorizare puls: <1mW
  + Output de date rapid
  + Dimensiuni mm: 12.7 x 12.7mm
  + Rezistență la interferențe date de mișcare
  + Temperatura de operare: -40°C - +85°C

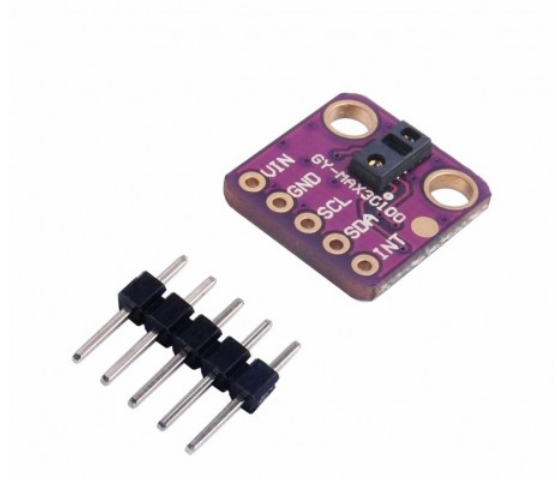


Figura 2.Imagine pulsoximetru GY-MAX30100 fata [1]

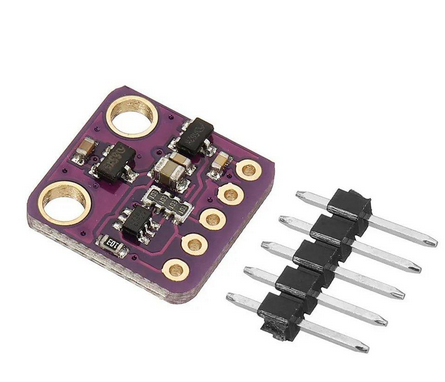


Figura 3.Imagine pulsoximetru GY-MAX30100 spate[1]

În figura 2 și figura 3 este prezentat modulul pulsoximetru-lui GYMAX30100, cu ajutorul căruia am măsurat pulsul pacientului, pozat pe ambele părți ale acestuia.

## ESP32-WROOM-32

ESP32-WROOM-32 este un modul de dezvoltare Wi-Fi și Bluetooth dual-core bazat pe chipset-ul ESP32 al companiei Espressif. Acesta este unul dintre cele mai populare module din gama ESP32, datorită puterii de procesare, a conectivității Wi-Fi și Bluetooth și a versatilității sale.

Acest modul poate fi programat cu ușurință cu ajutorul unui mediu de dezvoltare integrat (IDE), cum ar fi Arduino IDE sau platforma de dezvoltare ESP-IDF (Espressif IoT Development Framework). Modulul este dotat cu o serie de pini GPIO care permit conectarea la o varietate de componente și senzori, cum ar fi senzori de temperatură, accelerometre, LED-uri și multe altele.

ESP32-WROOM-32 este utilizat în diverse proiecte și aplicații IoT, cum ar fi sistem de supraveghere și control, automatizarea caselor inteligente, brățări de fitness sau dispozitive medicale portabile. Acesta este apreciat pentru performanțele sale bune, conectivitatea puternică și versatilitatea în dezvoltarea de soluții IoT.

Specificatii:

* + Procesor: Dual-core Tensilica LX6 cu frecvența de până la 240MHz
  + Memorie flash: 4MB
  + Memorie SRAM: 520KB
  + Modulul suportă conexiune WI-Fi 802.11 b/g/n
  + Suportă Bluetooth
  + Tensiune de funcționare: 3.3V

Pentru acest proiect noi am folosit următoarele conexiuni ale pinilor:

* + Alimentările ambelor componente se face prin intermediul modului ESP32 de la orice sursă de curent prin Mini USB
  + SCL (Serial Clock) într-un puls oximetru este utilizat în comunicarea serială între microcontroler și senzorul de oximetrie. Acesta este un semnal de sincronizare care determină momentul trimiterii și recepționării datelor de la și către senzor.
  + SDA (Serial Data) într-un puls oximetru este utilizat în comunicarea serială între microcontroler și senzorul de oximetrie. Acesta este un semnal care transmite datele de la și către senzor.
  + GND este folosit pentru a conecta senzorul la pământul comun al circuitului, astfel încât măsurătorile să fie făcute relativ la aceeași referință de tensiune.

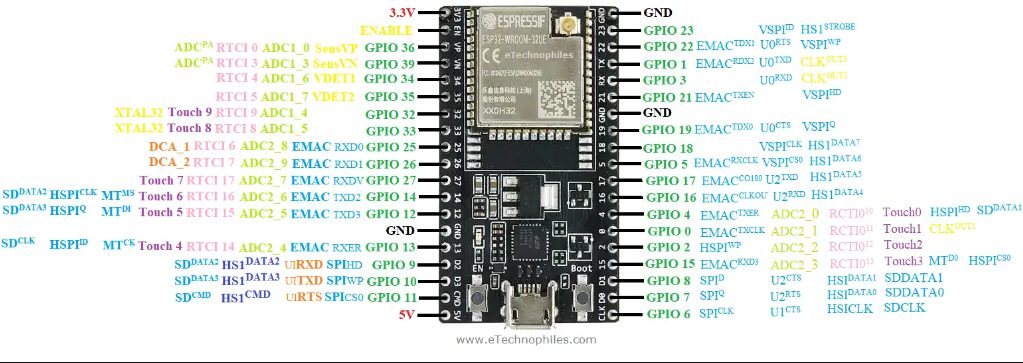


Figura 4.Functionalitati pini ESP32-WROOM-32s [2]

În figura 4, este prezentată o diagramă care ilustrează funcționalitatea fiecărui pin al microcontrolerului ESP32. Fiecare pin este etichetat cu numele său și este asociat cu o anumită funcționalitate. De exemplu, există pini desți pentru comunicarea serială (TX și RX), pini pentru interfețe de comunicație (SPI, I2C), pini pentru semnale digitale de intrare/ieșire (GPIO), pini pentru semnale analogice de intrare (ADC), precum și alte tipuri de pini. Diagrama este utilă pentru a înțelege cum poate fi utilizat fiecare pin într-un proiect specific, în funcție de cerințele funcționale.



Figura 5.Imagine ESP32-WROOM-32S

În figura 5 este prezentat modulul ESP32-WROOm-32S.

# Arhitectura proiectului

## Diagrama electrica

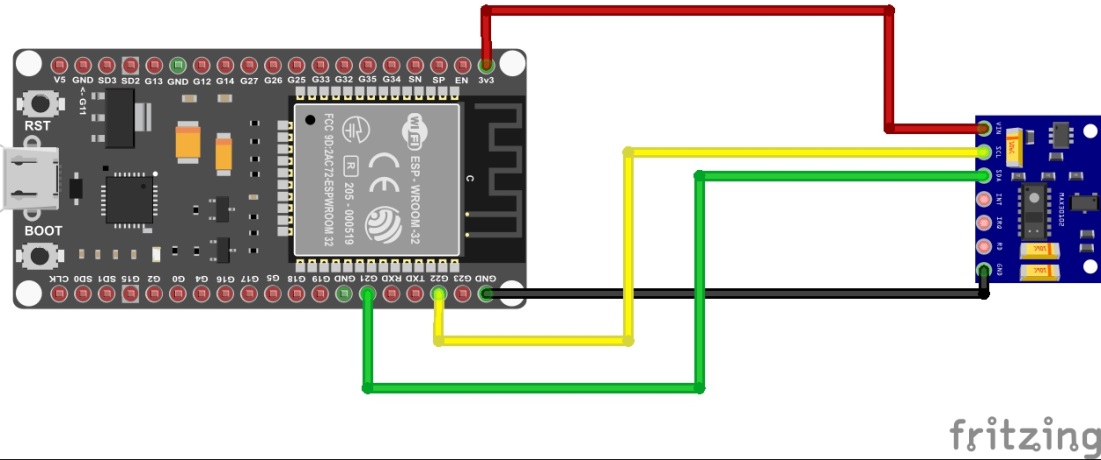


Figura 7. Diagrama electrica

În figura 7 este prezentată cum a fost realizată conectarea dintre ESP32 și pulsoximetru.

1. De la modulul ESP32 se conectează cu un fir roșu de 3,3V până la alimentarea pulsoximetrului la VIN
2. Firul negru este folosit pentru a lega ground-ul între cele două plăci.
3. De pe modulul ESP32 se face legătura cu un fir de culoare galben, de la GPIO022 la SCL (Serial Clock).

Ultimul fir, cel verde, face legătura de la GPIO21 până la SDA (Serial Data).

GPIO (General Purpose Input/Output) este un pin cu funcționalitate multiplă, care poate fi utilizat în funcție de necesitățile utilizatorului. De exemplu, acestea pot fi folosite pentru:

1. Intrare digitală: Pinul poate fi configurat pentru a citi semnale digitale de la alte dispozitive periferice, cum ar fi senzorii sau butoanele.
2. Ieșire digitală: Pinul poate fi configurat pentru a transmite semnale digitale către alte dispozitive periferice, cum ar fi LED-uri sau motoare.
3. Comunicare I2C: Pinul poate fi utilizat pentru a controla dispozitive periferice prin protocolul I2C (Inter-Integrated Circuit), care este un protocol de comunicație serială pentru dispozitivele electronice.

## Diagrama functionala a modulului GY-MAX30100

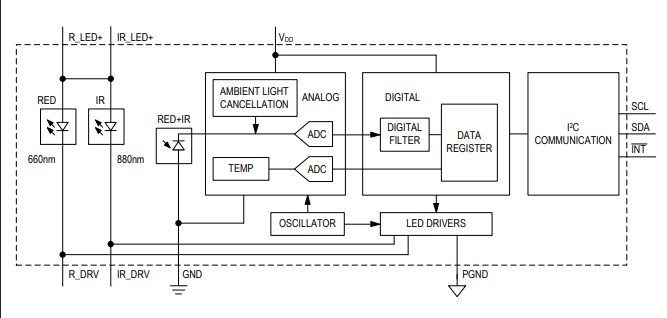
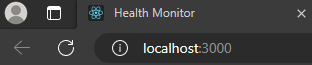


Figura 8 Diagrama functionala a modulului gy-max30100[3]

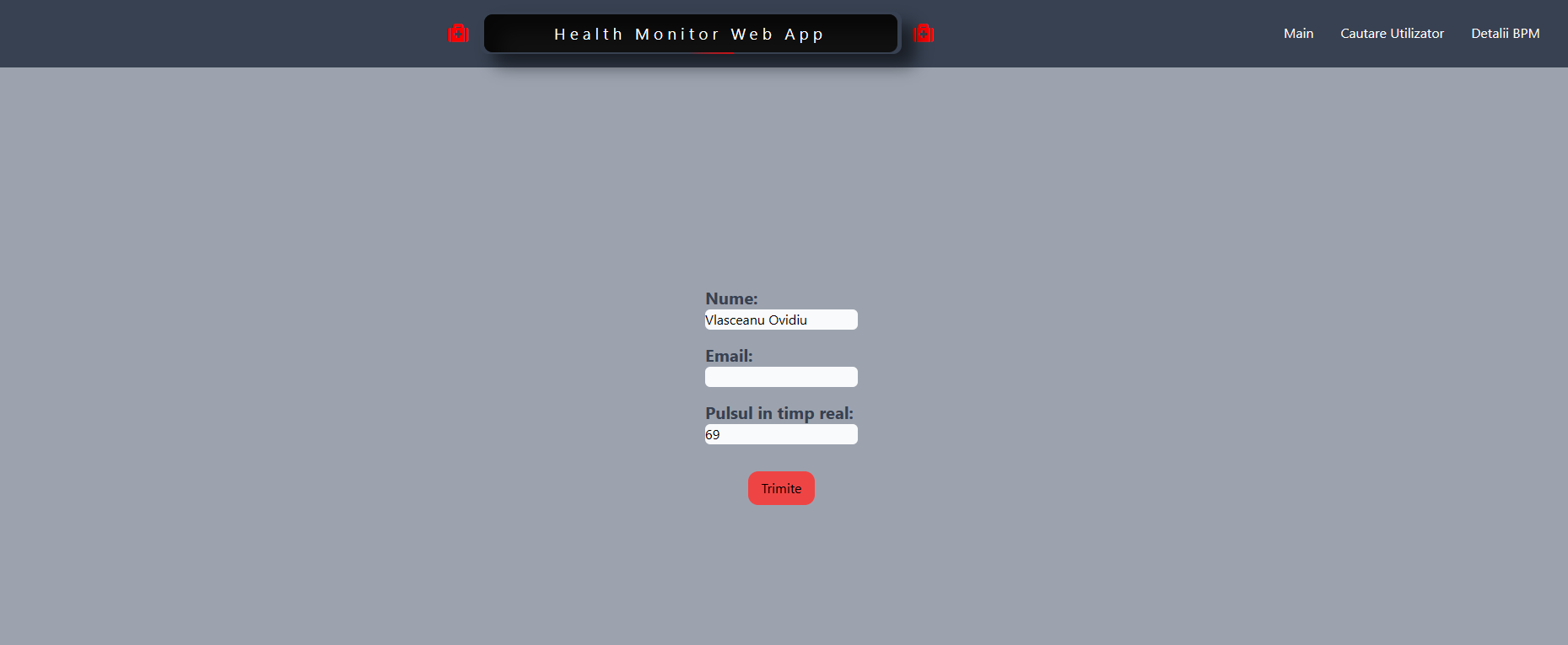
În figura 8 este reprezentată diagrama funcțională a modulului GY-MAX30100, care include o diodă emițătoare de lumină (LED) și un fotodetector care sunt plasați în partea de sus a modulului și care formează o pereche de diode emițătoare-fotodetector. Dioda LED emite o lumină roșie și infraroșie, care este absorbită de hemoglobina din sânge. Fotodetectorul măsoară intensitatea luminii care a trecut prin deget sau altă parte a corpului și care ajunge la fotodetectorul plasat pe partea opusă a LED-ului. Datele colectate de fotodetector sunt apoi prelucrate și transformate în semnalul de puls și nivelul de oxigen din sânge utilizând algoritmi specifici

# Mini Tutorial pentru folosire

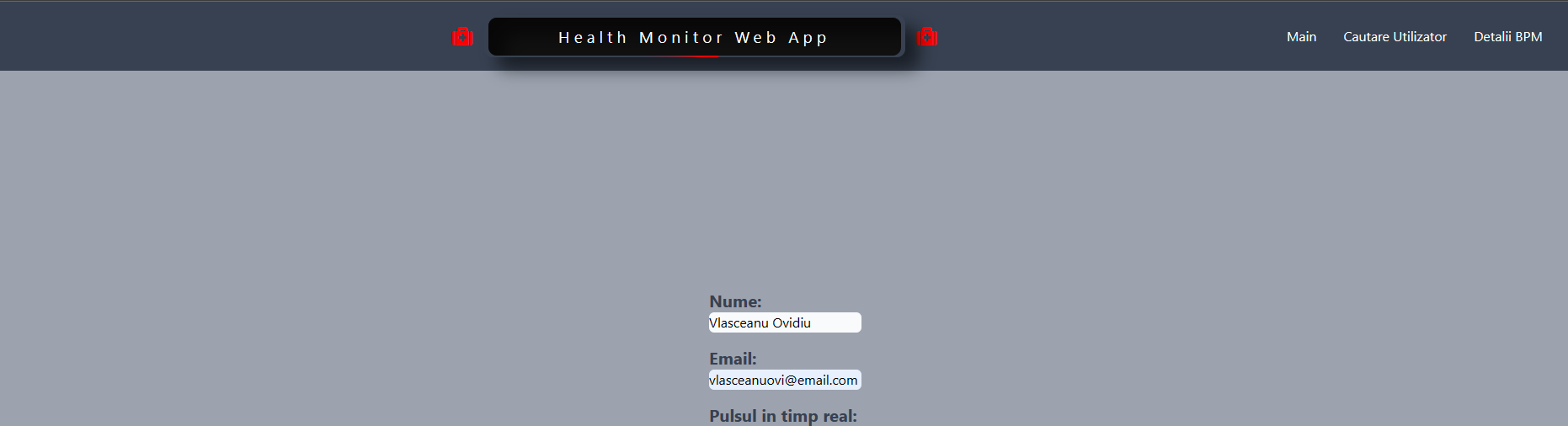
1.Se deschide un motor de cautare si se acceseaza pagina locala : localhost:3000.



*Figura 9 Adresa pentru acesarea paginii web*

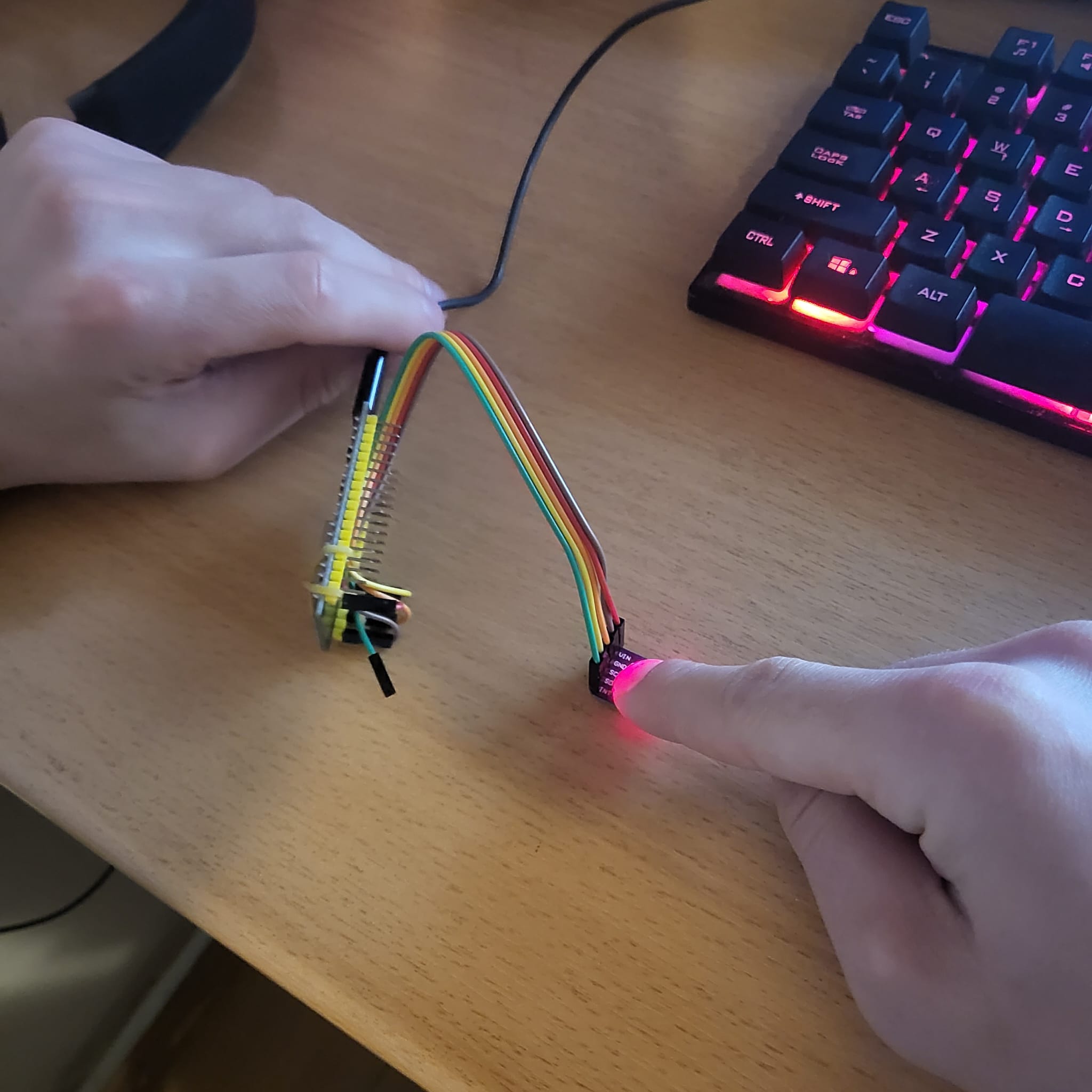
2.Se introduce numele pacientului(Nume+Prenume).

*Figura 10 Introducerea Numelui si a Prenumelui pacientului*

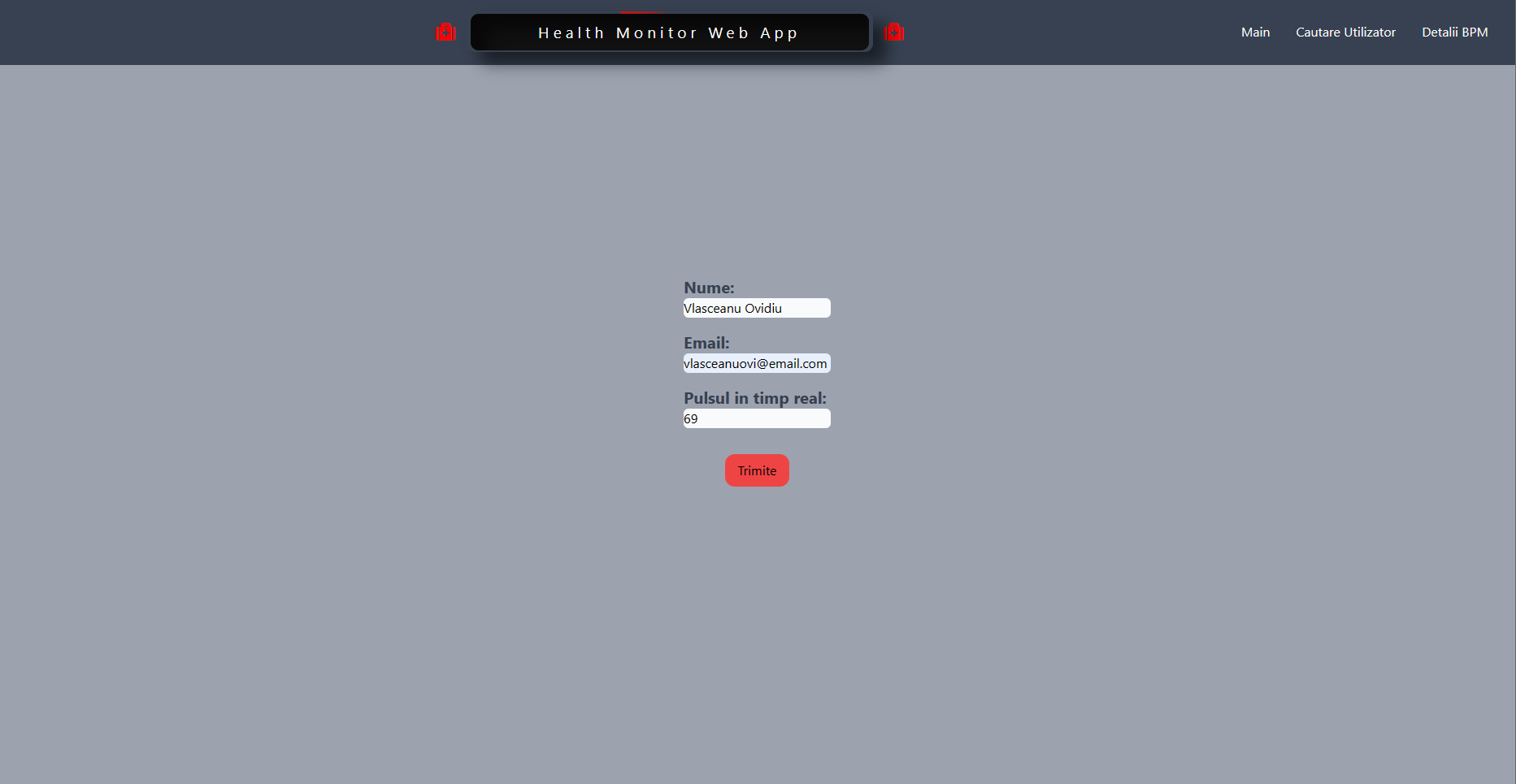
3.Se introduce email-ul pacientului.

*Figura 11 Introducerea email-ului pacientului*

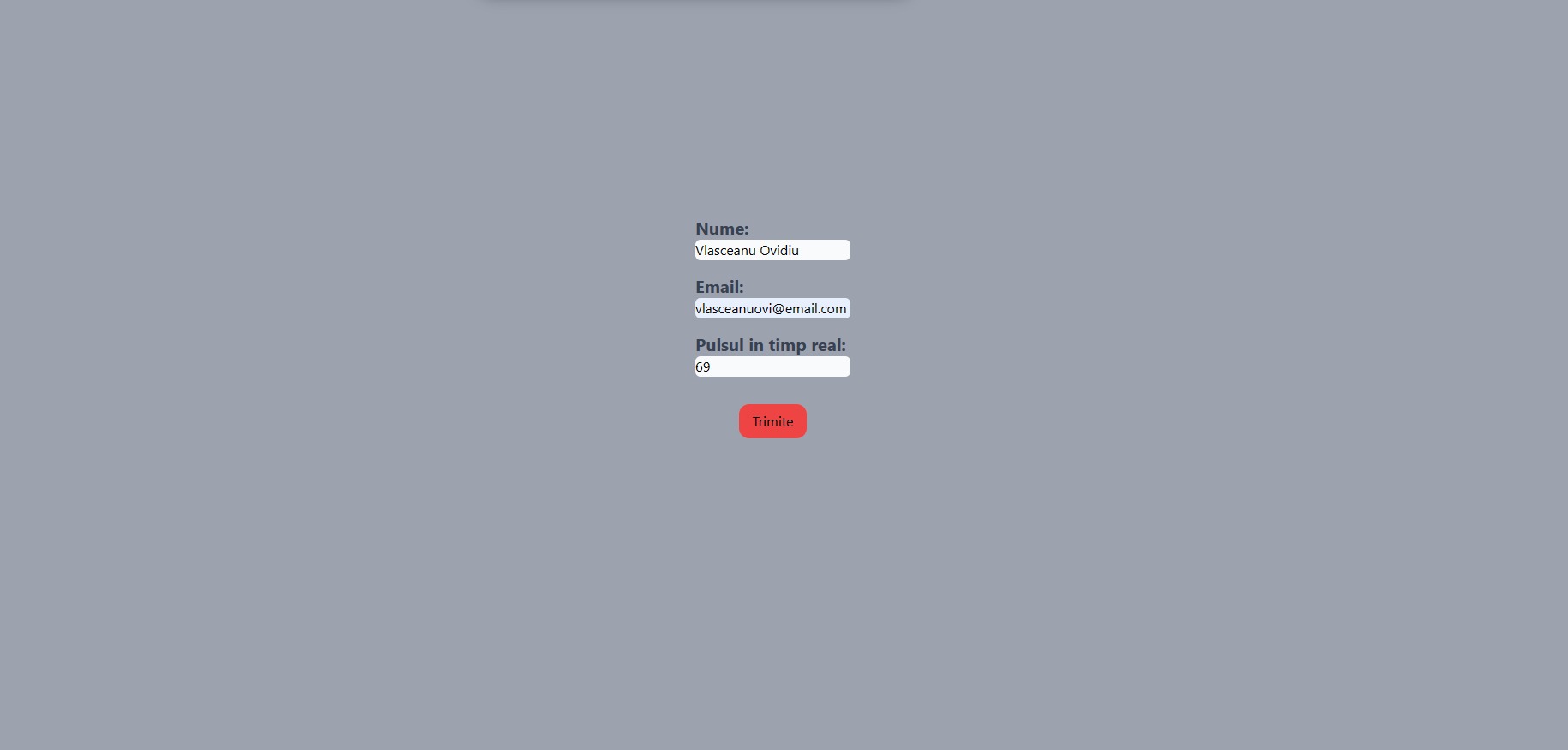
4.Se aseaza degetul pacientului pe pulsoximetru pentru cateva secunde,iar in input-ul “Nivelul de oxigen” se va afisa valoarea preluata de senzor.

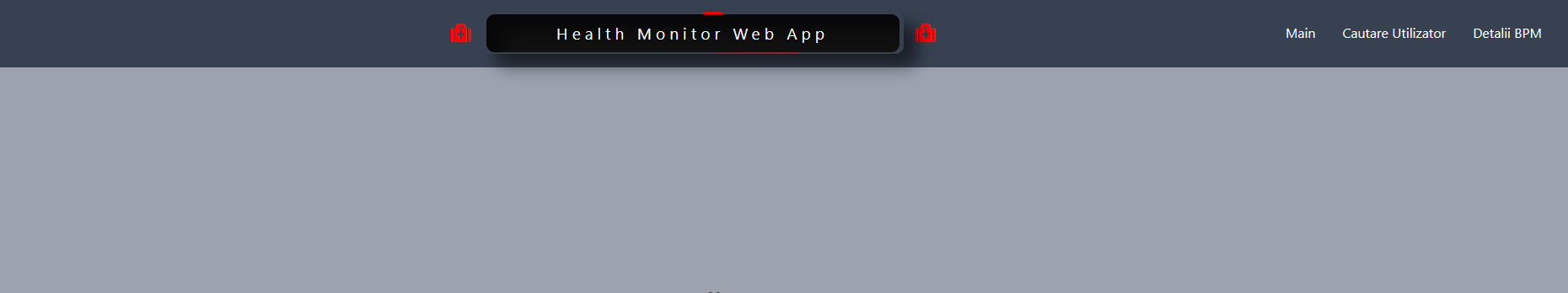


*Figura 12 Asezarea degetului pe pulsoximetru pentru preluarea oxigenuluidin sange si a pulsului*

*Figura 13 Valoarea preluata de catre senzor*

5.Se apasa pe butonul **Trimite** pentru a inregistra noul utilizator sau pentru a introduce o noua valoare a pulsului si a oxigenului din sange pentru un pacient deja inregistrat in baza de date.

*Figura 14 Butonul “Trimite” care inregistreaza in baza de date valoarea preluata de catre senzor*

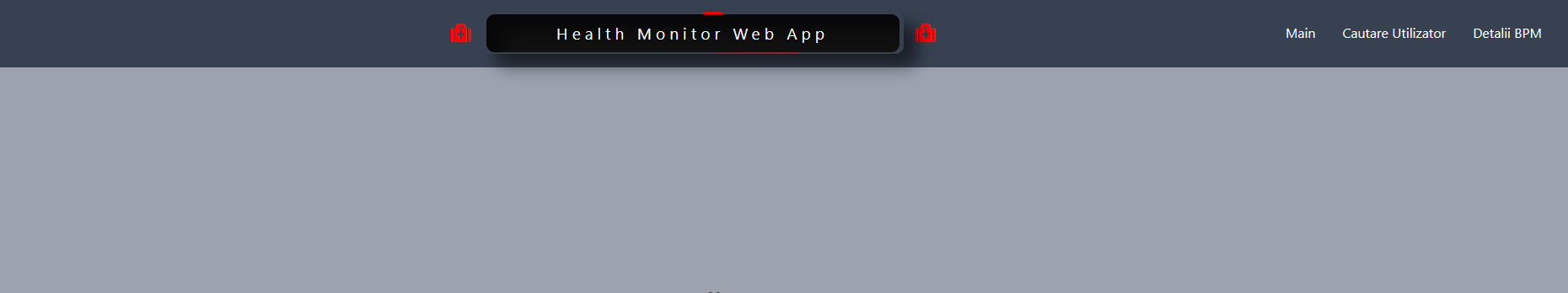
6.Pentru a putea vizualiza toate valorile care au fost inregistrate pentru un pacientu se face click pe tab-ul “Cautare Utilizator”.

*Figura 15 Tab-ul “Cautare Utilizator”*

7.In bara de search se cauta pacientul dorit,apoi sunt afisate datele acestuia alaturi de graficul facut dupa inregistrarea a minim 2 valori.



*Figura 16 Detaliile despre pacientul cautat*.

8.Pentru a verifica daca pulsul pacientului are o valoare buna se poate apasa pe tab-ul “Detalii BPM”.

*Figura 17 Tab-ul Detalii BPM*

# ****Tehnologii folosite pentru development****

## Arduino IDE

ArduinoIDE este un mediu de dezvoltare integrat (IDE) gratuit și open-source, folosit pentru programarea și dezvoltarea platformelor de prototipare hardware bazate pe placa Arduino. Acesta oferă o interfață simplă și intuitivă pentru programarea și depanarea codului, utilizând un limbaj de programare ușor de învățat și de folosit, numit Wiring. Cu ajutorul ArduinoIDE, utilizatorii pot crea proiecte interactive și experimentale care implică controlul diverselor senzori și actuatori, cum ar fi LED-uri, motoare, senzori de temperatură și umiditate, senzori de lumină, accelerometre și multe altele. Aceste proiecte pot fi conectate la o varietate de platforme de calcul, inclusiv calculatoare, tablete și smartphone-uri, pentru a oferi interacțiunea utilizatorului.

ArduinoIDE oferă, de asemenea, o bibliotecă extinsă de cod predefinit pentru a face mai ușoară dezvoltarea proiectelor complexe. Utilizatorii pot descărca și instala noi biblioteci și exemple de cod din comunitatea Arduino, care oferă o mulțime de resurse și suport pentru începători și avansați.

Un alt avantaj al ArduinoIDE este că este compatibil cu mai multe plăci Arduino, inclusiv Arduino Uno, Arduino Nano și Arduino Mega, printre altele. Acest lucru oferă utilizatorilor posibilitatea de a lucra cu plăci diferite, în funcție de cerințele proiectului lor.

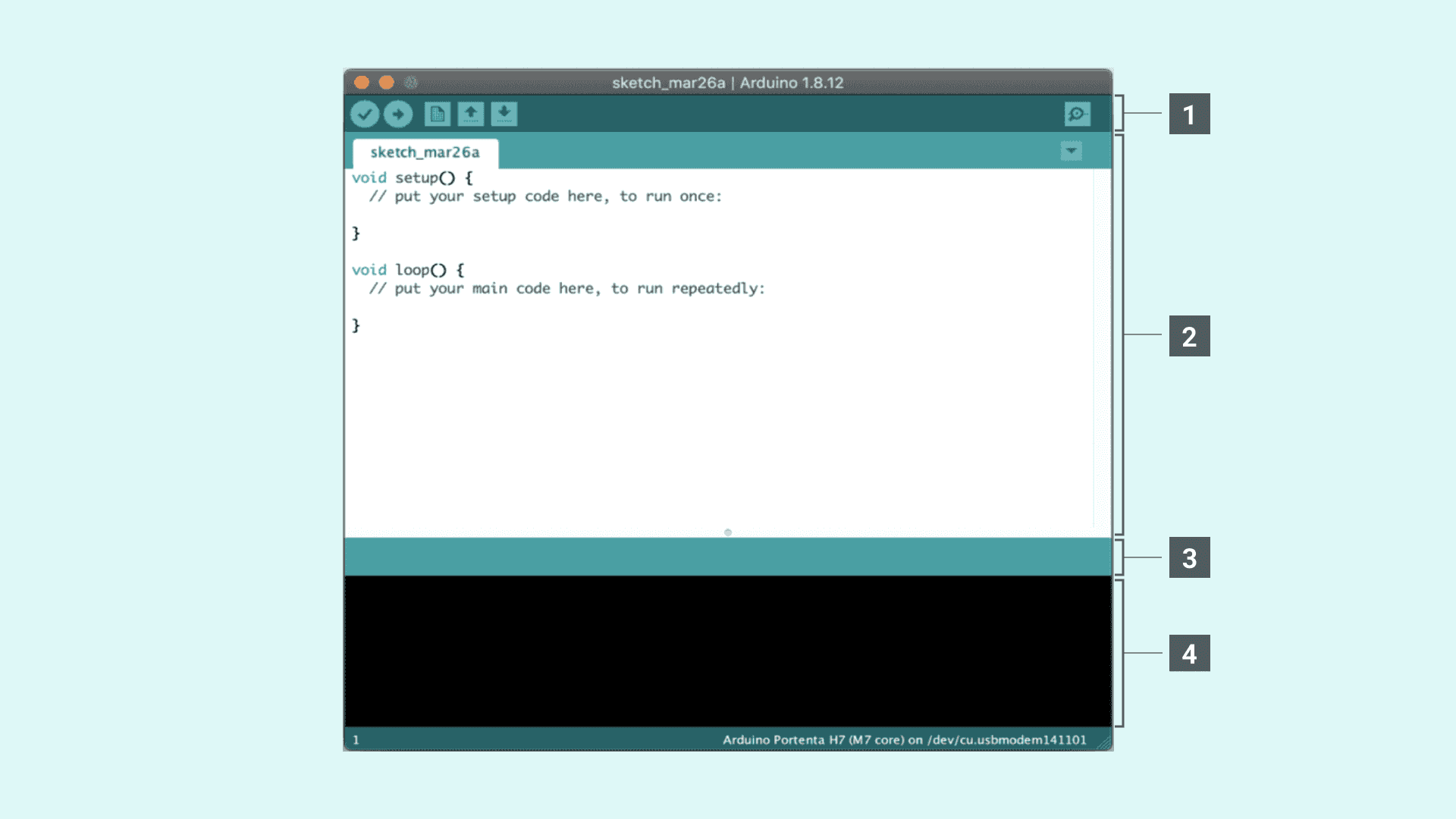


Figura 18 Arduino IDE Software[4]

În figura 9 este prezentată interfața software-ului folosit, ARDUINO IDE, cu ajutorul acestuia am testat și calibrat pulsoximetrul.

## React.js

React.js este o bibliotecă open-source pentru construirea interfețelor utilizator în aplicații web, dezvoltată de Facebook. Aceasta este o bibliotecă JavaScript care utilizează un model de programare declarativ pentru a construi interfețe utilizator complexe din componente simple și reutilizabile.[5]

React.js este de obicei utilizat împreună cu alte tehnologii web, cum ar fi Redux, React Router, TypeScript și multe altele pentru a oferi funcționalități extinse pentru dezvoltatorii de aplicații web.

Cu ajutorul bibliotecii React.js, am putut crea site-ul web al acestui proiect prin care putem căuta rezultatele pacienților după numele acestora, pentru a vizualiza nivelul de oxigen al acestora și data când a fost înregistrat rezultatul.

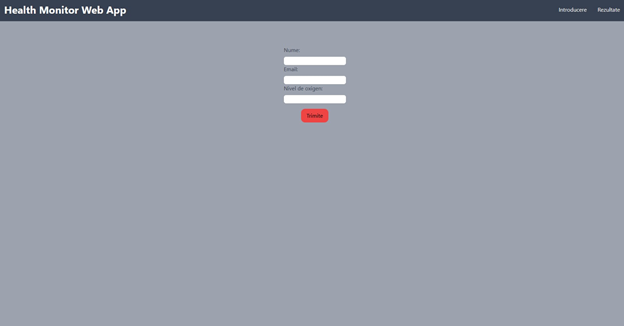


Figura 19 Interfata Web creeata cu React.js

În figura 9 este prezentată interfața web care a fost creată cu React.js, iar prin intermediul acesteia putem adăuga utilizatori, înregistra pulsul pacienților și oxigenul din sânge, urmând ca după să se poată vizualiza rezultatul alături de un grafic după înregistrarea mai multor valori ale pulsului unui pacient.

Am ales React.js pentru a crea interfața web al acestui site deoarece acesta utilizează un model de actualizare optimizat numit Virtual DOM, care oferă o viteză și performanță a actualizarilor de interfață a utilizatorilor față de alte framework-uri. Un alt avantaj al lui React.js sunt componentele reutilizabile, lucru care face ca dezvoltarea și întreținerea aplicațiilor să fie mai ușoară și mai rapidă. Iar ultimul aspect pe care l-am luat în considerare atunci când am ales React.js este integrarea cu alte tehnologii. React.js poate fi integrat cu alte tehnologii JavaScript, chiar și non-JavaScript, cum ar fi TypeScript, Redux, iar dacă proiectul este unul mai complex și implică o multitudine de tehnologii, această bibliotecă este o alegere bună.

În general, React.js este o alegere bună pentru dezvoltatorii care doresc să construiască interfețe utilizator rapide și performante, care sunt ușor de înțeles și de întreținut, și care pot fi integrate cu alte tehnologii pentru a construi aplicații complexe.

## Firebase

Firebase este o platformă de dezvoltare a aplicațiilor mobile și web, oferită de Google. Platforma oferă o gamă largă de servicii, inclusiv autentificare, bază de date în timp real, stocare de fișiere, notificări push și multe altele.

Firebase este cunoscută pentru ușurința cu care poate fi integrată în aplicații și pentru faptul că necesită puțină experiență în administrarea serverelor. Acest lucru face ca dezvoltarea aplicațiilor să fie mai rapidă și mai eficientă.

Firebase oferă API-uri pentru a accesa baza de date de la distanță. Acest lucru înseamnă că poți scrie cod care să acceseze baza de date Firebase, indiferent de locația ta fizică. Există API-uri disponibile pentru mai multe limbaje de programare, inclusiv JavaScript, Java, Objective-C și Swift.[7]

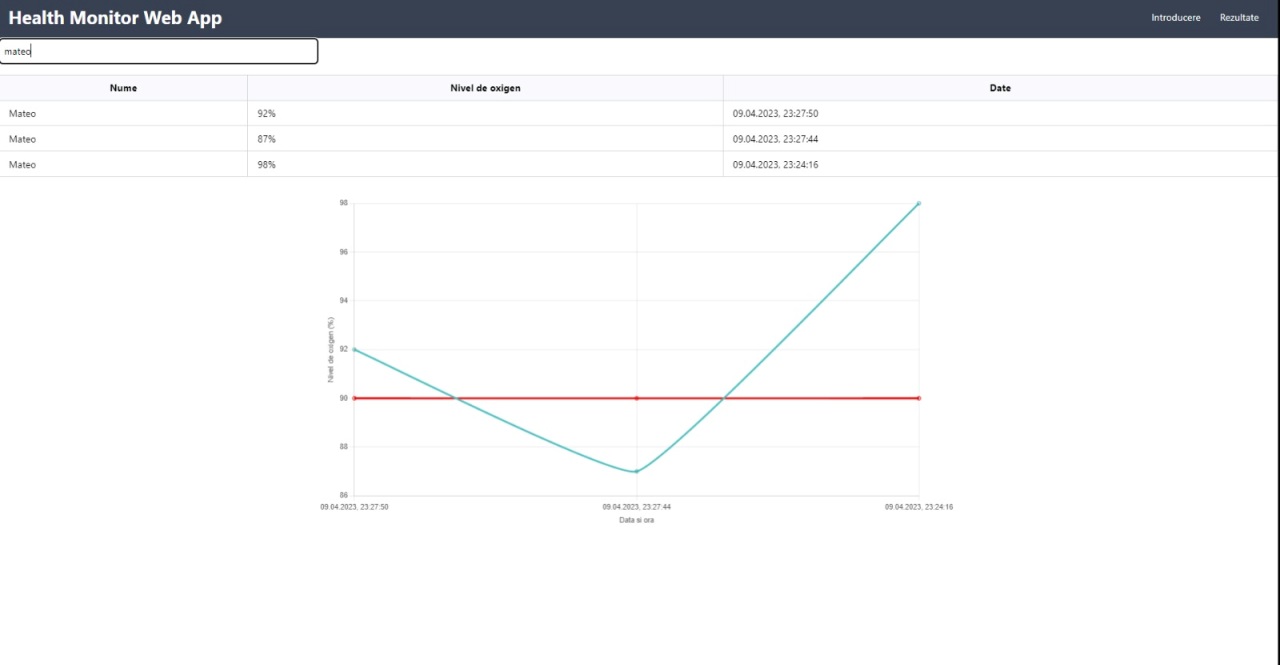


Figura 20 Interfata bazei de date

În figura 10, este prezentată interfața bazei de date, unde se pot căuta pacienți și vizualiza pulsul acestora alături de un grafic cu o medie a valorilor înregistrate.

În momentul în care se introduce în bara de căutare a site-ului web, se vor afișa toți pacienții cu numele introdus, alături de nivelul de oxigen al acestora și data când a fost înregistrată această informație. Pe baza acestor rezultate, se va genera un grafic cu nivelul de oxigen și ora.

## Wifi.h

WiFi.h este o bibliotecă din Arduino IDE utilizată pentru a permite interacțiunea cu modulele WiFi, cum ar fi ESP32 cel folosit de catre noi la acest proiect. Această bibliotecă conține funcții pentru inițializarea și configurarea conexiunii WiFi, precum și pentru trimiterea și primirea de date prin această conexiune.

Biblioteca WiFi.h este foarte utilă pentru dezvoltarea de aplicații care necesită comunicare fără fir, cum ar fi aplicațiile IoT (Internet of Things) sau controlul dispozitivelor prin intermediul unei rețele WiFi. Aceasta permite ca dispozitivele să comunice între ele sau să acceseze resursele de pe internet prin intermediul unei rețele WiFi, permițând astfel crearea unor aplicații complexe și conectate.

Pentru a folosi biblioteca WiFi.h în Arduino IDE, trebuie să se includă biblioteca prin intermediul comenzii "#include <WiFi.h>" în codul sursă al proiectului.



Figura 21 Introducerea librariei Wifi.h în proiect

În figura 11 se poate vedea cum a fost introdusă biblioteca wifi.h, această bibliotecă fiind utilizată în proiectele ce implică conexiunea la rețele WiFi și este inclusă în platforma de dezvoltare Arduino IDE.

# Repository GitHub

## Generalitati

GitHub este o platformă de dezvoltare software bazată pe sistemul de control al versiunilor Git. Aceasta permite dezvoltatorilor să colaboreze la proiecte și să își împărtășească codul sursă.

Un repository Git este un spațiu de stocare în care se găsește întregul istoric al unui proiect de dezvoltare software. Acesta este format dintr-o serie de fișiere și directoare care conțin codul sursă, documentația și alte fișiere relevante pentru proiectul respectiv.

Repository-urile Git sunt gestionate prin intermediul comenzilor Git, care permit utilizatorilor să monitorizeze, să modifice și să distribuie conținutul dintr-un repository.

Un repository Git poate fi găzduit pe o varietate de platforme, inclusiv GitHub, Bitbucket și GitLab. Aceste platforme oferă un set de instrumente și funcționalități care permit dezvoltatorilor să colaboreze în mod eficient și să împărtășească codul sursă cu alți dezvoltatori sau cu comunități mai largi.

## Avantaje

Pentru a partaja proiectul și dezvoltarea progresivă a acestuia, am ales să folosim GitHub datorită mai multor avantaje pe care acesta le oferă.

1. GitHub oferă un sistem de control al versiunilor tuturor proiectelor, lucru care permite dezvoltatorilor implicați în proiect să urmărească fiecare modificare făcută asupra acestuia și în cazul în care sunt probleme după o anumită modificare, să se poată reveni la versiunea anterioară al proiectului.
2. Comunitatea GitHub este o comunitate plină de dezvoltatori, lucru care face mai ușoară munca, deoarece aceștia pot să primească feedback la codul lor sursă, precum și să colaboreze între ei pentru a duce un proiect la bun sfârșit.
3. Accesibilitatea este un alt punct forte al celor de la GitHub, deoarece oricine are acces la internet poate accesa repository-ul pe care îl dorește.
4. Majoritatea dezvoltatorilor folosesc GitHub pentru a crea un portofoliu unde pot prezenta toate proiectele la care au lucrat în colaborare cu alți dezvoltatori sau chiar singur, adăugând și codul sursă și contribuțiile aduse la proiect.

# Testare

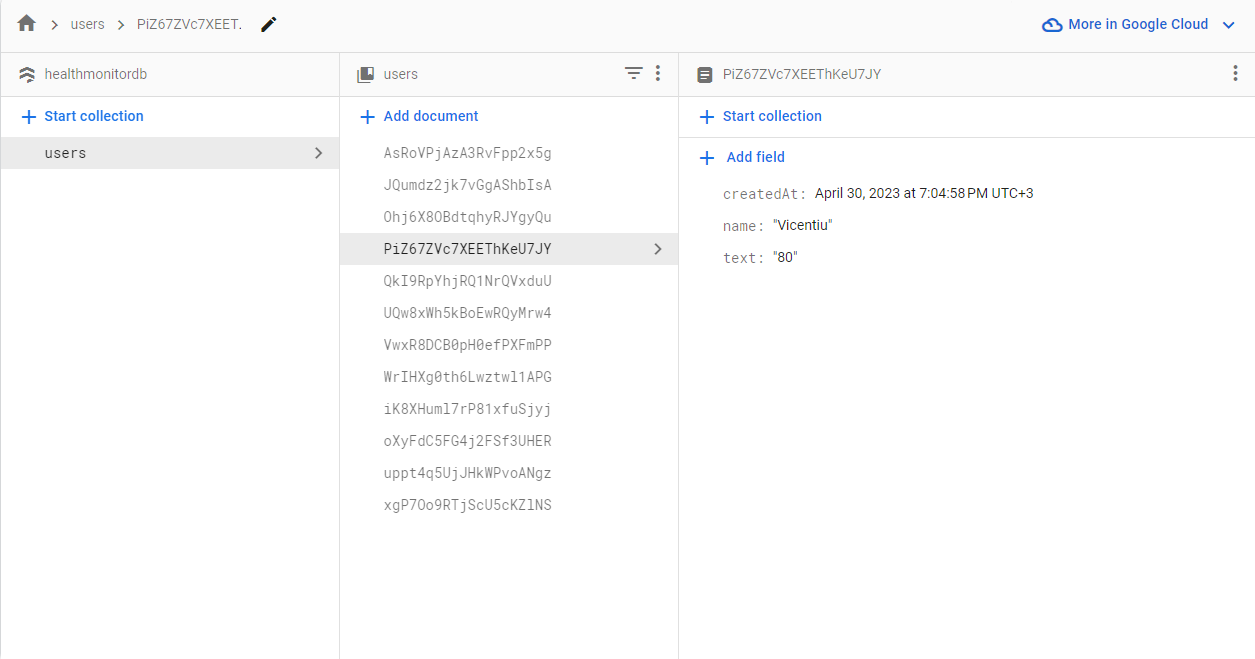
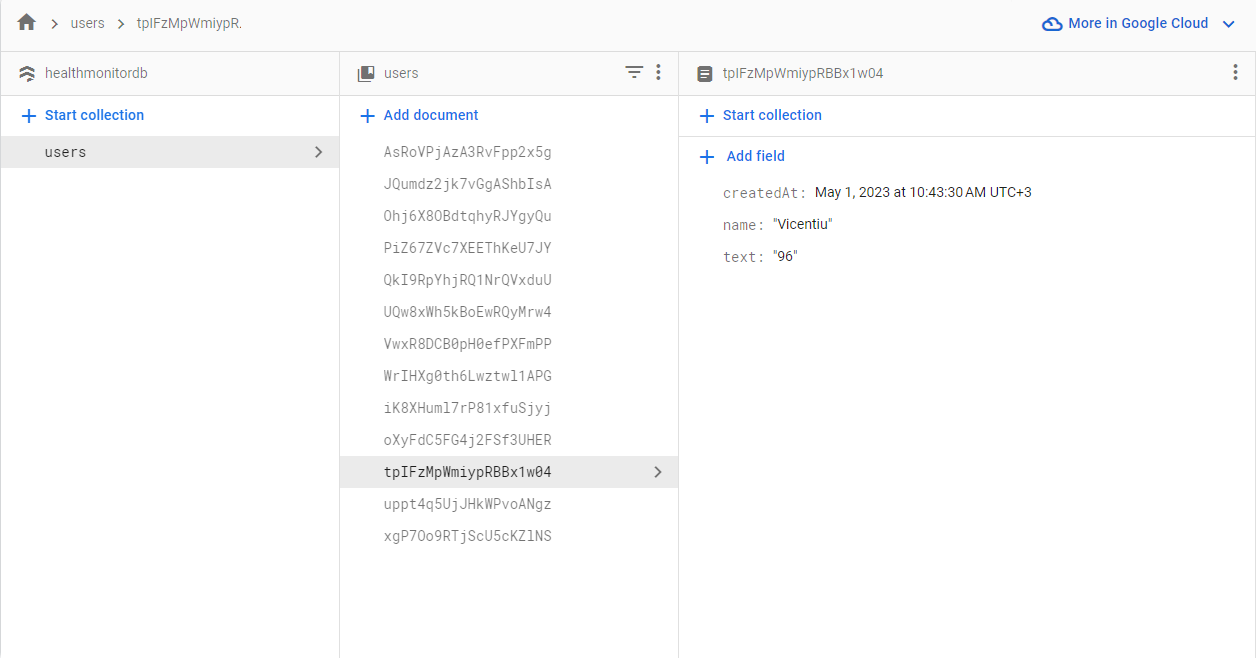
La acest capitol s-a dorit adăugarea unui nou utilizator prin intermediul interfeței create. S-a urmărit realizarea procesului de creare a unui nou utilizator în baza de date, urmând să fie afișat pe pagina web.

Figura 22 Baza de date inainte de adaugarea unui nou utilizator

Figura 23 Baza de date dupa adaugarea unui nou utilizator

# Management

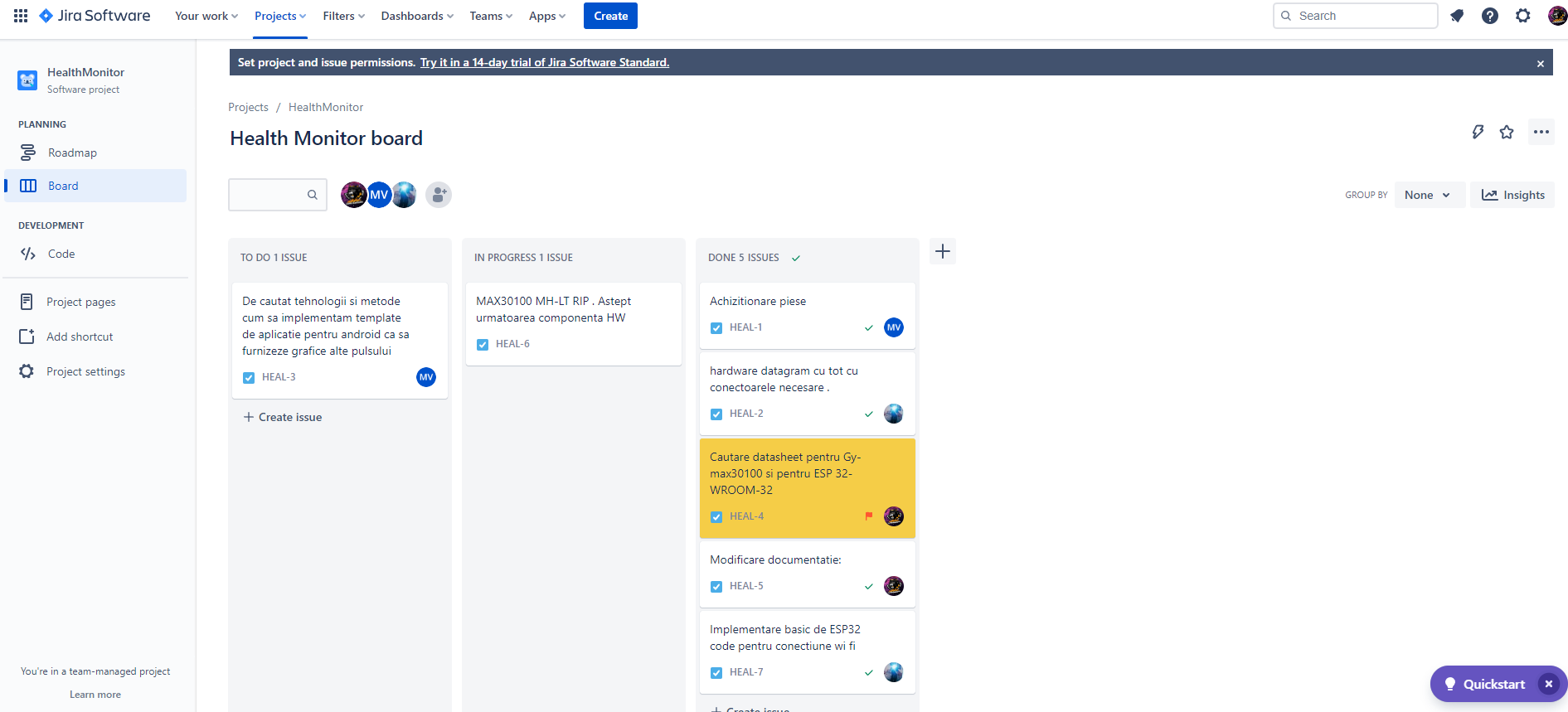
## Importanța Management-ului în realizarea unui proiect

Managementul unui proiect este esențial pentru a asigura că acesta este finalizat în timp util, cu resursele și costurile alocate. Un bun management de proiect implică stabilirea unui plan detaliat pentru proiect, care să includă obiectivele și scopul, resursele necesare, termene limită și un plan de acțiune pentru a ajunge la aceste obiective. Managementul eficient al proiectului implică, de asemenea, monitorizarea și urmărirea progresului, identificarea și abordarea potențialelor probleme și riscuri, și comunicarea clară și eficientă cu membrii echipei și cu toate părțile interesate.

## Jira Management Software

Jira este o platformă de management pentru proiecte, utilizată pe scară largă la multe întreprinderi pentru a gestiona proiecte de tip software și nu numai. Jira permite utilizatorilor să creeze sarcini de lucru, să monitorizeze progresul și să raporteze probleme în cazul în care apar în urma rezolvării unei sarcini.

Jira are mai multe funcții de bază precum

1. Crearea de sarcini și probleme, unde utilizatorii pot adăuga sarcini, cerințe pentru proiect, inclusiv informații detaliate asupra sarcinii, cât și termene limită și priorități.
2. Administratorii proiectului pot atribui utilizatorilor sarcini către membrii echipei.
3. Este posibilă monitorizarea progresului sarcinilor și problemelor cu un sistem integrat de rapoarte și grafice.
4. Cea mai importantă funcție de bază a programului Jira este colaborarea și comunicarea utilizatorilor cu membrii echipei prin intermediul comentariilor și actualizărilor.

*Figura 24 Sarcinile alocate pe Jira*

În figura 16 se poate observa câteva din sarcinile atribuite fiecărui utilizator din cadrul acestui proiect, stadiul în care se află sarcina și importanța acestora.

# Lecții invatate

Pe parcursul dezvoltării proiectului Health Monitor s-au folosit o multitudine de tehnologii, care au dus la acumularea unor noi informații.

1. Conectarea și implementarea codului modulului ESP32 în Arduino IDE.
2. Realizarea conexiunii modulului ESP32 la WiFi.
3. Colectarea datelor de la PulsOximetru și transmiterea acestora către o bază de date.
4. Crearea interfeței pentru adăugarea și vizualizarea rezultatelor fiecărui client.
5. Înțelegerea importanței bunei organizări în cadrul unui proiect unde participă mai multe persoane, fiecare având un rol diferit, cu ajutorul platformei Jira.
6. Acomodarea utilizării site-ului GitHub pentru partajarea informațiilor și vizualizarea noilor modificări făcute de către ceilalți utilizatori.

# Concluzii

Considerăm că proiectul "Health Monitor" are o aplicabilitate mare în viața modernă, fiind o aplicație cu avantaje în realizarea unei vieți mai bune pentru persoanele care suferă de probleme respiratorii sau cardiace.

După realizarea acestui proiect, putem asigura un trai mai bun persoanelor bolnave și o siguranță sporită asupra vieții, fără a trăi cu frica că în orice moment pot avea o scădere sau creștere drastică a pulsului, cu ajutorul graficelor și nimeni nu știe despre situația în care se află. Dacă ai asupra ta acest dispozitiv, ai siguranța că ești monitorizat permanent de către un doctor specialist și în orice moment, dacă ai o problemă, poți lua măsurile necesare.

În concluzie, acest proiect aduce beneficii înspre un trai mai bun și poate fi dezvoltat într-o aplicație mai complexă, care poate și înștiința cel mai apropiat doctor specialist de locația utilizatorului printr-un SMS sau e-mail, sau în cazuri extreme să sune direct la 112.

# Codul sursa

## Conexțiune Wifi

#include <Wire.h>

#include "MAX30105.h"

#include "spo2\_algorithm.h"

#include <WiFi.h>

#include <ThingSpeak.h>

MAX30105 particleSensor;

#define MAX\_BRIGHTNESS 255

#if defined(\_\_AVR\_ATmega328P\_\_) || defined(\_\_AVR\_ATmega168\_\_)

uint16\_t irBuffer[100]; //infrared LED sensor data

uint16\_t redBuffer[100]; //red LED sensor data

#else

uint32\_t irBuffer[100]; //infrared LED sensor data

uint32\_t redBuffer[100]; //red LED sensor data

#endif

int32\_t bufferLength; //data length

int32\_t spo2; //SPO2 value

int8\_t validSPO2; //indicator to show if the SPO2 calculation is valid

int32\_t heartRate; //heart rate value

int8\_t validHeartRate; //indicator to show if the heart rate calculation is valid

byte pulseLED = 11; //Must be on PWM pin

byte readLED = 13; //Blinks with each data read

long prevMillisSensor = 0;

int intervalSensor = 1000;

long prevMillisThingSpeak = 10000;

int intervalThingSpeak = 15000; // Minimum ThingSpeak write interval is 15 seconds

const char\* ssid = "402"; // your network SSID (name)

const char\* password = "faraparola"; // your network password

const long CHANNEL = 2132916;

const char \*WRITE\_API = "CIA2ZBO9VHDVS7BO";

WiFiClient client;

void setup() {

Serial.begin(115200); // initialize serial communication at 115200 bits per second:

pinMode(pulseLED, OUTPUT);

pinMode(readLED, OUTPUT);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.println("Connecting to WiFi...");

}

Serial.println("Connected to WiFi");

ThingSpeak.begin(client);

if (!particleSensor.begin(Wire, I2C\_SPEED\_FAST)) //Use default I2C port, 400kHz speed

{

Serial.println(F("MAX30105 was not found. Please check wiring/power."));

while (1);

}

Serial.println(F("Attach sensor to finger with rubber band. Press any key to start conversion"));

while (Serial.available() == 0); //wait until user presses a key

Serial.read();

byte ledBrightness = 60; //Options: 0=Off to 255=50mA

byte sampleAverage = 4; //Options: 1, 2, 4, 8, 16, 32

byte ledMode = 2; //Options: 1 = Red only, 2 = Red + IR, 3 = Red + IR + Green

byte sampleRate = 100; //Options: 50, 100, 200, 400, 800, 1000, 1600, 3200

int pulseWidth = 411; //Options: 69, 118, 215, 411

int adcRange = 4096; //Options: 2048, 4096, 8192, 16384

particleSensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate, pulseWidth, adcRange); //Configure sensor with these settings

}

void loop()

{

bufferLength = 100; //buffer length of 100 stores 4 seconds of samples running at 25sps

for (byte i = 0 ; i < bufferLength ; i++)

{

while (particleSensor.available() == false) //do we have new data?

particleSensor.check(); //Check the sensor for new data

redBuffer[i] = particleSensor.getRed();

irBuffer[i] = particleSensor.getIR();

particleSensor.nextSample(); //We're finished with this sample so move to next sample

Serial.print(F("red="));

Serial.print(redBuffer[i], DEC);

Serial.print(F(", ir="));

Serial.println(irBuffer[i], DEC);

}

maxim\_heart\_rate\_and\_oxygen\_saturation(irBuffer, bufferLength, redBuffer, &spo2, &validSPO2, &heartRate, &validHeartRate);

while (1)

{

//dumping the first 25 sets of samples in the memory and shift the last 75 sets of samples to the top

for (byte i = 25; i < 100; i++)

{

redBuffer[i - 25] = redBuffer[i];

irBuffer[i - 25] = irBuffer[i];

}

for (byte i = 75; i < 100; i++)

{

while (particleSensor.available() == false) //do we have new data?

particleSensor.check(); //Check the sensor for new data

digitalWrite(readLED, !digitalRead(readLED)); //Blink onboard LED with every data read

redBuffer[i] = particleSensor.getRed();

irBuffer[i] = particleSensor.getIR();

particleSensor.nextSample(); //We're finished with this sample so move to next sample

Serial.print(F("red="));

Serial.print(redBuffer[i], DEC);

Serial.print(F(", ir="));

Serial.print(irBuffer[i], DEC);

Serial.print(F(", HR="));

Serial.print(heartRate, DEC);

Serial.print(F(", HRvalid="));

Serial.print(validHeartRate, DEC);

Serial.print(F(", SPO2="));

Serial.print(spo2, DEC);

Serial.print(F(", SPO2Valid="));

Serial.println(validSPO2, DEC);

}

maxim\_heart\_rate\_and\_oxygen\_saturation(irBuffer, bufferLength, redBuffer, &spo2, &validSPO2, &heartRate, &validHeartRate);

if (validSPO2==1 and validHeartRate==1)

if (millis() - prevMillisThingSpeak > intervalThingSpeak) {

// Set the fields with the values

ThingSpeak.setField(1, spo2);

ThingSpeak.setField(2, heartRate);

// Write to the ThingSpeak channel

int x = ThingSpeak.writeFields(CHANNEL, WRITE\_API);

if (x == 200) {

Serial.println("Channel update successful.");

}

else {

Serial.println("Problem updating channel. HTTP error code " + String(x));

}

prevMillisThingSpeak = millis();

}

}

} [8]

## Header.jsx

import React, { useEffect } from 'react';

import { Link } from "react-router-dom";

import './header.css';

function Header() {

useEffect(() => {

document.body.style.overflow = 'hidden';

return () => {

document.body.style.overflow = 'unset';

}

}, []);

return (

<div className='flex bg-gray-700 justify-center items-center p-4 text-white'>

<div className='w-3/4 flex justify-center' style={{ marginLeft: '200px' }}>

<i className="fas fa-medkit" style={{ fontSize: '24px', width: '24px', height: '24px', color: 'red', marginRight:'1%', marginTop: '0.5%' , paddingTop: '6px'}}></i>

<div className='animation'></div>

<i className="fas fa-medkit" style={{ fontSize: '24px', width: '24px', height: '24px', color: 'red', marginLeft:'1%', marginTop: '0.5%' , paddingTop: '6px'}}></i>

</div>

<ul className='flex'>

<li className='px-4 cursor-pointer capitalize hover:scale-105 duration-200'>

<Link to="/">Main</Link>

</li>

<li className='px-4 cursor-pointer capitalize hover:scale-105 duration-200'>

<Link to="/rezultate">Cautare utilizator</Link>

</li>

</ul>

</div>

)

}

export default Header

## Introducere.jsx

import React, { useEffect, useState } from 'react'

import { addDoc, collection, serverTimestamp } from 'firebase/firestore'

import axios from 'axios'

import { db } from '../firebase'

import './Introducere.css'

function Introducere() {

const [name, setName] = useState('');

const [email, setEmail] = useState('');

const [message, setMessage] = useState(null);

const handleSubmit = async (e) => {

e.preventDefault();

const messageDB = {

text: message,

name: name,

createdAt: serverTimestamp(),

}

await addDoc(collection(db, 'users'),

messageDB

)

setName('');

setEmail('');

setMessage(null);

};

useEffect(() => {

const fetchData = async () => {

const response = await axios.get(

'https://api.thingspeak.com/channels/2132916/fields/1/last.json'

);

setMessage(response.data.field1);

};

fetchData();

}, []);

const isDisabled = name === '' || email === '' || message === '';

return (

<div className="w-full min-h-screen bg-gray-400 p-12">

<div className="flex flex-col p-6 font-bold items-center justify-center text-2xl">

<h1 className='mb-6'>Real time Pulse :</h1>

<iframe width="450" height="260" src="https://thingspeak.com/channels/2132916/widgets/643941"></iframe>

</div>

<form

onSubmit={handleSubmit}

className="flex flex-col p-6 items-center justify-center"

>

<label className="block mb-4">

<span className="text-gray-700 font-bold text-xl mb-2">

Nume: <br></br>

</span>

<input

type="text"

value={name}

onChange={(e) => setName(e.target.value)}

className="input-style"

/>

</label>

<label className="block mb-4">

<span className="text-gray-700 font-bold text-xl mb-2">

Email: <br></br>

</span>

<input

type="email"

value={email}

onChange={(e) => setEmail(e.target.value)}

className="input-style"

/>

</label>

<label className="block mb-4">

<span className="text-gray-700 font-bold text-xl mb-2">

Nivel de oxigen: <br></br>

</span>

<input

type="number"

onChange={(e) => setMessage(e.target.value)}

className="input-style"

placeholder="Nivel de oxigen"

value={message}

/>

</label>

<button

className="mt-4 px-4 py-2 bg-red-500 border-black rounded-xl"

type="submit"

disabled={isDisabled}

>

Trimite

</button>

</form>

</div>

);

}

export default Introducere

## Rezultate.jsx

import React, { useState, useEffect } from 'react';

import { db } from '../firebase';

import { collection, onSnapshot, orderBy, query, where } from 'firebase/firestore';

import { Line } from 'react-chartjs-2';

import { Chart, CategoryScale, LinearScale, LineController, PointElement, LineElement } from 'chart.js';

import './web.css';

Chart.register(CategoryScale, LinearScale, LineController, PointElement, LineElement);

function Rezultate() {

// Declare state variables for search query and search results

const [searchName, setSearchName] = useState('');

const [text, setText] = useState([]);

const [loading, setLoading] = useState(false);

// Define chart state

const [chartData, setChartData] = useState({});

// Define a function to handle search input changes

const handleSearchChange = (e) => {

setSearchName(e.target.value);

};

// Use useEffect to fetch search results from the database

useEffect(() => {

// Set loading state to true while fetching data

setLoading(true);

// Define a query to search for messages that match the search query

const q = query(

collection(db, 'users'),

where('name', '==', searchName),

orderBy('createdAt', 'desc')

);

// Subscribe to the query and update state when new data is received

const unsubscribe = onSnapshot(q, (querySnapshot) => {

const messages = [];

querySnapshot.forEach((doc) => {

messages.push({ id: doc.id, ...doc.data() });

});

setText(messages);

setLoading(false);

});

// Unsubscribe from the query when the component unmounts

return () => unsubscribe();

}, [searchName]);

// Use useEffect to update the chart data when the search results change

useEffect(() => {

// Define arrays to hold chart data

const labels = [];

const data = [];

// Loop through search results and add data to chart arrays

text.forEach((message) => {

labels.push(new Date(message.createdAt.toDate()).toLocaleString());

data.push(message.text);

});

// Define chart data

const chartData = {

labels: text.map((message) => new Date(message.createdAt.toDate()).toLocaleString()),

datasets: [

{

label: 'Nivel de oxigen',

data: text.map((message) => message.text),

fill: false,

borderColor: 'rgb(75, 192, 192)',

tension: 0.1

},

{

type: 'line',

label: 'Limita de siguranta',

data: Array(text.length).fill(90),

fill: false,

borderColor: 'red',

tension: 0.1

}

]

};

// Update chart data state

setChartData(chartData);

}, [text]);

const chartOptions = {

responsive: true,

scales: {

x: {

title: {

display: true,

text: 'Data si ora'

}

},

y: {

title: {

display: true,

text: 'Nivel de oxigen (%)'

}

}

},

plugins: {

title: {

display: true,

text: 'Nivel de oxigen in timp'

}

}

};

return (

<div className='flex flex-col'>

<input type="text" value={searchName} onChange={handleSearchChange} placeholder="Căutare după numele pacientului" className='search-bar' />

{loading && <div>Loading...</div>}

{!loading && text.length > 0 && (

<div className="">

<table className="table-auto border-collapse w-full">

<thead className="table-header">

<tr>

<th className="border border-gray-300 px-4 py-2">Nume</th>

<th className="border border-gray-300 px-4 py-2">Nivel de oxigen</th>

<th className="border border-gray-300 px-4 py-2">Date</th>

</tr>

</thead>

<tbody>

{text.map((message) => (

<tr key={message.id}>

<td className="border border-gray-300 px-4 py-2 capitalize">{message.name}</td>

<td className="border border-gray-300 px-4 py-2">{message.text}%</td>

<td className="border border-gray-300 px-4 py-2">{new Date(message.createdAt.toDate()).toLocaleString()}</td>

</tr>

))}

</tbody>

</table>

<div className='my-8 w-full md:w-2/3 lg:w-3/4 xl:w-1/2 mx-auto min-h-1/2 table-background'>

<Line data={chartData} options={chartOptions} />

</div>

</div>

)}

</div>

);

}

export default Rezultate;

# Bibliografie

[1] Imagini modul senzor puls cariac MAX30100 <https://www.emag.ro/modul-senzor-de-frecventa-cardiaca-puls-oximetru-max30100-ai465-s281/pd/DPRL27MBM/?X-Search-Id=248df114786a700de094&X-Product-Id=64974538&X-Search-Page=1&X-Search-Position=0&X-Section=search&X-MB=0&X-Search-Action=view>

[2] Imagine pini ESP32 [pinout.png (1617×548) (etechnophiles.com)](https://www.etechnophiles.com/wp-content/uploads/2021/02/pinout.png)

[3] Imagine pentru diagrama functionala <https://www.analog.com/media/en/technical-documentation/data-sheets/max30100.pdf>

[4]ArduinoIDE <https://docs.arduino.cc/learn/starting-guide/the-arduino-software-ide>

[5]React.js <https://blog.hubspot.com/website/react-js>

[7]Firebase <https://fotc.com/ro/blog/firebase/?gclid=CjwKCAjwitShBhA6EiwAq3RqAzCYgVSzoyzV6tSsILg3FHZxF9VKuRePEoB5Vd2k7GbkChwiSDbJmRoCby8QAvD_BwE>

[8]https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/#:~:text=The%20ESP32%20can%20scan%20nearby,range%20of%20your%20ESP32%20board.