

UNIVERSITATEA DIN BUCUREŞTI
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENȚĂ

Aplicație full-stack JavaScript

respir.info

Serviciu de monitorizare a calității aerului în București

COORDONATOR ȘTIINȚIFIC
Lect. dr. Carmen Chirita

ABSORVENT
Vladimir-Cristian Ioana

BUCUREŞTI
2018

Cuprins

1 Introducere	6
1.1 Ce? (sau tema lucrării)	6
1.2 De ce? (sau justificare)	6
1.2.1 Inovație și aplicabilitate	6
1.2.2 Demonstrarea cunoștințelor	8
1.3 Cum? (sau structura aplicației - pe scurt)	9
1.3.1 Proiect pilot	10
1.3.2 Sistemul de măsurare	10
1.3.3 Aplicația web	11
2 Noțiuni teoretice	12
2.1 Calibrarea senzorilor. Măsurători	12
2.1.1 Introducere	12
Cum funcționează un senzor de gaz?	12
2.1.2 Calibrarea senzorilor	14
Calibrarea unui senzor MQ-135 pentru măsurarea nivelului de CO ₂	15
2.2 Citirea poziției geografice. Protocolul NMEA 0183	16
2.2.1 Cum funcționează un receptor GPS?	16
2.2.2 Protocolul NMEA 0183	16
2.2.3 Implementarea unei biblioteci pentru parsarea NMEA 0183	18
2.3 Full Stack JavaScript. MEAN	19
2.3.1 Ce înseamnă dezvoltare “Full Stack”?	19
Despre JavaScript	20
2.3.2 MEAN	20
Despre JavaScript și Node, sau de ce JavaScript pe server?	21
2.3.3 Considerente pro și contra	22
PRO:	22
Contra:	25
2.3.4 Recomandări de utilizare	25
2.4 Baze de date SQL vs noSQL. MongoDB	26
2.4.1 Mituri și adevăruri	26
Mit: NoSQL va înlocui SQL	26
Mit: NoSQL e mai bun/mai prost decât SQL	27

Mit: Există o distincție clară între SQL și NoSQL	27
Mit: Limbajul sau platforma determină alegerea bazei de date	27
2.4.2 SQL vs NoSQL: în practică	27
Tabele SQL vs documente NoSQL	27
Scheme și constrângeri	29
Normalizare	29
Relaționare via JOIN	30
Integritatea datelor	31
Tranzacții	31
Operații CRUD	31
Performanță	32
În practică	33
2.4.3 Comparație - pe scurt	33
2.4.4 Studiu de caz: Oracle DB vs Mongo DB	36
Când se recomandă Oracle DB?	39
Când se recomandă Mongo DB?	39
3 Proiectarea aplicației	40
3.1 Sistemul de măsurare	40
3.1.1 Descriere	40
Ce este Arduino?	40
3.1.2 Proiectarea dispozitivului	41
Selecția componentelor	41
Proiectarea circuitului	41
Asamblarea circuitului:	45
3.1.3 Programarea circuitului	46
Configurarea mediului de lucru	46
Programarea microprocesorului	47
Scrierea pe serială	47
Scrierea pe cardul SD	47
Citirea senzorilor	48
Citirea informațiilor GPS	49
Programul principal	50
3.1.4 Îmbunătățiri viitoare (sau TODO list)	50
3.2 Aplicația web	51
3.2.1 Alegerea tehnologiei. Motivații	51
Express.js	52
Mongo DB. Mongoose	52

3.2.2 Proiect pilot. Deployment	53
Mediul de lucru. Redeployment	54
3.2.2 Configurarea și indexarea bazei de date	55
Configurarea aplicației	55
Indexarea	56
3.2.3 Proiectarea aplicației	57
Structura	57
Dezvoltare	58
3.2.3.1 Gestiunea utilizatorilor	59
3.2.3.2 Încărcarea măsurătorilor	64
3.2.3.3 Afisare hartă	66
3.2.3.4 Header & footer. Stilizare	69
3.2.3.5 Articole	69
Testare	70
4 Exploatarea aplicației	71
Efectuarea măsurătorilor	71
Încărcarea măsurătorilor în aplicația web	71
5 Ghid de utilizare	73
5.1 Administrare	73
5.1.1 Operații de bază	73
Accesarea aplicației	73
Înregistrare	74
Autentificare	76
Ieșire	77
Contact	77
Informații generale	78
5.1.2 Utilizatori	78
Modificarea datelor personale	78
Configurarea preferințelor	79
Administrarea permisiunilor	80
5.2 Continut	81
Adăugarea articolelor	81
Modificarea unui articol	83
Adăugarea măsurătorilor	84
Afisarea hărții	87
Blog	89

6 Concluzii	90
6.1 Un punct de vedere asupra tehnologiilor utilizate	90
6.2 Concluzii la nivel practic	90
6.3 Dezvoltări ulterioare ale aplicației	91
Bibliografie	93
ANEXE	94
1 Dicționar de termeni și acronime	94
2 Agregarea recursivă a măsurătorilor în Node.js	97

1 Introducere

1.1 Ce? (sau tema lucrării)

Propun dezvoltarea unui sistem de monitorizare a calității aerului în zone cu risc ridicat de poluare (în principal în orașe). Aplicația urmărește informarea utilizatorilor cu privire la:

- calitatea aerului în zona de interes (în funcție de concentrațiile de gaze toxice urmărite)
- posibile cauze și consecințe ale condițiilor descrise (surse de poluare, evoluții pe timp scurt, mediu și lung, impact asupra sănătății)
- recomandări pentru a elimina cauzele poluării

Pe baza acestor informații utilizatorii pot lua decizii optime cu privire la:

- alegerea unor zone pentru agrement, recreere, activități sportive
- alegerea unei zone pentru locuit
- combaterea factorilor de poluare

Aplicația poate fi extinsă prin interconectarea cu aplicații de vânzări imobiliare, de programări activități sportive, și.a.

1.2 De ce? (sau justificare)

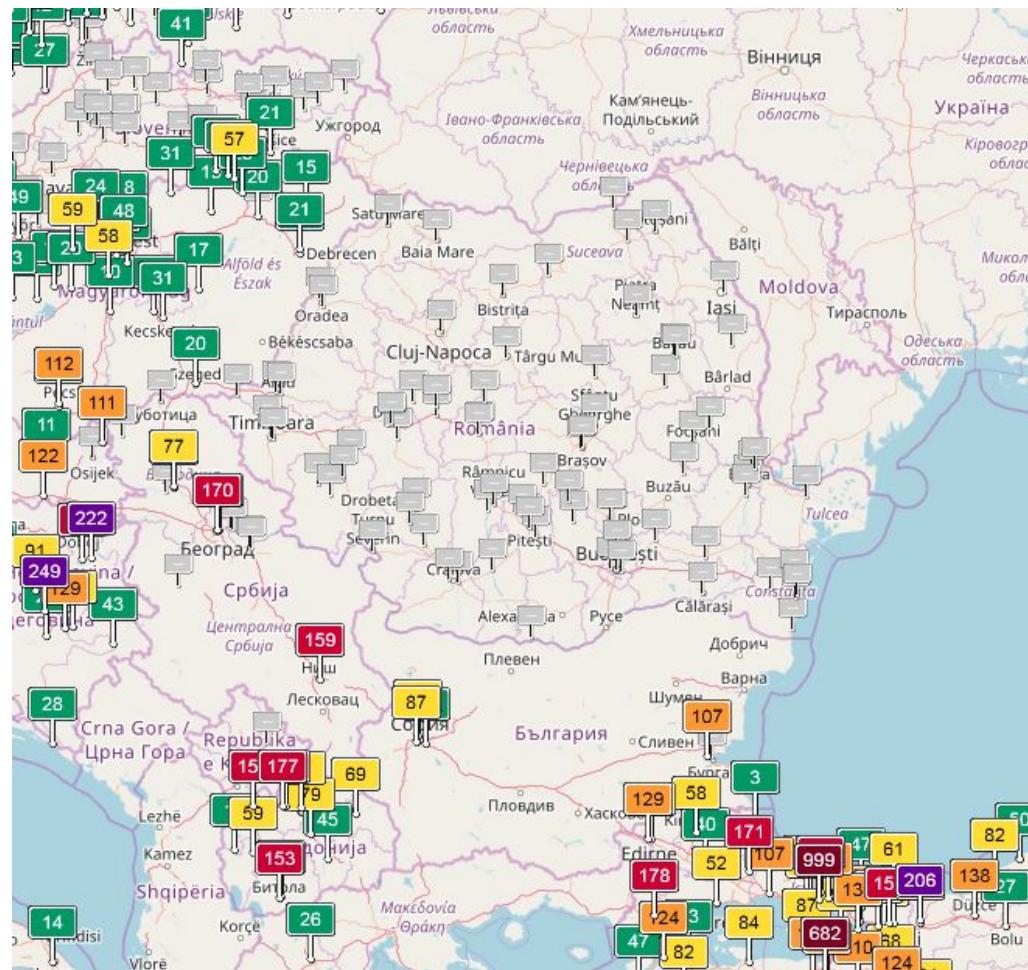
1.2.1 Inovație și aplicabilitate

Există multe discuții în spațiul public și la nivel politic despre evoluția poluării în România, pericolul ei în zone rezidențiale și măsuri împotriva poluării (fie ele taxe, restricții sau inițiative).

Cu toate acestea, există foarte puține date publice concrete despre situația actuală a poluării în marile orașe. După o căutare aprofundată, am identificat doar următoarele:

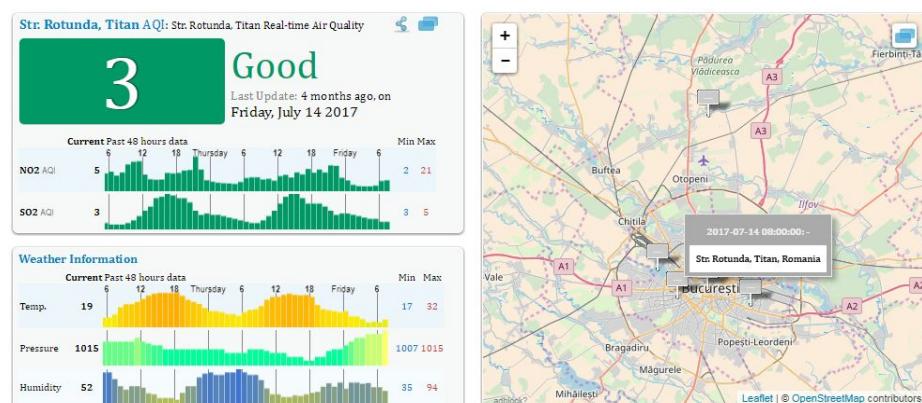
- site-ul AQI/ANPM cu date despre calitatea aerului (aqicn.org): doar 4 centre fixe de măsurare în București (142 la nivel național), toate inactive și unul nefuncțional (la data redactării lucrării - noiembrie 2017 - doar cele din Titan, calea Victoriei și lacul Morii aveau un istoric activ); există și site-ul mai actualizat

[calitateaer.ro](#), dar nu oferă informații relevante și se bazează pe aceleași sisteme de măsurare



Calea Victoriei, Cercul Militar Air Pollution: Real-time Air Quality Index (AQI)

CALEA VICTORIEI	STR. DRUMUL TABEREI, DRUMUL	STR. ROTUNDA, TITAN	STR. LACUL MORII, LACUL MORII	COMUNA BALOTESTI,	BUL BUCURESTI, PLOIESTI	LOCATE THE NEAREST CITY	SEARCH FOR YOUR CITY
-----------------	-----------------------------	---------------------	-------------------------------	-------------------	-------------------------	-------------------------	----------------------



Datele din Romania și situația globală a indicatorilor principali de poluare în zonă conform [aqicn.org](#) (noiembrie 2017)

- studii generale ale organizațiilor europene, locale sau independente (de pildă studiul de caz al Gabrielei Iorga privind monitorizarea poluării aerului¹)
- știri apărute în media pe baza investigațiilor independente - mai mult însă despre posibile cauze și efecte ale poluării (de pildă alarmarea în urma creșterii numărului de mașini în București) decât date concrete despre aceasta

De asemenea, suspect este că măsurătorile din București arată valori ale poluării sub cele din afara orașelor din Ungaria, în timp ce valorile măsurate în Belgrad, Sofia sau Istanbul depășesc cu mult limitele permise.

În aceste condiții se conturează nevoia unei aplicații care să ajute la identificarea problemelor, prevenirea cauzelor, educarea cetățenilor precum și optimizarea recomandărilor în vederea deciziilor acestora, pentru îmbunătățirea calității vieții tuturor.

Aplicația își propune măsurători periodice cu ajutorul unor dispozitive mobile, pe baza cărora să se construiască o hartă a nivelului de poluare și a calității aerului. Utilizatorii vor avea posibilitatea, prin intermediul unei platforme online, să:

- verifice calitatea aerului și factorii principali de poluare într-o zonă selectată pe hartă
- vizualizeze impactul condițiilor prezентate, pe termen scurt, mediu și lung asupra sănătății
- primească recomandări pentru diverse alegeri: zone rezidențiale, zone pentru activități sportive, etc
- să citească știri agregate din domeniu

1.2.2 Demonstrarea cunoștințelor

În general, o lucrare de licență urmărește să demonstreze că absolventul este un specialist care poate gestiona o aplicație software de la cap la coadă.

Avantajul acestei lucrări este că include o serie de provocări din arii variate de specialitate:

- Dispozitivul hardware de măsurare a calității aerului:
 - Studiul și proiectarea unui dispozitiv inteligent
 - Asamblarea unui circuit electronic
 - Programarea circuitului electronic
 - Calibrarea senzorilor și efectuarea de măsurători

¹ Air Pollution Monitoring: A case Study from Romania - Gabriela Iorga

- Prelucrarea și stocarea datelor:
 - Extragerea datelor
 - Prelucrarea și salvarea datelor
 - Algoritmi de gestiune a datelor, optimizare și statistică
 - Proiectarea unei baze de date
- Platforma online pentru informarea utilizatorilor:
 - Proiectarea unei aplicații web cu interfață grafică
 - Realizarea de documentație tehnică

Pe lângă asta, mi-am propus să experimentez tehnologii "la modă" în lumea programării - aplicațiile web full-stack JavaScript. Am explorat diverse soluții pentru construirea unei aplicații care să urmeze această direcție, iar în paginile de teorie - pe partea aplicației web - se vor vedea mai mult motivațiile alegerilor mele decât reluarea elementelor constitutive - obicei profesional dobândit în anii de realizare de documentație tehnică de arhitectură și design software.

1.3 Cum? (sau structura aplicației - pe scurt)

Aplicația are 2 componente principale:

1. Sistemul de măsurare: un *microcontroller* ce gestionează senzori de gaze, un modul GPS și un modul SD
2. Aplicația web: o aplicație în care sunt introduse datele măsurate de sistem și care servește utilizatorilor, prin intermediul unei pagini web, un conținut text/grafic cu informații despre zone de interes (selectabile pe hartă)

Pentru efectuarea de măsurători, administratorul site-ului (și, mai târziu, utilizatorii) introduce un card SD în dispozitivul de măsurare și îl pornește prin conectarea la o sursă de alimentare. Pe parcursul deplasării în zone de interes, dispozitivul înregistrează pe card date despre calitatea aerului și locația măsurătorii, într-un format standard (prestabilit).

Pentru încărcarea măsurătorilor în baza de date, un utilizator autorizat încarcă fișierele stocate pe card pe platforma online. Măsurătorile sunt validate și salvate în baza de date.

Orice utilizator poate solicita aplicației date despre o locație, selectând-o pe hartă. Aplicația va afișa măsurători relevante (conform cu un algoritm care să țină cont de loc,

oră, ş.a.) conform cu datele existente și recomandări bazate pe configurații predefinite și aplicații partenere.

1.3.1 Proiect pilot

În fapt, acest proiect își propune să fie un proiect pilot (*PoC*). Asta deoarece:

1. Pentru a măsura precis concentrația gazelor de interes se recomandă folosirea unor senzori de calitate, în consecință relativ scumpi
2. Pentru măsurarea tuturor gazelor de interes este necesară achiziționarea (și în special calibrarea mai multor senzori)
3. Pentru acoperirea optimă a suprafeței de interes (în acest caz întreg Bucureștiul) ar trebui investit timp în multe măsurători (eventual regulat de către administratorul site-ului, sau cu implicarea utilizatorilor în proces)
4. Calitatea aerului se schimbă permanent, în funcție de curentii de aer. În aceste condiții prezintă relevanță statisticile bazate pe un număr mare de date și predicțiile care se pot realiza pe baza acestor statistici

Valoarea măsurătorilor senzorilor MQ folosiți reflectă doar o tendință a evoluției concentrației gazelor la momentul măsurării, nereprezentând concentrația exactă de gaze.

Având în vedere bugetul necesar pentru implementarea completă a sistemului, proiectul de față își propune demonstrarea posibilităților și beneficiilor, urmând ca implementarea completă să fie subiectul unor dezvoltări ulterioare.

Se recomandă dezvoltări ulterioare ale aplicației pentru:

- îmbunătățirea datelor prin folosirea unor senzori mai performanți, mai preciși
- implicarea utilizatorilor pentru acoperirea unei arii mai largi și pentru efectuarea de măsurători periodice

1.3.2 Sistemul de măsurare

Sistemul de măsurare este construit pe platforma *Arduino Industrial*, pe care se montează urmatoarele elemente:

- senzori de măsurare a concentrației gazelor:
 - MQ-4: gaz metan (CH_4)
 - MQ-9: monoxid de carbon (CO), gaz lichefiat (LPG), gaz metan (CH_4)
 - MQ-135: calitatea aerului, detectând amoniac(NH_3), oxizii de azot(NO_x), alcool, benzen, fum, dioxid de carbon(CO_2), ş.a.

- antena GPS
- modul card SD
- baterie

Programarea sistemului de măsurare se realizează pe platforma Arduino, folosind limbajul C și biblioteci specifice.

Pentru efectuarea de măsurători, sistemul este activat prin alimentare și transportat în mașină (pe parcursul măsurătorii senzorii sunt expuși la exterior).

1.3.3 Aplicația web

Aplicația web este formată din:

- Baza de date, pe care sunt stocate măsurătorile, precum și datele utilizatorilor, configurări, informații generale, articole, și.a.
- Platforma web, care afișează diverse informații pe baza măsurătorilor, hărți, permite gestionarea preferințelor și configurațiilor utilizatorilor, încărcarea de noi măsurători, interconectarea cu diverse alte interfețe (de exemplu platforme de vânzări imobiliare)

Tehnologiile folosite:

- Baza de date: *mongoDB*
- Platforma web: aplicație *full-stack*² *Javascript*, folosind *framework*³-ul *Express*, precum și *Google Maps API*, și.a. (mai multe detalii în capitolul *Proiectarea Aplicației*)

² Aplicație *full-stack [JS]* - termen intraductibil ce descrie o aplicație care folosește aceeași tehnologie pe toate nivelele (*front-end* și *back-end*)

³ sau "platformă". Fiind un termen des întâlnit, am preferat folosirea termenului englezesc, pentru a evita confuzii

2 Noțiuni teoretice

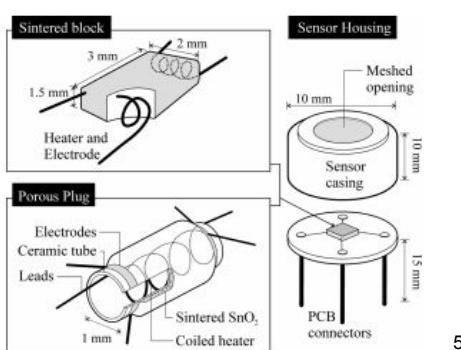
2.1 Calibrarea senzorilor. Măsurători

2.1.1 Introducere

Există mai multe metode de detecție a gazelor, printre care: cromatografia gazelor, spectroscopie în infraroșu cu transformare Fourier, detectori de chemiluminescență, spectroscopie de masă, dar cea mai avantajoasă este cea bazată pe semiconductori, deoarece senzorii sunt ieftini, ușor de miniaturizat, rezistenți în timp și în condiții grele de lucru - inclusiv temperaturi ridicate. De asemenea permit detectarea mai multor categorii de gaze în același timp și sensibilitatea poate ajunge la detecția variației unei concentrații de 1 la 1.000.000 (1 ppm⁴).

Cum funcționează un senzor de gaz?

Un senzor de gaz este compus dintr-un exoschelet metalic care găzduiește un semiconductor sensibil la gazele de interes (cel mai răspândit fiind dioxidul de staniu - SnO_2 - folosit în particular pentru depistarea monoxidului de carbon). Când semiconductorul este conectat la sursa de curent, acesta se încălzește, iar gazele din apropiere sunt ionizate și absorbite. Aceasta determină variația rezistenței semiconductorului, care la rândul ei determină variația tensiunii curentului care circulă prin circuit. Aceasta este valoarea pe care ne bazăm pentru a face măsuratoarea. Mai departe, consider explicarea mecanismului electric în afara scopului acestei lucrări.



5



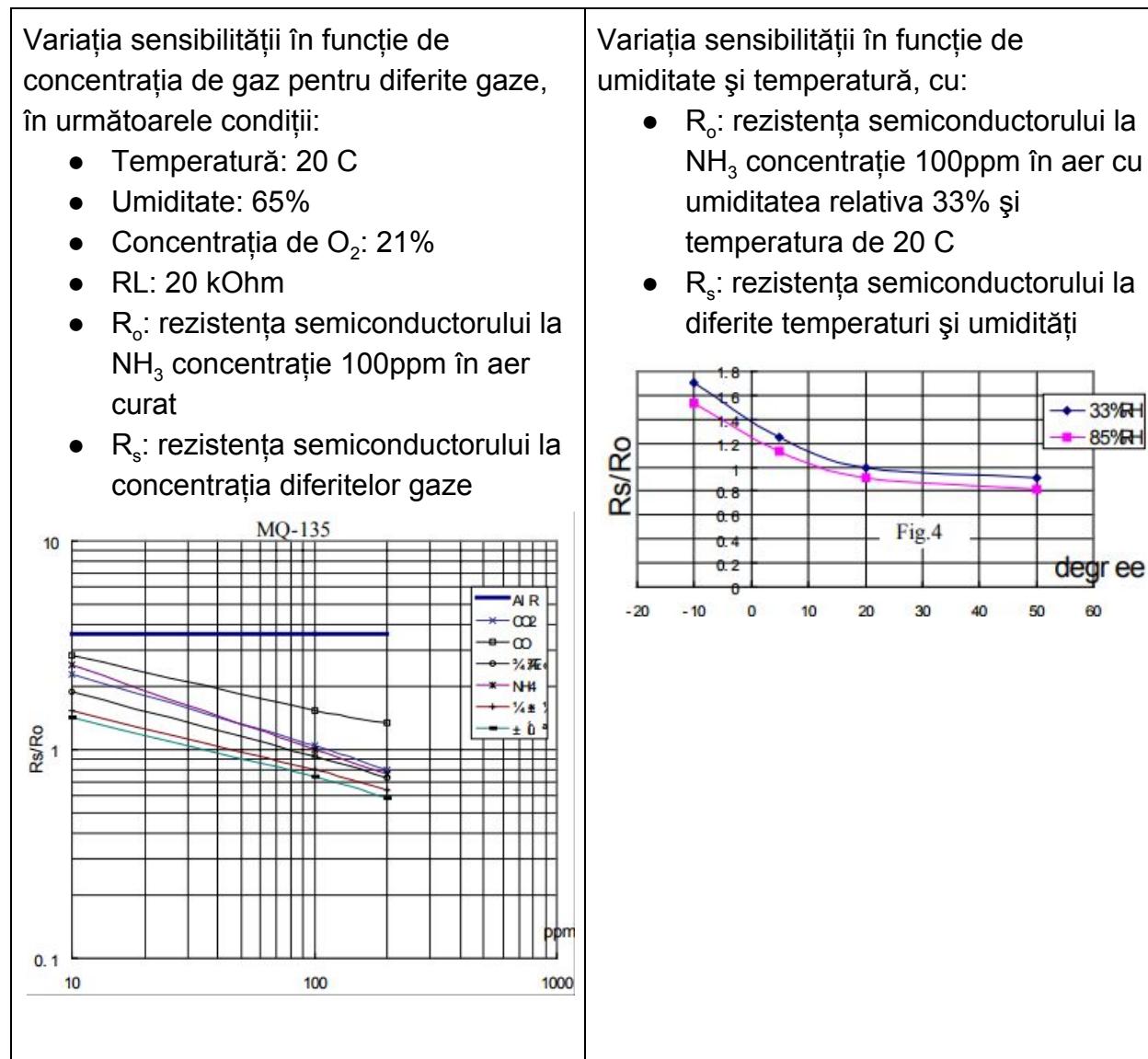
6

⁴ ppm - parts per million

⁵ Mecanismul electric. Preluat din lucrarea *Nanostructured Tin Dioxide Materials for Gas Sensor Applications* - T. A. Miller, S. D. Bakrania, C. Perez, M. S. Wooldridge, împreună cu fragmentul introductiv

⁶ Un senzor fizic

Sensibilitatea senzorului este determinată de raportul între rezistența semiconductorului în aer și rezistența acestuia în gazul de interes (R_s/R_o). Acesta însă depinde de structura, calitatea, dimensiunea semiconductorului, temperatura de operare și nu în ultimul rând prezența altor aditivi. Ecuatiile dependențelor sunt de obicei descrise în specificațiile senzorului (mai jos specificațiile pentru senzorul MQ-135 produs de Olimex).



- Când gazul de interes nu există în prezența senzorului, ieșirea digitală (DO) are valoarea 1, iar ieșirea analogică (AO) are valoarea 1023.
- Când există gaz de interes într-o concentrație oarecare în prezența senzorului, ieșirea digitală are valoarea 0 iar ieșirea analogică are o valoare mult mai mică

de 1023. Prin conectarea cu un potențiomетru se poate controla activarea ieșirii digitale la o anumită valoare a ieșirii analogice.

Pentru a controla sensibilitatea senzorului, se montează o rezistență între ieșirea GND (ground) și pin-ul GND al plăcii. Valoarea acesteia poate fi între 2 kOhm și 47 kOhm (cu cât mai mare, cu atât mai sensibil este senzorul).

2.1.2 Calibrarea senzorilor

Pentru acest tip de senzori - relativ ieftini - calitatea, compoziția și dimensiunile semiconductorului variază între seriile de produse. Se poate vedea din imaginea de mai jos (a interiorului unui senzor) de ce ne putem aștepta la diferențe mari: cel puțin dimensiunile semiconductorului diferă probabil de la senzor la senzor.



7

În consecință nu putem vorbi despre valori absolute ale rezistenței raportate la variații ale concentrației de gaz. Pentru măsurări precise, fiecare senzor trebuie calibrat în condiții cunoscute (ale rezultatelor așteptate):

- Dacă intenția este să se măsoare un singur tip de gaz dintre cele la care senzorul este sensibil, rezistența acestuia poate fi calibrată prin expunerea la o concentrație cunoscută de gaz
- Dacă intenția este să se măsoare orice gaz (precum la senzorul MQ-135 pentru calitatea aerului), se poate lua ca referință o valoare a rezistenței pentru aer curat. Valoarea de referință se poate lua doar după un timp de încălzire de circa 24 de ore

⁷ Secțiune printr-un senzor de gaz

Calibrarea unui senzor MQ-135 pentru măsurarea nivelului de CO₂^{8 9}

MQ135 este un senzor folosit pentru măsurarea calității aerului, fiind sensibil la amoniac(NH₃), oxizii de azot(NO_x), alcool, benzen, fum, dioxid de carbon(CO₂) și altele.

Din start, luăm ca marjă (și ignorăm) erorile generate de variațiile determinate de temperatură, umiditate și variația concentrației celorlalte gaze măsurate.

Conform graficului producătorului din figura de mai sus (al variației sensibilității în funcție de concentrația de gaze) dat pentru a corela datele de ieșire ale senzorului cu concentrația de gaz măsurată în ppm, lucrăm cu o funcție exponentială:

$$y = a * x^b \text{ sau } ppm = a * \left(\frac{Rs}{Ro}\right)^b$$

Folosind graficul, putem extrage suficiente valori x și y pentru a determina prin regresie exponentială factorul de scalare (a) și exponentul (b) pentru gazul pe care dorim să îl măsurăm, după care:

$$Ro = Rs * \sqrt[b]{\frac{a}{ppm}}$$

Pentru a identifica R_o vom avea nevoie să expunem senzorul la o concentrație cunoscută dintr-un gaz dat, apoi să citim valoarea R_s de ieșire. Știm cu aproximativă cantitatea de dioxid de carbon (CO₂) din atmosferă: ~403 ppm în condiții normale. Vom folosi această valoare pentru calibrare. Astfel, pentru CO₂, după ce extragem date pentru suficiente puncte pe grafic și aplicăm regresia exponentială obținem funcția:

$$ppm = 116.602 * \left(\frac{Rs}{Ro}\right)^{-2.769}$$

Pentru o calibrare cât mai precisă, este indicată încălzirea senzorului pentru 24 de ore înaintea măsurării R_s în aer liber. Am identificat valoarea R_s = 103.

Cu erorile de rigoare (mai mult generate de lipsa datelor precise privind concentrația de CO₂ decât de aproximativa coeficientilor funcției): R_o = 161.

În continuare se pot calcula valorile concentrației de CO₂ folosind direct măsurătorile, după formula:

$$ppm(CO_2) = 116.602 * \left(\frac{Rs}{161}\right)^{-2.769}$$

Trebuie totuși menționat că vom folosi valori relative la concentrația de CO₂ din locația calibrării senzorului.

⁸ În această lucrare ignor variația în funcție de temperatură și umiditate, deoarece marja de eroare este mică și efectuez măsurători cu un singur prototip în condiții relativ similare. Totuși, pentru măsurători mai precise și în condițiile efectuării mai multor măsurători cu dispozitive diferite, aceasta trebuie luată în considerare

⁹ O parte din raționament este inspirată de pe blogul davidegironi.blogspot.ro (*Cheap CO2 meter using the MQ135 sensor with AVR ATmega*)

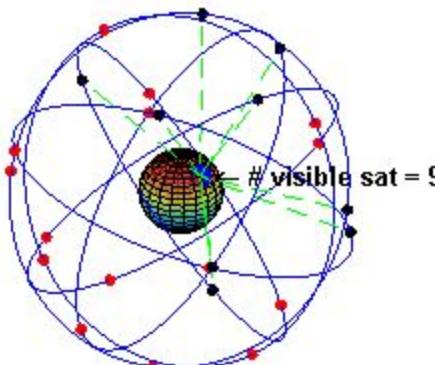
2.2 Citirea poziției geografice. Protocolul NMEA 0183

2.2.1 Cum funcționează un receptor GPS?

În fiecare dispozitiv capabil să afișeze poziția geografică (fie el smartphone, brățară fitness, etc) este încorporat un modul numit "receptor GPS". Aceștia se folosesc de o rețea de sateliți și stații terestre pentru a calcula poziția și timpul curent aproape oriunde în lume.

În orice moment există cel puțin 24 de sateliți care orbitează în jurul Pământului la o altitudine de peste 18.000 km, astfel poziționați încât cel mult 12 să aibă "vedere" la orice locație dată.

Rolul fiecărui satelit e să transmită permanent (în mod *broadcast*) date despre poziția lui și momentul transmisiei via unde radio (1.1 - 1.5 GHz).



10

Receptorul GPS primește date de la o serie de sateliți GPS - cei pe care îi are în raza directă, și poate calcula poziția lui geografică dacă "vede" cel puțin 4 sateliți, având următoarele date:

- momentul transmisiei/recepției datelor de la fiecare satelit
- poziția pe orbită a fiecărui satelit

Mai departe, este o simplă problema de fizică/geometrie calculul poziției receptorului GPS.

Evident, există mici erori de precizie - în general de până la 15 m.

2.2.2 Protocolul NMEA 0183

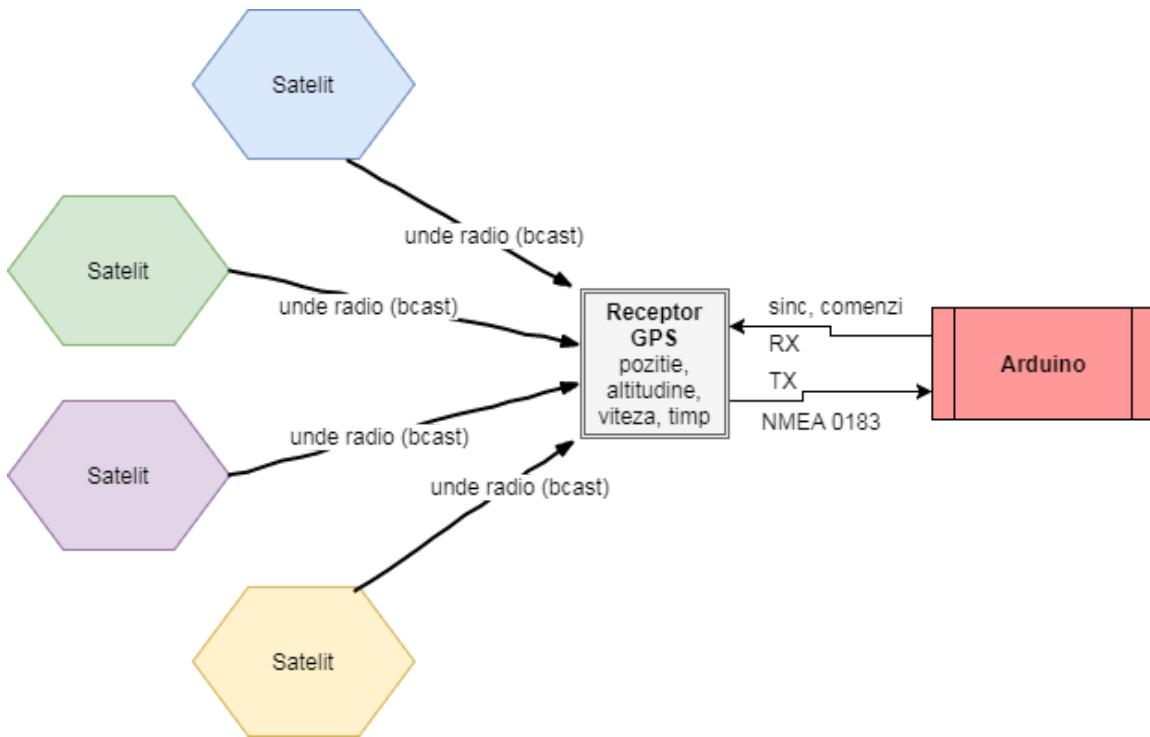
Pentru a transmite informațiile despre poziție și timp, mareala majoritatea a receptorilor GPS folosesc protocolul NMEA.

NMEA¹¹ a definit cu peste 30 de ani în urmă un standard de comunicație între diverse piese de echipament electronic folosite pentru navigație. O evoluție a acestui standard,

¹⁰ GPS Basics - learn.sparkfun.com/tutorials/gps-basics

¹¹ NMEA - National Marine Electronics Association

cunoscută drept *NMEA 0183*, a devenit metoda cea mai folosită pentru comunicarea între dispozitive în navigație. Astfel, aproape toate dispozitivele GPS implementează acest standard pentru transmisia de date.



Standardul NMEA definește protocolul prin care datele sunt transmise electronic unui alt dispozitiv pe o interfață serială, cu o viteza de transmisie configurabilă¹².

Protocolul NMEA 0183 definește o metodă de transmisie a datelor bazată pe “propoziții”. O propoziție este practic un sir de caractere ASCII (până la 80 de caractere) cu un anumit format, astfel:

- \$ttsss,d1,d2,...<CR><LF>**
- **\$** marchează începutul unei noi propoziții
- **tt** reprezintă tipul transmițătorului (de pildă *RA*=radar, *SD*=sonar, *GP*=receptor GPS)
- **sss** reprezintă tipul mesajului (de pildă *RMC*=date minime recomandate pentru GPS, *GGA*=actualizarea locației GPS))
- **d1,d2,...** - următoarele siruri de caractere separate prin virgulă reprezintă câmpurile transmisiei, funcționalitatea lor variind în funcție de tipul de mesaj definit la începutul propoziției
- **<CR><LF>**¹³ marchează sfârșitul propoziției

¹² Implicit 4800 baud; în implementarea acestui proiect am folosit 9600

¹³ <CR><LF> - *Carriage Return, Line Feed*; 2 caractere ce marchează în general sfârșitul unei linii

2.2.3 Implementarea unei biblioteci pentru parsarea NMEA 0183

Pentru citirea poziției prezintă interes 2 tipuri de propoziții: *GPGGA* și *GPRMS*.

Mai jos, specificațiile NMEA pentru sensul fiecărui câmp de tipul *GPGGA*:

1	2	3 4	5 6 7	8	9	10 11	12 13	14	15

\$--GGA, hhmmss.ss, llll.ll, a, yyyy.y, a, x, xx, x.x, x.x, M, x.x, M, x.x, xxxx*hh
1) Timp (UTC)
2) Latitudine
3) N/S (Nord sau Sud)
4) Longitudine
5) E/W (Est sau Vest)
6) Indicator semnal:
 0 - indisponibil,
 1 - GPS,
 2 - GPS diferențial
7) Număr de sateliți accesibili (00 - 12)
8) Variația erorii
9, 10, 11, 12) Altitudinea receptorului față de nivelul marii
13) Vârsta în secunde de la ultima actualizare DGPS¹⁴, altfel null
14) Id-ul stației DGPS
15) Suma de verificare (obișnuită prin XOR între toate caracterele propoziției)

Implementarea se bazează pe construcția unui obiect GPS.

Se "ascultă" fiecare caracter primit pe serială, astfel:

- citirea '\$' marchează începutul unei noi propoziții; se abandonează orice citire anterioară
- citirea ',' marchează trecerea la câmpul următor
- citirea '\r', '\n' sau '*' marchează validarea propoziției prin suma de verificare și interpretarea acesteia
- citirea oricărui alt caracter determină adăugarea caracterului la propoziția curentă și actualizarea câmpului de verificare

După interpretarea unei propoziții valide, câmpurile obiectului GPS (longitudine, latitudine, altitudine, timp, etc) sunt actualizate cu noile valori.

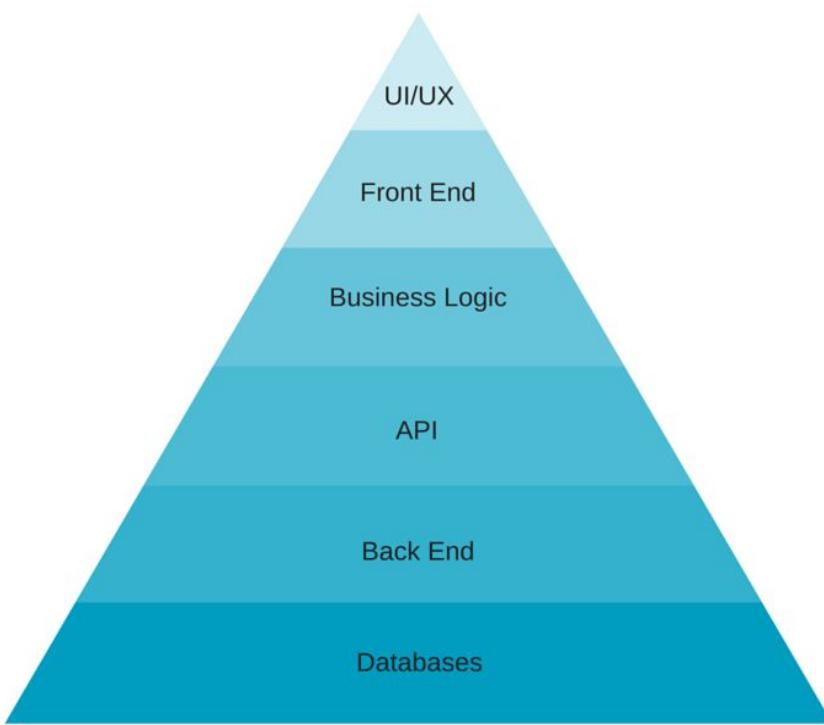
¹⁴ DGPS = GPS diferențial, folosește conectarea adițională cu stații terestre pentru micșorarea marjei de eroare de la circa 15m până la 10cm

2.3 Full Stack JavaScript. MEAN

2.3.1 Ce înseamnă dezvoltare “Full Stack”?

E aproape imposibilă dezvoltarea unui întreg produs software folosind o singură tehnologie. O asemenea sarcină implică utilizarea unei combinații de unelte și limbi de programare. O combinație standard de unelte și limbi este denumită în general “stivă” (sau *stack*). Dar o aplicație web constă din (cel puțin) o parte de client și o parte de server. Fiecare cu stiva lui.

Combinăția de tehnologii pentru dezvoltarea unei aplicații “de la cap la coadă” este denumită “*full stack*”. Abordarea e denumită “dezvoltare *full stack*”. Un expert generalist care are cunoștințe consistente în ce privește fiecare aspect al ingineriei software a produsului e de obicei denumit “dezvoltator *full stack*”.



Stiva cu modulele necesare pentru dezvoltarea unei aplicații web, în general

Experții care stăpânesc dezvoltarea la fiecare nivel, cu o duzină de tehnologii diferite, sunt extrem de rari. În consecință, datorită răspândirii ei și faptului că e *open-source*, este din ce în ce mai populară folosirea tehnologiei JavaScript pentru fiecare dintre aceste nivale.

Despre JavaScript

Limbajul JavaScript este un limbaj multilateral, arhitectura sa combinând stilul orientat pe obiecte, cel imperativ și cel funcțional. Astfel, deși e orientat pe obiecte, nu folosește clase ci se bazează pe prototipuri de obiecte pentru moștenire (metodele și câmpurile putând astfel fi schimbate dinamic pentru construcția unor noi obiecte). Funcțiile sunt și ele obiecte și pot fi trimise ca argumente la alte funcții precum în programarea funcțională. Funcțiile pot avea un număr variabil de argumente precum limbajele de scripting și suportă expresii regulate, similar cu Perl. Proprii JavaScript sunt obiectele asociative JSON, care prezintă avantajul unei structurări versatile a datelor la pachet cu mici costuri extra (comparând de pildă cu XML).

2.3.2 MEAN¹⁵

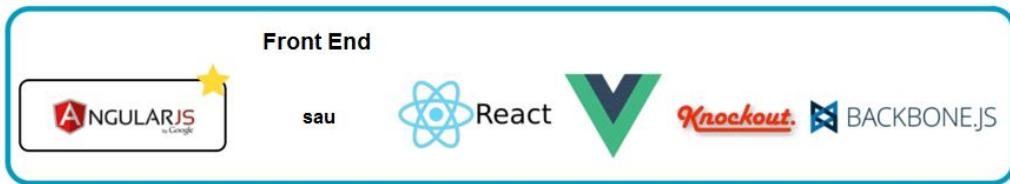
Stiva JavaScript se bazează pe o serie de *framework-uri*. Cea mai folosită este colecția MEAN, unde:

- MongoDB = baza de date
 - ◆ bază de date noSQL bazată pe fișiere ce folosește JavaScript ca și limbaj de interogare (mai multe în capitolul următor)
- Express.js = framework back-end
 - ◆ folosește conceptul de *middleware*, care împarte procesul între primirea request-ului și transmiterea răspunsului în componente (foarte similar *pipe*-ului din UNIX); astfel, fiecare componentă poate procesa/modifica doar ceea ce e nevoie, apoi delegă următoarei - o abordare foarte utilă pentru aplicații complexe; rezultă 2 tipuri de middleware:
 - intermediar, procesează request-ul și răspunsul și le transmite următorului middleware (care poate fi chiar clientul care a inițiat request-ul, dacă nu mai există alt middleware)
 - final, transmite răspunsul clientului
- Angular.js = framework front-end
- Node.js = mediu de rulare¹⁶, preluat din Chrome

Astfel, există și multe alternative:

¹⁵ Date preluate de pe blogul <https://www.altexsoft.com/blog>

¹⁶ Stiut ca "runtime environment"



Despre JavaScript și Node, sau de ce JavaScript pe server?

Node.js nu e prima încercare de a implementa partea de server în JavaScript, dar e prima încununată de succes, ba mai mult, un larg succes. Mai mulți factori au contribuit la el:

- motorul¹⁷ Google V8 pentru execuție cod JavaScript. Implementat în C++, obține performanțe foarte bune deoarece:
 - compilează codul JavaScript în cod nativ în loc să îl interpreteze în timp real
 - folosește un model de execuție asincron bazat pe evenimente și cozi de mesaje, ce permite stivuirea operațiilor astfel încât să execute un *callback* atunci când fiecare operație e gata
- limbajul JavaScript:
 - construit pe un model orientat pe evenimente, asincron, non-blocant, ce permite performanțe bune la execuție
 - limbaj interpretat, deci independent de platformă

¹⁷ cunoscut ca *engine*. *V8 engine*. Folosit mai întâi în browser-ul Google Chrome. *Open-source*, rulează pe Windows (7 și ulterioare), MacOS și Linux. Poate rula independent sau integrat în orice aplicație C++

- obiectele construite pe modelul JSON, native. Tipul JSON este deja standard pentru transmiterea de date pe web

Avantajele Node.js:

- interfață REST nativă; poate comunica cu toate celelalte componente via HTTP pentru operații CRUD bazate pe paradigma REST
- rulează pe un singur fir de execuție, lucrând cu evenimente non-blocante (alte sisteme, de pilda PHP, alocă un nou fir de execuție pentru fiecare request, necesitând altfel largi cantități de memorie). Folosește totuși un număr limitat de thread-uri la nivel kernel, pentru a garanta execuția asincronă a operațiilor fără blocarea cozii de evenimente (kernel-ul nu suportă ca toate operațiile să fie asincrone)



- NPM: managerul de pachete node, permite instalarea și gestionarea extrem de ușoară a modulelor pentru aplicație
- susținut de o comunitate largă, dedicată pentru îmbunătățirea lui constantă și la dezvoltarea a noi module pentru el

Dezavantajele Node.js:

- un singur thread înseamnă și un singur procesor, deci aplicația este mai dificil de scalat multiprocesor
- multe module înseamnă și că unele nu sunt testate suficient în producție; astfel, trebuie folosite precaut pentru aplicații folosite pe scară largă

2.3.3 Considerente pro si contra

Faptul că multe companii prestigioase, precum Groupon, Airbnb, Netflix, Medium și PayPal, dar și mici startup-uri au adoptat conceptul *full-stack JavaScript* pentru a-și

construi produsele software prezintă garantă în ceea ce privește avantajele și viitorul acestei tehnologii. Dar să vedem mai de aproape beneficiile, precum și problemele ei.

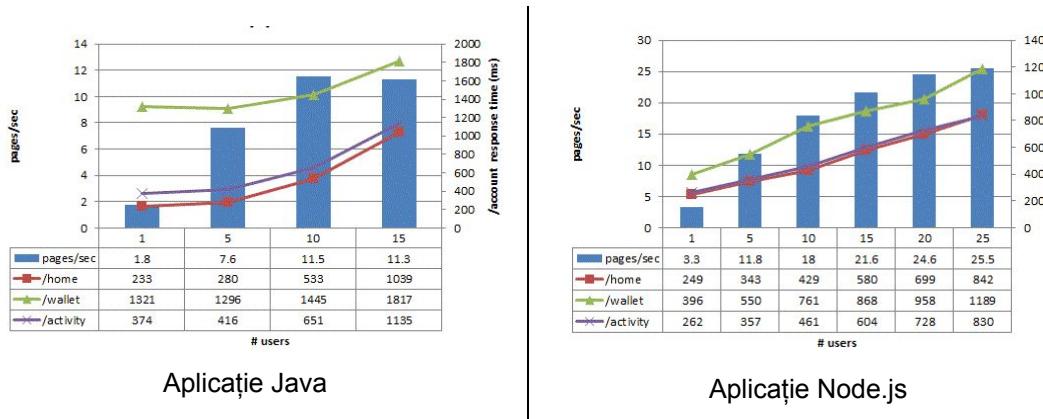
PRO:

- ★ Același limbaj înseamnă eficiență sporită cu mai puține resurse
 - Când toate părțile aplicației sunt scrise în JavaScript, membrii echipei își pot înțelege reciproc codul mai ușor. De obicei e o prăpastie între echipele front și back-end, se generează blocaje când unii așteaptă după alții, pe când astfel se pot combina dezvoltatorii într-o singură echipă multifuncțională. Comunicare mai bună înseamnă eficiență, care se traduc prin performanță și cost mai bun.
- ★ Refolosirea codului
 - Aceeași funcționalitate poate fi refolosită în front-end și back-end când logica e similară, pe principiul DRY¹⁸
- ★ Performanțe bune:
 - Node.js folosește un model orientat pe evenimente non-blocant, ceea ce îl face mult mai rapid comparat cu alte tehnologii de back-end, de pildă la rularea unui apel către baza de date:
 - în cazul clasic, aplicația așteaptă până primește răspuns, apoi trece mai departe:
 - `var resultset = db.query("SELECT * FROM 'table'");
drawTable(resultset);`
 - în schimb, Node.js definește o funcție ce se va executa la primirea răspunsului și trece mai departe după comandarea execuției fără să aștepte, fără tempi morți:
 - `db.query("SELECT * FROM 'table'", function(resultset){
drawTable(resultset);
});
doSomeThingElse();`
 - Mai jos, elemente dintr-un raport al PayPal¹⁹, comparând performanța Node.js după o migrare de la Java la Node.js, raport ce a determinat migrarea întregului pachet de software către Node:
 - dezvoltare de două ori mai rapidă, și cu mai puțini oameni (5 în Java, 2 cu Node)
 - cu 33% mai puține linii de cod
 - cu 40% mai puține fișiere

¹⁸ *Don't Repeat Yourself*, sau "nu te repeta". Principiu de programare ce recomandă abstractizarea în locurile unde implementarea ar fi repetitivă (opus lui WET - *Write Everything Twice*)

¹⁹ *Node.js at PayPal* - Jeff Harrell, 22 noiembrie 2013

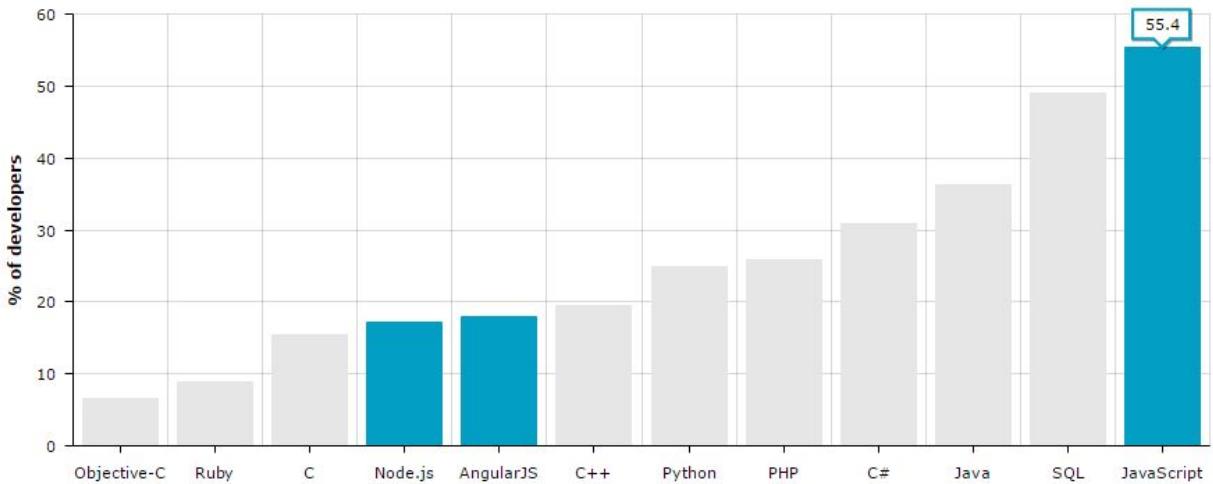
- de cel puțin două ori mai multe *request*-uri pe secundă, chiar și cu hardware mai puțin performant (1 vs 5 *core*-uri)



- timp de răspuns cu 35% mai scurt pentru aceeași pagină

★ Plaja de recrutare largă:

- conform cu sondajul anual *stackoverflow*²⁰, în 2016 JavaScript a fost de departe cel mai popular limbaj, folosit de 55.4% dintre cei ce au răspuns



- putem deci presupune că dintre cei aproximativ 19 milioane de programatori din întreaga lume²¹, circa 10 milioane folosesc JavaScript

★ Comunitate mare, în creștere, și cu o largă bază de cunoștințe

- Fiind susținut de giganți precum Facebook și Google, JavaScript are o comunitate ce crește rapid. Conform cu statisticile *StackOverflow* menționate mai sus, limbajul este în vârful topului cu cele mai populare limbiage. De asemenea, site-ul indică existența a peste 1.1 milioane de

²⁰ <https://insights.stackoverflow.com/survey/2016#technology>

²¹ conform cu studiul EDC: *Global Developer Population and Demographic Study 2017 Vol. 1*

întrebări cu tag-ul “JavaScript”, ceea ce demonstrează o puternică activitate a comunității de dezvoltatori și o cantitate vastă de informații și soluții

★ Totul este gratuit, *open-source*

- Majoritatea uneltelelor de dezvoltare JavaScript sunt gratuite și open-source. Asta le garantează și o evoluție rapidă datorată comunității active. De asemenea, ne putem baza pe cele peste 250.000 de pachete gestionate de npm.

Contra:

★ Performanța lasă de dorit când sunt implicate multe calcule și procesare de date

- rulând pe un singur procesor și secvențial, aplicația poate fi ușor blocată de un singur *task* care necesită multă putere de calcul; există soluții (de pildă împărțirea în *subtask*-uri mai digerabile sau crearea de procese copil), totuși există tehnologii mult mai performante pentru acest tip de sarcini (*machine learning*, procese ce implică algoritmi sau calcule matematice complexe)

★ Tehnologia este relativ Tânără

- lipsa experienței dezvoltatorilor (s-au lovit de mai puține probleme)
- integrare limitată cu anumite sisteme (de pildă baze de date relaționale: MySQL, Microsoft SQL Server, PostgreSQL)

★ Bun la toate, expert în nimic?

- Se crede că un dezvoltator poate stăpâni cu adevarat o singură arie și, cu fiecare abilitate dobândită, calitatea expertizei lui scade. În general sintaxa unui limbaj nu e principala dificultate în învățare, iar pentru a stăpâni dezvoltarea *full-stack*, pe lângă front-end dezvoltatorul trebuie să stăpânească programarea back-end, ce presupune cunoștințe în ceea ce privește protocolul HTTP, apele asincrone, elemente de stocarea datelor, etc. Există păreri că niciun dezvoltator nu e cu adevărat full-stack, ci este orientat ori spre front-end, ori spre back-end

2.3.4 Recomandări de utilizare

Deși sunt câteva puncte negative ale abordării full-stack JavaScript, e limpede că este folosită masiv și evoluează rapid. Numărul de proiecte de pe *GitHub* (peste 2 milioane) o demonstrează.

Astfel, se recomandă folosirea acestei abordări în special pentru:

- proiecte MVP
- aplicații ce implică interacțiuni în timp real (platforme de mesagerie, vânzări de acțiuni, platforme sociale, jocuri online, instrumente de colaborare, partajare, etc)
- aplicații IoT
- soluții e-Commerce
- aplicații de administrare și monitorizare

2.4 Baze de date SQL vs noSQL. MongoDB

În ceea ce privește implementările de stocare a datelor, sunt cunoscute două mari categorii: relaționale și non-relaționale. Și, precum în toate direcțiile de dezvoltare ale omenirii, în funcție de nevoi fiecare direcție s-a adaptat, a cunoscut creșteri și descreșteri.

În ultimele decenii, bazele de date relaționale au fost modelul dominant pentru gestiunea bazelor de date. În momentul de față, însă, bazele non-relaționale - NoSQL, sau *cloud* - câștigă teren ca soluție de business, în condițiile în care experții conștientizează că bazele relaționale nu sunt soluția perfectă pentru orice.

În acest capitol îmi propun explicarea fiecărei direcții și sublinierea avantajelor și dezavantajelor implementării în soluții software, într-un mod cât mai nepărtinitoar (deși e omenește a înclina către soluția aleasă).

2.4.1 Mituri și adevăruri

Cred că este utilă, pentru o abordare obiectivă, demontarea inițială a unor mituri (răspândite din dorința omenească orgolioasă de tipul “eu am dreptate și soluția mea e cea mai bună”)

Mit: NoSQL va înlocui SQL

E ca și cum s-ar spune că mașinile ar înlocui bărcile, fiindcă sunt tehnologie mai nouă. Deși SQL și NoSQL fac același lucru - stochează date - ele folosesc abordări diferite, care pot genera avantaje sau probleme în proiect. Deși a dobândit un proaspăt avânt și e “la modă”, NoSQL nu e un înlocuitor pentru SQL, ci o alternativă.

Mit: NoSQL e mai bun/mai prost decât SQL

Unor proiecte li se potrivește mai bine o bază de date SQL, altora NoSQL, altele pot folosi oricare dintre variante cu performanțe similare. Totul depinde de nevoi, așteptări și resurse.

Mit: Există o distincție clară între SQL și NoSQL

Nu este mereu adevărat, în practică. Unele baze de date SQL adoptă practici NoSQL și vice-versa. Actualele implementări tind să devină un hibrid, iar aceste hibride pot deveni opțiuni interesante pentru viitor.

Mit: Limbajul sau platforma determină alegera bazei de date

Suntem obișnuiți să ne raportăm la stive de tehnologii²², precum:

- LAMP: Linux, Apache, MySQL, PHP
- MEAN: MongoDB, Express, Angular, Node.js
- .NET, IIS și SQL Server
- Java Apache și Oracle

Într-adevăr, există motive practice, istorice și comerciale pentru care aceste stive s-au dezvoltat în acest format, dar asta nu înseamnă că sunt reguli. Oricine poate folosi o bază de date MongoDB într-un proiect PHP sau .NET, precum și MySQL sau SQL Server cu Node.js. Într-adevăr, nu va găsi la fel de multe exemple și resurse. Important este însă ca cerințele aplicației să fie cele care determină ce baza de date e folosită, și nu limbajul de programare.

2.4.2 SQL vs NoSQL: în practică

Tabele SQL vs documente NoSQL

Bazele de date SQL implementează o colecție de tabele de date în relație unele cu altele. De pildă, pentru o aplicație tip bibliotecă, informațiile despre fiecare carte vor fi într-o tabelă *carti*, astfel:

serie	titlu	autor	format	preț
978088246	MongoDB: de la incepator la avansat	Adrian Niculescu	ebook	39.90

²² <https://stackshare.io/stacks>: liste cu stivele de tehnologii folosite de companii de top din lume

978088418	Oracle si achizitia MySQL	Dan Pop	ebook	17.99
-----------	---------------------------	---------	-------	-------

Fiecare linie va reține informații despre câte o carte. Structura este rigidă, și nu vom putea folosi aceeași tabelă pentru a stoca alte informații sau pentru a reține un *string* unde câmpul este definit ca număr.

O bază de date NoSQL va stoca însă o înregistrare ca obiect într-o colecție *carte*, în format JSON (sau similar):

```
{
  serie: 978088246,
  titlu: "MongoDB: de la incepator la avansat",
  autor: "Adrian Niculescu",
  format: "ebook",
  pret: 39.90
}
```

În aceeași colecție se va putea stoca, fără nicio problemă, și o înregistrare relativ diferită, de pildă:

```
{
  serie: 978088418,
  titlu: "Oracle si achizitia MySQL",
  autor: "Dan Pop",
  format: "ebook",
  pret: "free",
  rating: "5/5",
  review: [
    { nume: "G Ionescu", text: "O analiza de valoare a celei mai
controversate achizitii." },
    { nume: "G Popescu", text: "Recomandata oricui vrea sa inteleaga
piata software." }
  ]
}
```

Tabelele SQL crează un tipar strict, astfel încât să elimine erorile la modificare. NoSQL e mai flexibil, dar poate genera probleme de consistență a datelor.

Scheme și constrângeri

În bazele SQL nu se pot adăuga date până când nu e definită o schema și constrângeri (optionale) precum:

- chei primare: identificatori unici ai înregistrării (precum seria în exemplul de mai sus)
- indecsi: ordonarea câmpurilor după care se fac căutări pentru îmbunătățirea vitezei
- relații: legături logice între câmpuri
- funcționalitate: de exemplu triggeri sau proceduri stocate

Schema trebuie proiectată și implementată înainte să se poată efectua operații de orice fel la nivel de aplicație. Se pot face schimbări și mai târziu, dar cu cât sunt mai mari cu atât mai complicate.

În bazele NoSQL se pot adăuga date oricând, oricum. Nu e nevoie de o proiectare inițială și nici măcar de crearea unei colecții înainte de inserarea primului element. De pildă în MongoDB prima apelare a procedurii va crea un document într-o nouă colecție `carte` - dacă nu a fost creată anterior:

```
db.carti.insert(  
    serie: 978088286,  
    titlu: "SQL vs NoSQL: razboi",  
    autor: "Tiberiu Valcea",  
    format: "ebook",  
    pret: 57.90  
)
```

Astfel, NoSQL este mai fezabilă pentru aplicații la care structura datelor este dificil de determinat la începutul proiectului (este totuși indicată o proiectare bună a structurii bazei de date - cât mai devreme - altfel costurile pentru dezvoltare vor fi din ce în ce mai mari).

Normalizare

Să zicem că vrem să adăugăm informații despre editură în exemplul de mai sus.

În SQL vom crea o nouă tabelă, `editura`, și vom asocia id-urile elementelor în tabela `carte`. Astfel datele nu sunt redundante, iar actualizarea unui element din `editura` nu implică impact asupra elementului corespunzător din `carte`.

În NoSQL putem într-adevăr folosi o referință la un element dintr-o altă colecție, dar putem opta și pentru integrarea unui obiect direct în colecția inițială.

```
{
    serie: 978088246,
    titlu: "MongoDB: de la inceput la avansat",
    autor: "Adrian Niculescu",
    format: "ebook",
    pret: 39.90,
    editura_id: "ED001"
}
```

- mai sus se referențiază un document din colecția edituri:

```
{
    id: "ED001"
    nume: "MatrixRom",
    tara: "Romania",
    email: "contact@matrixrom.ro"
}
```

În cazul de mai sus actualizarea informațiilor va fi mai rapidă, în cel de mai jos căutarea va fi mai rapidă (după cum se va vedea și în subcapitolul următor).

```
{
    serie: 978088246,
    titlu: "MongoDB: de la inceput la avansat",
    autor: "Adrian Niculescu",
    format: "ebook",
    pret: 39.90,
    editura: {
        nume: "MatrixRom",
        tara: "Romania",
        email: "contact@matrixrom.ro"
    }
}
```

Relaționare via JOIN

Bazele SQL oferă un mecanism simplu de relaționare: operația JOIN. Se pot obține date din mai multe tabele printr-o singură instrucțiune SQL. De exemplu:

```
SELECT carte.titlu, carte.autor, editura.nume
FROM carte
LEFT JOIN carte.editura_id ON carte.id;
```

va returna toate cărțile, autorii și numele de edituri asociate.

NoSQL nu are un echivalent al JOIN (mai puțin aplicațiile hibride, evident), ceea ce pentru un programator obișnuit cu SQL poate fi un soc. Dacă am folosi colecții normalizate ca în primul exemplu din subcapitolul anterior, ar trebui să extragem toate cărțile și toate editurile, apoi să facem asocierea la nivelul aplicației.

Integritatea datelor

SQL implementează reguli care pot constrânge utilizatorul în ce privește query-urile pe care le execută. Pentru exemplul dat îl poate forța, de pildă:

- la inserarea unei cărți, să se asigure că id-ul editurii este valid și corespunde unei intrări din tabela *editura*
- să nu permită ștergerea unei edituri cât timp are cărți asociate

Nerespectarea constrângerii va arunca o eroare, iar tranzacția nu va fi executată.

În cazul NoSQL nu există asemenea constrângerii. Fiecare document din colecție este privit ca independent.

Tranzacții

În SQL se pot solicita 2 sau mai multe schimbări ale datelor în cadrul aceleiași tranzacții atomice, pe principiul "totul sau nimic". Astfel, în caz că cel puțin o instrucțiune eșuează, un mecanism de *rollback* reduce toate datele în starea inițială.

În NoSQL este atomică modificarea unui singur document, dar nu există echivalent al unei tranzacții pentru documente multiple.

Operații CRUD

Crearea, citirea, actualizarea și ștergerea sunt operații de baza în orice bază de date.

- SQL este un limbaj declarativ, simplu și puternic, care a devenit un standard internațional (chiar dacă există mici diferențe între implementări)
- bazele NoSQL folosesc interogări stil JavaScript, cu argumente stil JSON.
Operațiile de bază sunt simple, dar obiecte JSON imbricate pot genera instrucțiuni foarte întortocheate pentru interogări mai complexe

Câteva exemple:

SQL	NoSQL
actualizarea unei cărți	
<pre>UPDATE carte SET pret = 19.99 WHERE serie = '978088246'</pre>	<pre>db.carte.update({ serie: '978088246' }, { \$set: { pret: 19.99 } });</pre>
extragerea tuturor cărților cu prețul > 10	
<pre>SELECT titlu FROM carte WHERE pret > 10;</pre>	<pre>db.carte.find({ pret: { >: 10 } }, { _id: 0, titlu: 1 });</pre> <p>Al doilea obiect JSON este o proiecție. Campul <code>_id</code> e returnat implicit, așa că trebuie să fie exclus.</p>
extragerea numărului de formate de carte	
<pre>SELECT format, COUNT(1) AS `total` FROM carte GROUP BY format;</pre>	<pre>db.carte.aggregate([{ \$group: { _id: "\$format", total: { \$sum: 1 } } }]);</pre> <p>Agregarea implică o construcție a unui nou set de documente din setul original</p>

Performanță

Este probabil cel mai controversat aspect.

NoSQL este lăudat în general că e mai rapid decât SQL. Deloc surprinzător: într-o colecție indexată, varianta nenormalizată a unui document permite extragerea informațiilor dintr-o colecție într-un singur request, fără operații JOIN complexe (deci o singură iterare de căutare).

Totuși, cea mai importantă este proiectarea bazei de date. Totdeauna o bază de date SQL bine gândită va performa net superior uneia NoSQL prost proiectată și vice-versa.

În practică

Ceea ce nu trebuie uitat, un aspect pe care l-am menționat în mod repetat de-a lungul lucrării, este vechimea și stabilitatea proiectelor: bazele SQL s-au “învechit” pe piață și, fiind folosite pe scară largă, multe probleme au fost găsite și fixate, așa că ne putem aștepta să fie mai stabile. De asemenea, dezvoltatorii și administratorii vor avea mai multă experiență în exploatarea lor.

Acestea fiind zise și luând în considerare cele de mai sus, în practică vom dori mai degrabă un sistem:

→ SQL, când:

- ◆ relațiile dintre date pot fi identificate de la început
- ◆ integritatea datelor este esențială
- ◆ tehnologia trebuie să fie stabilă, și tehnicienii experimentați (deci nu e loc de experimente); de asemenea, unde poate fi necesar suport tehnic urgent, plătit - asta înseamnă, în mare, aplicații software ce presupun tranzacții

→ NoSQL, când:

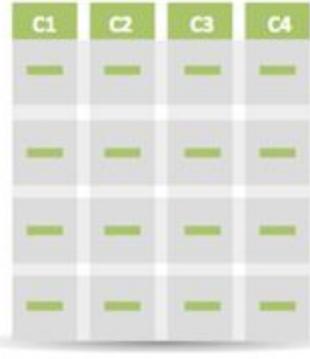
- ◆ cerințele aplicației evoluează mult pe parcursul dezvoltării
- ◆ obiectivele inițiale ale proiectului sunt simple, pentru a putea începe dezvoltarea imediat
- ◆ viteza și scalabilitatea sunt esențiale

2.4.3 Comparație - pe scurt

	Baze de date SQL	Baze de date NoSQL
Tipuri	Un singur tip (SQL) cu variații minore	Multe tipuri: cheie-valoare, baze document, grafuri, obiecte mari (<i>wide-column</i>)

Istoric	Dezvoltate în anii '70 cu primul val de aplicații pentru stocare date. Timp de 4 decenii au fost prima opțiune pentru stocare.	Există încă din anii '60, dar s-au dezvoltat după 2000 pentru a suplini limitările bazelor SQL: scalabilitate, date structurate pe mai multe nivele, date distribuite geografic, dezvoltarea agile
Exemple	MySQL, Postgres, Microsoft SQL Server, Oracle DB	Mongo DB, Cassandra, HBase, Neo4
Modalități de stocare	Datele individuale (de pildă, "angajați") sunt stocate ca linii în tabele, cu fiecare coloană stocând un anume câmp predefinit (de pildă, "manager", "data angajării", etc). Datele asociate sunt stocate în tabele separate, iar pentru a le agrega se execută query-uri complexe. De pildă, definim o tabelă "angajați" și o tabelă "sucursale". Pentru a identifica adresa locului de muncă al unui angajat, baza de date va potrivi/relaționa și agrega datele din cele 2 tabele.	Variază în funcție de tipul de bază de date. De pildă, cele cheie-valoare seamănă cu bazele de date SQL, doar că au doar 2 coloane ("cheie" și "valoare"), cu informațiile mai complexe stocate ca BLOB ²³ în coloana "valoare". Bazele de date tip document renunță complet la modelul tabelar, stocând toate datele liniar într-un document în format JSON, XML sau similar, care poate organiza datele ierarhic.
Schema	Structură rigidă, tipurile de date sunt definite în avans. Pentru a adăuga un nou tip de date, întreaga bază de date trebuie modificată (timp în care nu poate fi activă în producție)	Dinamică, adăugând cel mult reguli de validare a datelor. Aplicațiile pot adăuga câmpuri pe parcurs și , spre deosebire de tabelele SQL, câmpuri diferite pot fi stocate împreună dacă e necesar.

²³ Binary Large OBject, o colecție binară de date stocate ca o singură entitate într-o bază de date

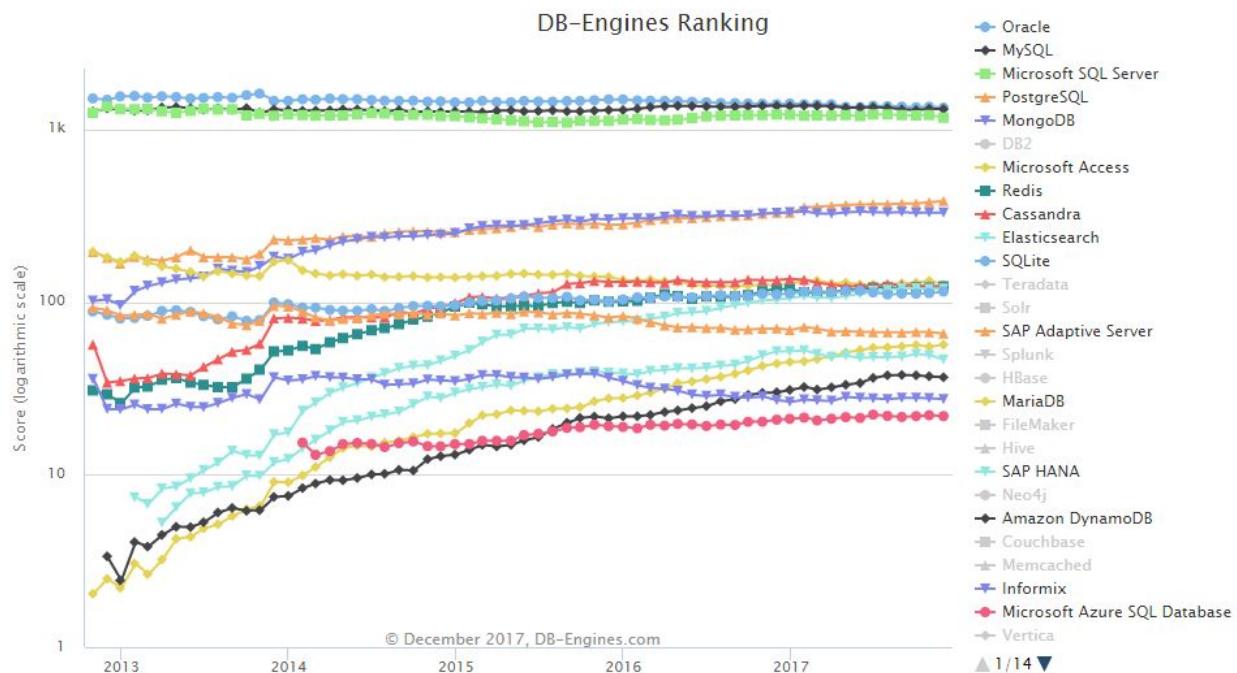
		
Scalare	Verticală: pentru a crește capacitatea trebuie adăugate resurse unui singur server. Distribuția pe mai multe servere este complicată și avantajele relaționării - integritate și atomicitate - sunt parțial pierdute. O soluție recentă sunt soluțiile cloud la nivelul sistemului de operare.	Orizontală: pentru a crește capacitatea, un administrator are nevoie doar să adauge mai multe servere sau instanțe cloud. Baza de date răspândește datele între resursele disponibile după cum este necesar.
Model de dezvoltare	Unele open-source (Postgres, MySQL), altele nu (Oracle DB)	Exclusiv open-source
Suportă tranzacții (atomice)	Da, actualizările pot fi configurate să se execute în totalitate sau deloc. Un mecanism de rollback e definit pentru a asigura revenirea datelor la starea inițială în caz de insucces.	Până la un anumit nivel da, însă în majoritatea cazurilor logica trebuie implementată la nivel de aplicație.
Manipulare a datelor	Limbajul diferă foarte puțin între implementări, bazându-se pe SQL	Prin interfețe orientate obiect (conectori ORM)
Consistența datelor	Poate fi configurată pentru o consistență sporită.	Depinde de produs. Unele se asigură de consistența datelor (de pildă Mongo DB), iar altele nu (de exemplu Cassandra)

2.4.4 Studiu de caz: Oracle DB vs Mongo DB

Piața bazelor de date este estimată la circa 50 de miliarde de dolari în prezent. Este dominată de bazele de date relaționale, totuși creșterea acestei categorii este de circa 5% în ultimul an, față de 31% în cazul celor non-relaționale.

Rank			DBMS	Database Model	Score		
	Dec 2017	Nov 2017			Dec 2017	Nov 2017	Dec 2016
1.	1.	1.	Oracle +	Relational DBMS	1341.54	-18.51	-62.86
2.	2.	2.	MySQL +	Relational DBMS	1318.07	-3.96	-56.34
3.	3.	3.	Microsoft SQL Server +	Relational DBMS	1172.48	-42.59	-54.17
4.	4.	4.	PostgreSQL +	Relational DBMS	385.43	+5.51	+55.41
5.	5.	5.	MongoDB +	Document store	330.77	+0.29	+2.09
6.	6.	6.	DB2 +	Relational DBMS	189.58	-4.48	+5.24
7.	7.	↑ 8.	Microsoft Access	Relational DBMS	125.88	-7.43	+1.18
8.	↑ 9.	↑ 9.	Redis +	Key-value store	123.24	+2.05	+3.34
9.	↓ 8.	↓ 7.	Cassandra +	Wide column store	123.21	-1.00	-11.07
10.	10.	↑ 11.	Elasticsearch +	Search engine	119.78	+0.37	+16.51

Topul general (pe criterii de popularitate) realizat de site-ul *db-engines* arată evoluția ascendentă a bazelor non-relaționale în ultimii ani, în comparație cu cele relaționale:



Deși, de fapt și de drept, se poate observa că de cea mai bună evoluție se bucură noile baze de date hibride (precum SAP HANA).

Oracle și MongoDB pot fi considerați exponenți ai celor 2 categorii. Sunt, de altfel, reprezentanții pe locul 1 la popularitate pentru fiecare dintre ele.

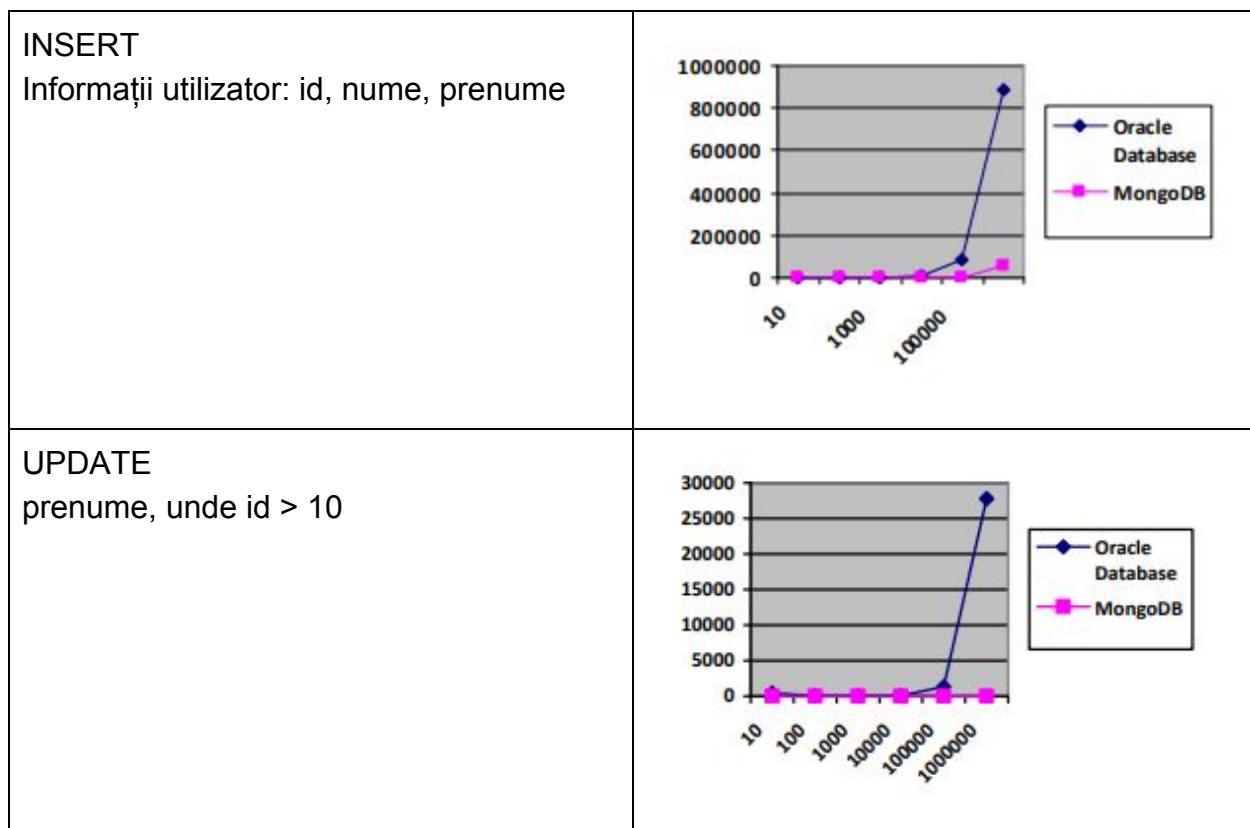
Câteva date despre fiecare sistem:

Nume	MongoDB	Oracle
Model	colecție de documente	relațional
Dezvoltator	MongoDB, începând din 2009	Oracle, începând din 1980
Versiunea curentă (decembrie 2017)	3.6.0	12.2.0.1
Licență	open-source	comercială
Limbaj de implementare/SO	C++/Linux, OS X, Solaris, Windows	C și C++/AIX, HP-UX, Linux, OS X, Solaris, Windows
Metode de acces/API	protocol propriu folosind JSON	ODP.NET, Oracle Call Interface (OCI), JDBC, ODBC
Limbaj de scripting pe server	JavaScript	PL/SQL
Metode de partitioare între noduri	orizontală (sharding)	partiționare orizontală (optională)
Metode de replicare (pentru stocarea redundantă pe mai multe noduri)	replicare master-slave	replicare master-master sau master-slave
Consistență datelor	imediată (optională) sau circumstanțială	imediată
Integritatea datelor	N/A	chei externe

		ACID ²⁴
Controlul accesului, permisiuni, securitate	drepturi de acces și roluri pentru utilizatori	granularitate mică a permisiunilor conform standardului SQL
Dimensiunea maximă a unui camp	16MB	4KB
Capabilități de agregare map-reduce	Da	Nu

Am văzut mai sus (în capitolele 2.4.2 și 2.4.3) avantajele fiecărui sistem.

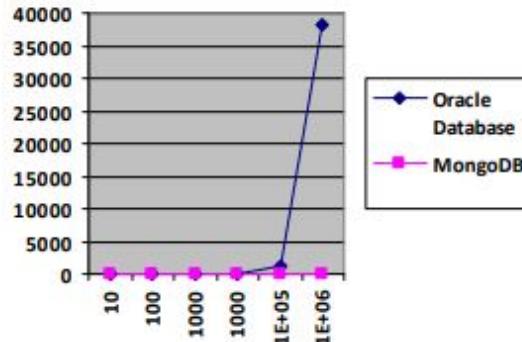
Mai jos, o scurtă comparație din punct de vedere al performanței pentru operații de bază pe seturi de date de până la 1000000 de înregistrări²⁵.



²⁴ ACID - Atomicitate, Consistență, Izolare, Durabilitate - un set de proprietăți al tranzacțiilor în baze de date ce garantează validitatea datelor chiar și în caz de erori, căderi de curent, etc. Spunem că o tranzacție satisfacă aceste proprietăți când, deși implică mai multe schimbări în diverse tabele, poate fi privită ca o singură operație logică.

²⁵ datele sunt preluate din lucrarea "MongoDB vs Oracle" - Alexandru Boicea

`DELETE
unde id > 10`



Concluzia este că, liniar, MongoDB este mult mai rapid. Pentru relaționare, complexitatea va crește logaritmic.

Când se recomandă Oracle DB?

Pentru aplicațiile care implementează tranzacții complexe (sisteme de gestiune, sisteme bancare, etc) sau acolo unde este necesar suportul tehnic imediat, precum și garantarea unor tempi de răspuns (de pildă aplicații enterprise de telefonie), Oracle este soluția.

Când se recomandă Mongo DB?

În orice alt caz: dezvoltarea este mai simplă și mai rapidă iar costurile sunt cu peste 70% mai mici (luând în considerare atât licența cât și costul hardware și productivitatea).

În principiu, când performanța și scalabilitatea sunt mai importante decât integritatea datelor.

Evident, pentru aplicațiile vaste, enterprise, un sistem hibrid este soluția: tranzacțiile pe Oracle și restul cu Mongo DB.

3 Proiectarea aplicației

Se vor livra următoarele:

1. Prototipul fizic pentru măsurarea calității aerului
2. O aplicație de măsurare a calității aerului care se instalează pe prototipul fizic
3. O aplicație web full-stack JavaScript cu interfață prietenoasă cu utilizatorul care permite administrarea și vizualizarea măsurătorilor, precum și funcționalități adiționale

3.1 Sistemul de măsurare

3.1.1 Descriere

Pentru realizarea unei hărți a orașului în ceea ce privește poluarea am identificat necesitatea unui dispozitiv mobil care să mapeze poziția geografică a măsurătorilor cu valorile concentrațiilor de gaze toxice și să stocheze aceste date pentru folosirea ulterioară.

Am ales platforma *Arduino* pentru realizarea prototipului datorită versatilității proiectării hardware și software a acesteia pentru proiecte de mici dimensiuni, fără a presupune cunoștințe avansate de electronică pentru proiectare.

Ce este Arduino?

*Arduino*²⁶ este o platformă *open-source* de dezvoltare a dispozitivelor electronice bazată pe componente hardware și software ușor de folosit.

Plăcuțele *Arduino* sunt proiectate, în mare, pentru a citi date de intrare (o apăsare pe buton, un mesaj SMS, lumina pe un senzor) și a le transforma în date de ieșire (aprinderea unor leduri, activarea unui motor, publicarea unui articol online). Utilizatorul programează microcontroller-ul plăcuței trimițându-i un set de instrucțiuni (bazat pe limbajul C).

Scopul *Arduino* este realizarea rapidă de prototipuri, publicul țintă fiind studenți cu oarece experiență în electronică și programare. Versatilitatea platformei permite folosirea acesteia pentru o plajă largă de aplicații, de la un simplu întreupător la aplicații IoT, imprimante 3d, accesorii portabile inteligente, etc.

²⁶ <https://www.arduino.cc/en/Guide/Introduction>

Detalierea proiectării electronice a fiecăreia dintre componentele folosite este în afara scopului acestei lucrări.

3.1.2 Proiectarea dispozitivului

Selectia componentelor

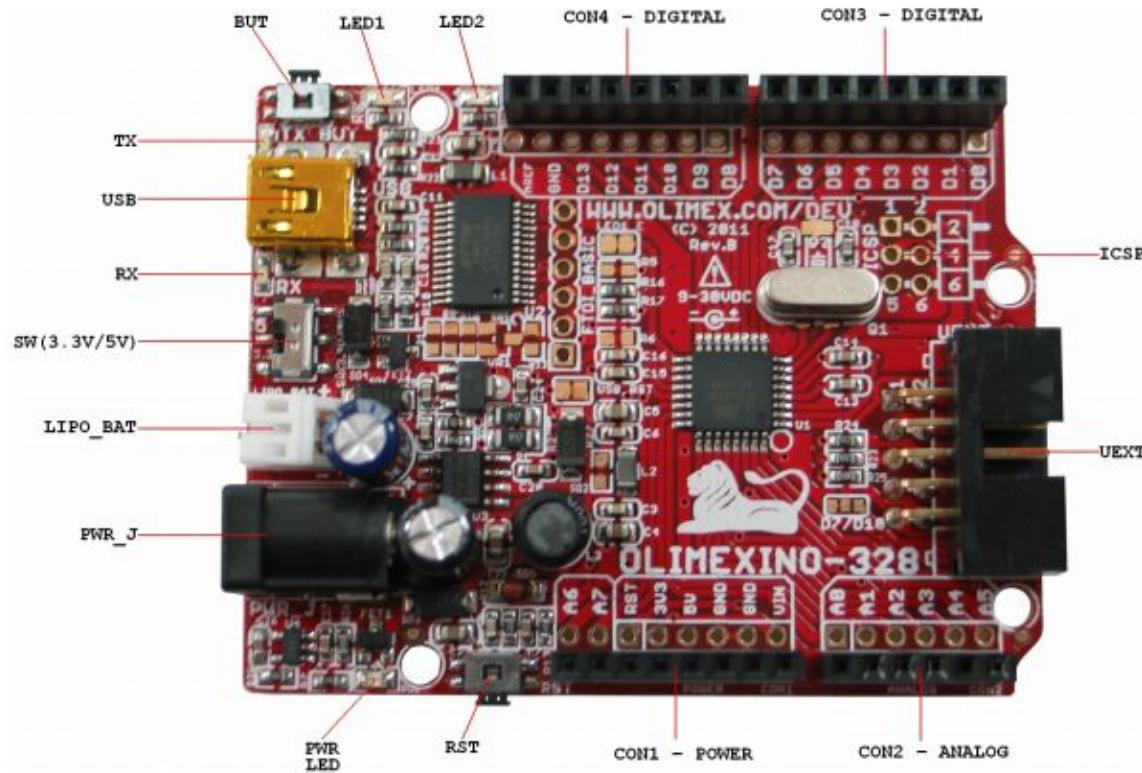
Pentru construcția dispozitivului am ales următoarele componente, din motivele descrise mai jos:

- plăcuța Arduino Industrial (platforma Olimexino 328) - este similară cu Arduino UNO și compatibilă cu mediul de dezvoltare Arduino, dar permite în plus:
 - alimentare cu baterii și reîncărcarea bateriei Li-Ion conectate
 - suportă conectare de tip UEXT (facilitează conectarea modulului GPS)
- modulul integrat GPS; dispune de:
 - antenă încorporată
 - consum: 38mA
 - funcționează între -40 - +85 grade C
 - GPS protocol NMEA (suportat de majoritatea bibliotecilor)
 - conector UEXT
- senzori de măsurare a concentrației gazelor:
 - MQ-4: sensibil la gaz metan (CH_4)
 - MQ-9: sensibil la monoxid de carbon (CO), gaz lichefiat (LPG), gaz metan (CH_4)
 - MQ-135: sensibil la o serie de gaze ce determină calitatea aerului: amoniac(NH_3), oxizii de azot(NO_x), alcool, benzen, fum, dioxid de carbon(CO_2), s.a
- Baterie reîncărcabilă de 3.7V (ce poate fi reîncărcată direct din placă Arduino, în timp ce aceasta e conectată via mini USB la un computer)
- Modulul integrat microSD (cu suport software: biblioteca Arduino IDE SD), pentru stocarea datelor în timpul deplasării

Proiectarea circuitului

Asamblarea circuitului se realizează conform specificațiilor fiecărei componente, după cum urmează:

Placa Olimexino 328 (Arduino Industrial) se conectează în mod similar cu Arduino UNO (majoritatea componentelor sunt proiectate pentru Arduino). Schema plăcii²⁷ este următoarea:

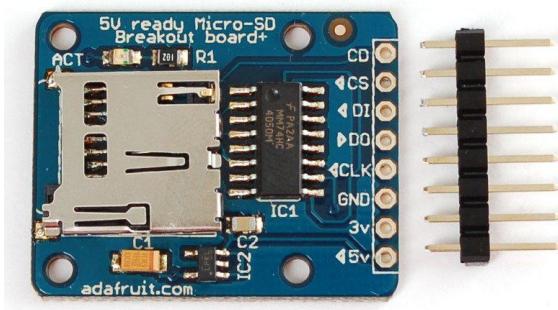


Pentru alimentare și programare, placa se conectează la un computer cu un cablu miniUSB - USB. Pentru mobilitate, placa se poate alimenta de la **bateria** de 3.7V și poate stoca măsurătorile pe un card microSD.

Modulul integrat microSD

Deși cardul SD trebuie alimentat la un curent continuu de 3.3V, modulul are integrat un regulator de tensiune astfel încât să suporte 3.3V sau 5V.

²⁷ Olimexino-328 development board Users Manual



Împreună cu un conector de 8 pini, modulul SD este conectat la placuță în felul urmator:

Modul SD pin	Arduino pin
5V	5V
GND(ground)	GND
CLK(clock)	D13
DO(digital output)	D12
DI(digital input)	D11
CS(chip/slave select)	D10

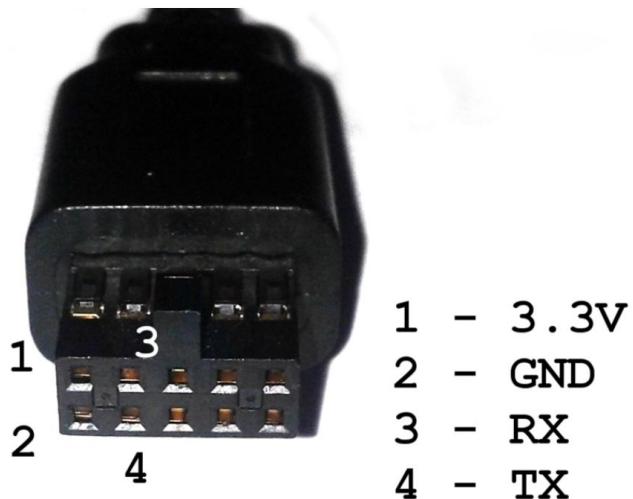
Se va ignora pin-ul CD(*card detect*). Prezența cardului va fi detectată la nivel software.

Pentru compatibilitate între dispozitiv și calculatoare, cardul va fi formatat folosind sistemul de fișiere FAT32.

Modulul GPS

Modulul GPS se conectează fie la portul UEXT, fie la pini de citire digital și alimentare, ca mai jos, unde:

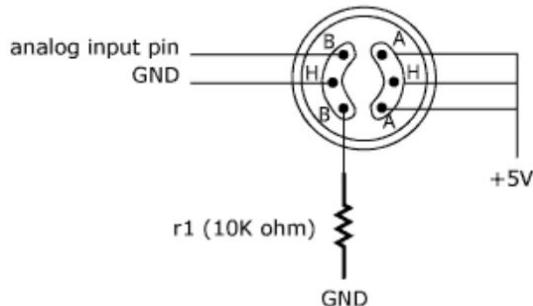
- TX se conectează la pin-ul D3 al Arduino
- RX se conectează la pin-ul D4 al Arduino



În modul conectat la computer, portul UEXT nu poate fi folosit, deoarece este considerat prioritățि ca port intrare/ieșire.

Senzorii de gaz

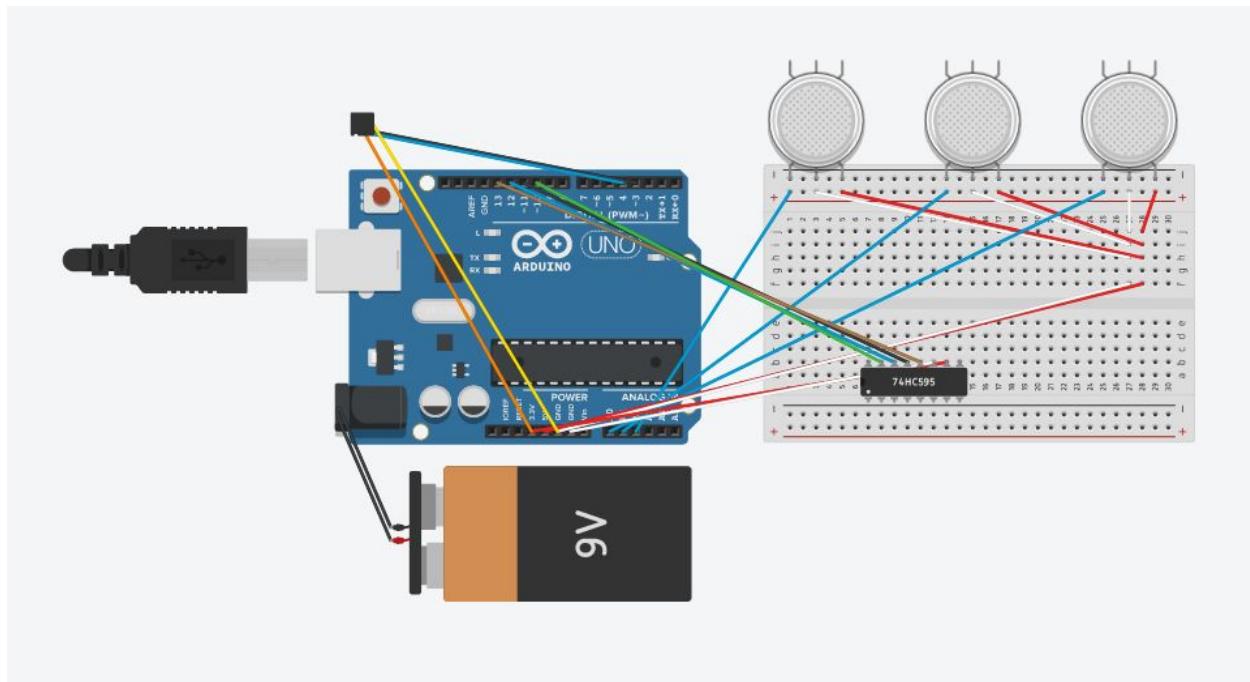
Fiecare senzor este cablat după schema următoare:



Senzorii sunt conectați precum în tabelul de mai jos. Pin-ul DO (*digital output*) nu este folosit, deoarece poate fi programat să ia doar valorile 0 sau 1 (e folosit de pildă pentru alarme), iar noi suntem interesați de valoarea brută măsurată.

MQ-4	VCC	5V
	GND	GND
	DO	-
	AO	A2
MQ-9	VCC	5V
	GND	GND
	DO	-
	AO	A0
MQ-135	VCC	5V
	GND	GND
	DO	-
	AO	A1

Asamblând toate componentele, rezultă schema următoare (simulare cu ajutorul aplicației online www.tinkercad.com/circuits):

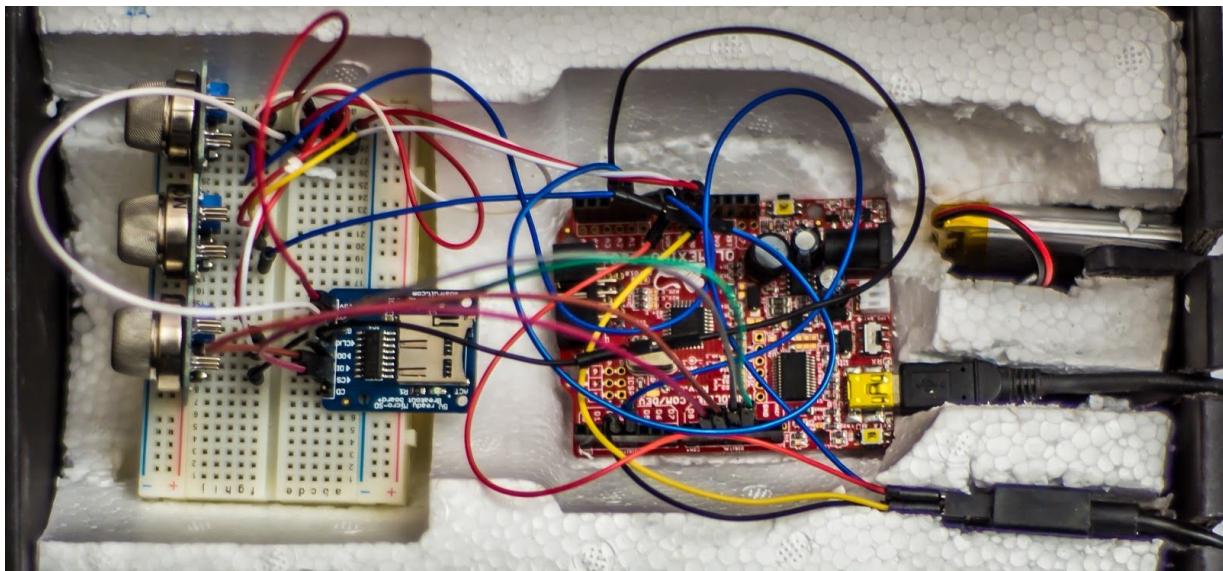


28

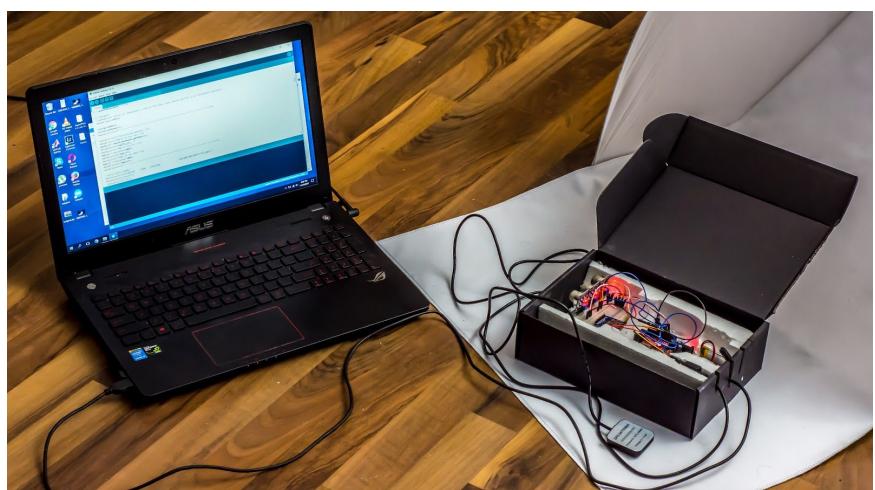
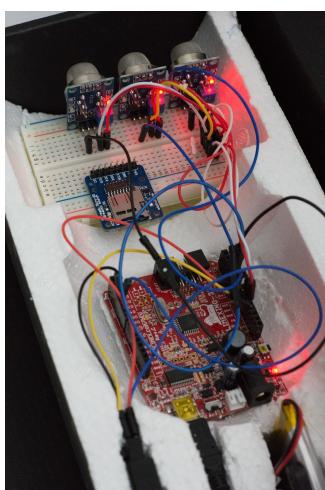
Asamblarea circuitului:

Circuitul a fost asamblat după schema simulată și validată în *Tinkercad*. Temporar, pentru prototip, am folosit o cutie de carton și polistiren pentru protejarea circuitului. Prototipul arată precum în imaginile următoare:

²⁸ Datorită numărului limitat de elemente din *Tinkercad*, am folosit componente diferite în schema, similare celor folosite pentru placă, baterie, senzori și modulul SD.



Modulul SD conectat, dispozitivul deconectat de la sursa de alimentare



Dispozitivul conectat la calculator via interfața serială (USB)

3.1.3 Programarea circuitului

Programarea circuitului se realizează folosind software-ul dedicat Arduino IDE și limbajul C++. Programul este încărcat pe placă via USB.

Configurarea mediului de lucru

Se conectează prin portul USB placa asamblată conform configurației descrise în capitolul anterior.

Se instalează ultima versiune de Arduino IDE (1.6.13) și se configurează astfel:

→ Meniul *Tools*:

- ◆ *Board*: se selectează “Arduino Duemilanove or Diecimilla”
- ◆ *Processor*: se selectează “ATmega328”
- ◆ *Port*: se selectează portul activ
- ◆ *Programmer*: se selectează “AVRISP mkII”
- ◆ Se pornește *Serial Monitor*:
 - Se configerează: *baud rate* = 9600

Se crează un nou proiect.

Se copiază în proiect biblioteca TinyGPS, pe care o vom folosi pentru parsarea datelor provenite de la modulul GPS în formatul NMEA 0183.

Programarea microprocesorului

Scrierea pe serială

Se încarcă biblioteca integrată pentru scrierea pe serială:

Sketch > Include Library > SoftwareSerial

```
#include <SoftwareSerial.h>
```

Scrierea pe serială se poate face cu funcțiile dedicate, după ce clasa a fost inițializată:

```
Serial.begin(DEFAULT_BAUD_RATE);  
Serial.println(...);
```

Scrierea pe cardul SD

Se încarcă biblioteca integrată pentru interfațarea cu modulul SD:

Sketch > Include Library > SD

```
#include <SD.h>
```

Se initializează cardul:

```
pinMode(10, OUTPUT); //se rezerva pin-ul hardware SS pentru ieșire  
File SDfile = SD.open(sdFileName, FILE_WRITE);  
int count = 0;
```

Pentru ușurință testării, se implementează o funcție care să alterneze scrierea datelor:

- Pe cardul SD, când modulul SD este conectat și există un card activ și formatat corespunzător (FAT16 sau FAT32)
- Pe interfața serială, altfel

```
if (SDfile){
    SDfile.println(data);
    count++;
} else{
    Serial.println(data);
}
```

Când ajunge la o anumita dimensiune, fișierul curent (în modul SD) este închis și se deschide un nou fișier pentru scriere.

Citirea senzorilor

Valoarea R_s se citește pe pin-ul analog pentru fiecare senzor conectat:

```
int mq9_sensorValue = analogRead(A0);
int mq135_sensorValue = analogRead(A1);
int mq4_sensorValue = analogRead(A2);
```

Se calibrează senzorul MQ-135: după o perioadă de încălzire de 24 de ore, se calculează și stochează valoarea R_o , funcție de valorile a, b și CO_2 ppm identificate pe baza datelor tehnice ale producătorului (capitolul 2.1.2):

```
#define PARA          116.6020682
#define PARB          2.769034857
#define ATMOCO2        403.13
[...]
float rZero = mq135_sensorValue * pow((ATMOCO2/PARA), (1./PARB));
Serial.println(rZero);
[...]
#define RZERO         161.95
```

Similar se pot calibra și MQ-4 și MQ-9. Totuși, deoarece nu avem date precise pentru calibrarea celorlalți senzori (adică un mediu controlat cu o concentrație știută de CO sau CH_4) vom folosi valori relative, considerând minimul măsurat ca cea mai bună calitate a aerului.

Concentrațiile exacte din fiecare gaz sunt subiect pentru o evoluție ulterioară a aplicației.

```
float ppmCO2 = PARA * pow(mq135_sensorValue/RZERO), -PARB);
```

Valorile citite pentru fiecare senzor, precum și concentrația de CO₂ sunt scrise pe interfață de ieșire (fie fișier, fie serială)²⁹.

Citirea informațiilor GPS

Se citesc caractere de pe serială aferente portului conectat pentru citire la o frecvență de 9600 baud

```
#define RXD          4
#define TXD          3
SoftwareSerial gpsSS(RXD, TXD);
TinyGPS gps;
gpsSS.begin(9600);
```

până la validarea (cu *TinyGPS*) unei propoziții tip *GPGGA* sau *GPRMS*, sau expirarea timpului predefinit:

```
for (unsigned long start = millis(); millis() - start < READ_INTERVAL;)
{
    while (gpsSS.available())
    {
        char c = gpsSS.read();
        if (gps.encode(c)){ // returneaza true la validarea unei propozitii
            gpsDataAvailable = true;
        }else{
#ifndef DEBUG
            gpsDataAvailable = false;
#endif
        }
    }
}
```

Folosind biblioteca *tinyGPS*, se interpretează câmpurile și se scriu în fișier sau pe portul de ieșire.

```
if (gpsDataAvailable)
{
    float flat, flon;
    unsigned long age;
    gps.f_get_position(&flat, &flon, &age);
    print_int(gps.satellites(), TinyGPS::GPS_INVALID_SATELLITES);
```

²⁹ Deoarece citirea se face analogic, vor exista variații ale valorii citite în funcție de temperatură. Se recomanda o calibrare la fiecare masuratoare (calibrarea se poate face prin mai multe citiri succesive la fiecare masuratoare)

```

print_int(gps.hdop(), TinyGPS::GPS_INVALID_HDOP);
print_float(flat, TinyGPS::GPS_INVALID_F_ANGLE, 6);
print_float(flon, TinyGPS::GPS_INVALID_F_ANGLE, 6);
print_date(gps);
}

```

Programul principal

În programul principal se inițializează toate variabilele și porturile:

- se inițializează citirea scrierea pe serială la *9600 baud*
- se inițializează obiectele GPS și SD card
- se scriu valorile R_o curente pentru calibrarea senzorilor
- se scrie header-ul tabelului cu măsurători, pentru a identifica ordinea câmpurilor

Apoi se pornește o buclă infinită, care, la fiecare 2 secunde, citește date GPS și, dacă poziția geografică e identificată, procesează datele senzorilor și le scrie pe o nouă linie. Pentru delimitare între câmpuri se folosește caracterul ‘|’ (*pipe*).

Un exemplu concret de date de ieșire este următorul:

LAT	LONG	DateTime	Age	MQ4	MQ9	MQ135	C02_ppm
44.370079	26.152246	11/11/2017 17:38:22	361	148	334	106	408
44.370082	26.152244	11/11/2017 17:38:24	366	148	333	106	408
44.370086	26.152240	11/11/2017 17:38:26	369	148	334	106	408
44.370090	26.152240	11/11/2017 17:38:28	376	148	333	106	408
44.370094	26.152240	11/11/2017 17:38:30	384	148	333	106	408
44.370098	26.152244	11/11/2017 17:38:32	395	148	334	107	413
44.370105	26.152250	11/11/2017 17:38:34	401	148	333	107	413
44.370105	26.152259	11/11/2017 17:38:36	406	148	333	106	408

Imediat ce este alimentată, placa va rula programul și va scrie, în funcție de disponibilitate, pe portul serial de ieșire sau într-un fișier pe cardul SD.

3.1.4 Îmbunătățiri viitoare (sau *TODO list*)

Pe parcursul dezvoltării dispozitivului, o serie de *workaround*³⁰-uri au fost necesare față de datele propuse în faza de proiectare:

- dispozitivul trebuie alimentat via USB, deoarece senzorii de gaz consumă foarte repede bateria

³⁰ *workaround* - cale ocolitoare, alternativă, pentru rezolvarea unei probleme

- la conectarea modului GPS pe interfața UEXT apar probleme de scriere pe serială, în consecință modulul a fost conectat direct la porturile de alimentare și analogice
- am identificat că variația valorilor măsurate în funcție de temperatură și umiditate este totuși semnificativă, aşa că în condiții diferite de măsurare au fost necesare recalibrări

Astfel, pentru un produs de sine stătător și vandabil, propun următoarele îmbunătățiri:

- renunțarea la modulul microSD și cel GPS în favoarea comunicării prin wifi cu un smartphone și folosirea locației acestuia; tot pe smartphone vor fi afișate și informațiile, deoarece e necesară o conexiune permanentă pentru asocierea locației cu măsurătorile
- deoarece senzorii au un consum mare de energie, folosirea unei baterii nu este fezabilă, deci se recomandă folosirea încărcării via USB; pentru versatilitate se recomandă microUSB în loc de miniUSB
- adăugarea unui senzor de temperatură și umiditate, pentru a recalibra automat în funcție de valoarea acestora
- proiectarea și dezvoltarea folosind componente independente (doar cele necesare aplicației - microcontroller, modul wi-fi, senzori de gaz, ecran, baterie)
- proiectarea unei carcase rezistente la intemperii

3.2 Aplicația web

3.2.1 Alegerea tehnologiei. Motivații

La alegerea unor tehnologii pentru implementarea unui produs nou, următoarele lucruri trebuie luate în primul rând în considerare:

- stabilitatea tehnologiei: trebuie să se fi "învechit" pe piață, astfel încât problemelor celor mai frecvente să le fi fost identificate deja solutii
- o comunitate vastă și de preferat performanței: astfel încât la întâlnirea unei probleme să existe o plajă largă de experți care să poată oferi o soluție, fără costuri pentru suport
- viitorul tehnologiei pe piață - poate fi estimat în funcție de popularitatea ei

Astfel, am ales:

- pentru aplicația web: full stack JavaScript (cea mai populară tehnologie în momentul de față):
 - ◆ backend:

- Node JS cu *framework*-urile Express JS, Passport, Mongoose
- ◆ frontend:
 - HTML, CSS, JavaScript cu Bootstrap și JQuery
- pentru baza de date: MongoDB

Express.js

Deși pe platforma Node.js se poate implementa direct un server web, există *framework*-uri care optimizează și reduc din timpul de implementare. Cel mai popular este Express.js, deoarece este rapid, simplu de utilizat și are multe module construite auxiliare (de pildă cel de autentificare, cel pentru gestiunea sesiunii, etc).

Am ales Express.js din următoarele considerente:

- codul pentru implementarea de bază a unui server este simplist; după furnizarea câtorva parametri, construcția unei aplicații bazate pe apeluri REST este ușoară
- folosește conceptul de *middleware* (explicit în capitolul 2.3.2); în aplicația curentă îl vom folosi împreună cu *body-parser*, un modul ce parsează mesajul din request pentru accesarea directă a parametrilor pentru diverse validări, inclusiv pentru permisii
- în funcție de URL, apelurile pot fi rutate către fișiere specifice care să implementeze răspunsul (fișiere *ejs*)
- are o comunitate largă, în consecință e ușor de găsit răspuns la probleme și are multe module adiționale dezvoltate, dintre care folosim:
 - ◆ *express-session*: pentru gestiunea sesiunii
 - ◆ *passport*: pentru autentificare
 - ◆ *method-override*: pentru a suporta request-uri HTTP PUT

Mongo DB. Mongoose

Capitolul 2.4 descrie avantajele unei baze de date non-relaționale.

În plus, folosind JavaScript full-stack, este avantajoasă orientarea către o bază de date ce se bazează pe obiecte JSON, astfel încât să poată fi procesate atât de server cât și de client fără a fi nevoie de conversii.

Printre avantajele Mongo am avut în vedere:

- lipsa de restricții în adăugarea de obiecte și de câmpuri obiectelor, necesară dezvoltării prin metoda *agile* și cu atât mai utilă cu cât inclusiv cerințele aplicației sunt identificate pe parcurs; astfel obiectele pot avea inclusiv tipuri de câmpuri diferite față de celelalte din aceeași colecție
- indexarea: fiecare câmp indexat este adăugat într-un tablou sortat, astfel încât căutarea se face foarte repede

- suportul pentru localizare: suportă nativ indexare după coordonate geo-spațiale
- JavaScript și JSON îi sunt native
- posibilitate de scalare: în producție se poate scala orizontal, împărțind fiecare colecție pe mai multe servere
- comunitatea mare și activă; există multe tutoriale și răspunsuri online
- conectorul Mongoose dezvoltat în Javascript
- *mLab*, un serviciu de găzduire în cloud, ușor de folosit, care oferă servicii free pentru baze de date de până la 500MB și scalarea ulterioară în funcție de necesități

Mongoose este un driver ORM³¹. Avantajul său față de driverul nativ MongoDB pentru Node este, pe lângă că facilitează gestiunea obiectelor asociate colecțiilor (*type casting*, validare, construcția de *query-uri*), limbajul de interacționare mai similar cu bazele de date relaționale (cu care sunt obișnuit).

3.2.2 Proiect pilot. Deployment

Pentru găzduirea aplicației PoC se vor folosi variante gratuite de găzduire, astfel:

- aplicația web: platforma Heroku
- VCS: Git pe platforma Heroku
- baza de date: platforma mLab *Database-as-a-Service*

Am configurat următoarele:

- Heroku:
 - ◆ aplicația: <https://enigmatic-ocean-67266.herokuapp.com/>
 - ◆ git³²: <https://git.heroku.com/enigmatic-ocean-67266.git>
- mLab:
 - ◆ cont: *respir.info*
 - ◆ baza de date: *respirinfo*
 - ◆ link conectare:
`mongodb://<dbuser>:<dbpassword>@ds135196.mlab.com:35196/respirinf`
 o

³¹ ORM - Object Relational Mapping, tehnică de conversie a datelor folosind programarea orientată pe obiecte. Creează în limbajul folosit obiecte "virtuale" corespunzătoare modelelor asociate.

³² datele de acces pentru git sunt aceleași ca și pentru contul de utilizator Heroku

Am configurat domeniul *respir.info* să redirecționeze la aplicația heroku via *frame redirect*³³.

Avantajul acestei configurații este că e complet gratuită. Dezavantajele sunt că spațiul de stocare și performanța sunt limitate (totuși suficiente pentru nevoile unui PoC), ba mai mult, serverul Heroku este în SUA iar serverul mLab în Irlanda, determinând un timp mare de răspuns. Aplicația are, la colorarea hărții, și operații non-atomice care, pentru o pagină, fac mai multe apeluri la baza de date. Astfel, viteza de răspuns este puternic impactată.

Mediul de lucru. Redeployment

Am ales pentru dezvoltare platforma **cloud9** a Amazon, din 2 motive:

- mediul de dezvoltare este preinstalat cu software-ul necesar dezvoltării, rulării și distribuirii software-ului, iar noi ușor de adăugat și configurat
- același mediu de dezvoltare poate fi ușor accesat din mai multe locații

Dezavantajul este că resursele sunt limitate pentru contul gratuit.

Am creat un nou spațiu de lucru bazat pe şablonul preconfigurat Node.js.

Aplicația de gestiune pentru *heroku* era preinstalată, așa că am configurat *repository-ul* git. Deployment-ul se face automat la fiecare actualizare, după configurarea *package.json* pentru a include:

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "start": "node index.js"  
}
```

Am rulat următoarele comenzi:

- pentru creare:

```
$ heroku Login  
$ git init  
$ git add .  
$ git commit -m "initial deploy"  
$ heroku create  
$ git push heroku master
```

- pentru fiecare actualizare:

```
$ git commit -m "<mesaj>"  
$ git push heroku master
```

³³ tehnica *frame redirect* permite folosirea unui domeniu scurt și ușor de reținut pentru a trimite către o adresă lungă și complicată. Utilizatorul va tasta în browser domeniul scurt, browserul va continua să afișeze domeniul tastat, iar conținutul va fi încărcat într-un *frame* (spre deosebire de *domain redirect*, unde adresa este schimbată cu adresa destinație și conținutul este încărcat direct)

→ pentru rularea comenziilor remote, pe heroku:

```
$ heroku Logs  
$ heroku run <command> (e.g. heroku run ls node_modules)
```

3.2.2 Configurarea și indexarea bazei de date

Configurarea aplicației

După cum am explicitat pe larg în capitolul 2.4, MongoDB este o bază de date non-relațională și nu are nevoie de proiectare inițială. Singura configurare necesară este setarea linkului în procedura specifică a framework-ului mongoose:

```
var promise =  
mongoose.connect('mongodb://<db_username>:<db_password>@ds135196.mlab.com:35196/respirinfo', {  
  useMongoClient: true  
});
```

Totuși, la nivel de PoC avem o idee despre ce va conține baza pentru început - va avea 3 colecții:

➤ *Users* - va stoca informații cu privire la utilizatori, preferințe și permisiuni - pornind³⁴ de la următoarea structură:

```
{  
  //informatii profil  
  username: String,  
  password: String,  
  firstname: String,  
  lastname: String,  
  email: String,  
  phone: Number,  
  country: String,  
  city: String,  
  registrationDate: Date,  
  
  //preferinte  
  alwaysCenterCity: Boolean,  
  dataEconomy: Boolean,  
  maxLines: Number,  
  maxColumns: Number,  
  
  //permisiuni  
  isContributor: Boolean,  
  isEditor: Boolean,  
  isApprover: Boolean,
```

³⁴ Se presupune că la modificări ulterioare nu se vor șterge câmpurile precizate, astfel nu se tratează aceste cazuri de eroare

```
    isAdmin: Boolean  
}
```

➤ *Measurements* - va stoca măsurători - cu următoarea structură:

```
{  
  time: Date,  
  long: Number,  
  lat: Number,  
  mq4: Number,  
  mq9: Number,  
  mq135: Number,  
  co2ppm: Number  
}
```

➤ *Articles* - va stoca știri și articole pentru blog-ul aplicației:

```
{  
  title: String,  
  imageCover: String,  
  description: String,  
  htmlContent: String,  
  author: {  
    id: {  
      type: mongoose.Schema.Types.ObjectId,  
      ref: "User"  
    },  
    username: String  
  },  
  creationDate: Date  
}
```

Indexarea

Pentru o căutare rapidă după coordonate este necesară indexarea după coordonatele geografice.

Se va crea un indice compus după longitudine și latitudine, în această ordine³⁵, prin apelarea următoarei operații în mediul de lucru pentru baza de date mongo:

```
> use respirinfo  
> db.measurements.createIndex( { long: 1, lat: 1 } )
```

³⁵ La efectuarea unei căutari contează ordinea parametrilor, pentru a obține performanță preconizată.

Câștigul de performanță este o căutare logaritmică în loc de o căutare liniară de ordin 2.

Indexarea pentru celelalte colecții este irelevantă pentru dimensiunea aplicației PoC.

3.2.3 Proiectarea aplicației

Structura

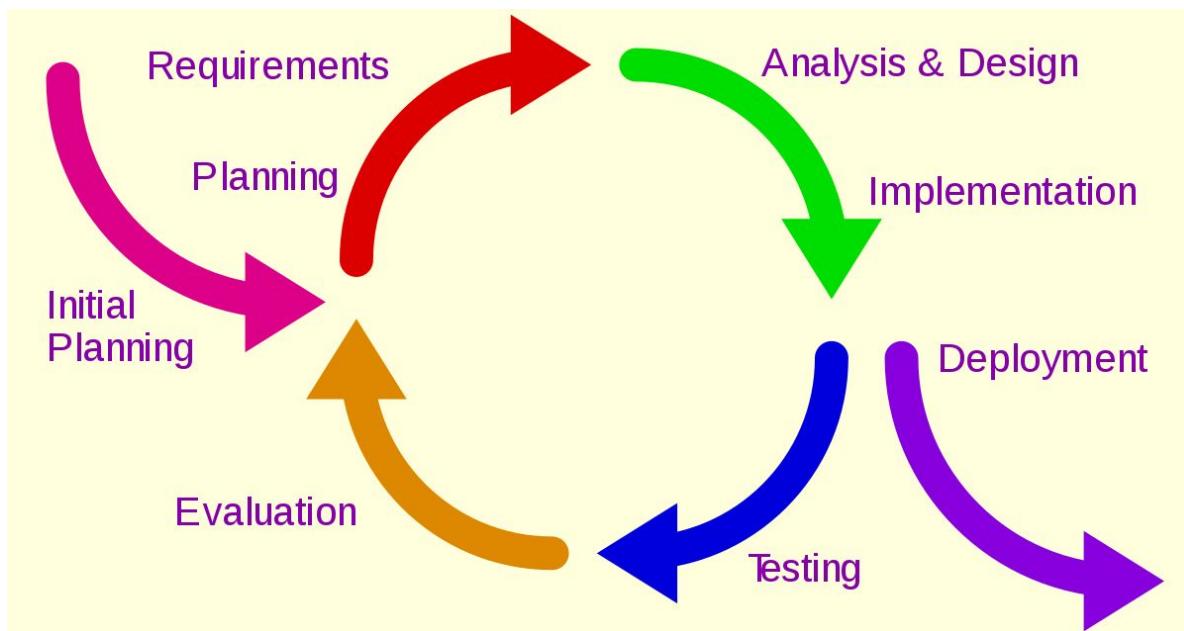
Structura aplicației va fi similară MVC, separând fișierele în directoare după rolul fiecărui, astfel:

- */node_models*: directorul implicit unde *npm* instalează pachetele necesare aplicației
- / - în rădăcină se vor găsi:
 - *index.js* - pagina principală a aplicației
 - *package.json* - fișier ce conține configurarea și dependințele aplicației
- */models*: conține schemele corespunzătoare fiecărei colecții din baza de date (conform structurii precizate în capitolul anterior):
 - *article.js*
 - *measurement.js*
 - *user.js*
- */routes*: gestionează logica aplicației, implementând acțiunile pentru fiecare rută a aplicației
- */middleware*: gestionează acțiunile tip *middleware*, de validare a acțiunilor în cadrul aplicației:
 - dacă un utilizator este autentificat când solicită o acțiune
 - ce permisii are utilizatorul autentificat
 - dacă utilizatorul autentificat este autor pentru un articol
- */views*: include toate paginile aplicației:
 - pagina inițială
 - înregistrare
 - login
 - pagina principală
 - acțiuni generale (încărcare măsurători, aprobări)
 - acțiuni utilizator (administrare, preferințe, date personale)
 - articole (adăugare, afișare, modificare)
 - partiale: header și footer
- */public*: include scripturi generice, configurații CSS și imagini

- */data*: în acest director sunt stocate dinamic fișiere folosite de aplicație (de pildă fișiere cu măsurători încărcate de utilizator)

Dezvoltare

Dezvoltarea aplicației va fi abordată iterativ, fiecare iteratăie având rolul de a adăuga o nouă funcționalitate aplicației.



La nivel de PoC, mi-am propus implementarea următoarelor funcționalități:

- gestiunea utilizatorilor:
 - ◆ înregistrare
 - ◆ login
 - ◆ logout
 - ◆ modificare date personale
 - ◆ preferințe
 - ◆ administrare
- încărcarea măsurătorilor:
 - ◆ încărcare fișier text
 - ◆ parsare fișier și stocarea datelor în baza de date
- afișare hartă:
 - ◆ harta generică
 - ◆ afișare măsurători

- ◆ afișare măsurători în funcție de setările utilizatorului
- ◆ afișare recomandări
- header & footer:
 - ◆ header: afișare meniu personalizat în funcție de permisiuni
 - ◆ footer: adăugare link și pagini pentru contact, misiune și informații generale
- articole:
 - ◆ adăugare
 - ◆ afișare
 - ◆ modificare
 - ◆ modificare în funcție de drepturile utilizatorului (doar de către autor sau administrator)

3.2.3.1 Gestiunea utilizatorilor

Prima iterație presupune realizarea configurațiilor inițiale:

- crearea aplicației:


```
$ npm init
```
- Încărcarea framework-urilor și bibliotecilor:
 - ◆ *express* și *express-session*: framework-ul principal pentru dezvoltarea aplicației web (detalii în capitolul 3.2.1)
 - ◆ *ejs*: framework pentru integrarea codului server cu codul client în paginile aplicației
 - ◆ *mongoose*: ORM pentru interconectarea cu MongoDB
 - ◆ *body-parser*: modul pentru parsarea obiectelor request și response
 - ◆ *passport*, *passport-local* și *passport-local-mongoose*: framework care gestionează autentificarea
 - ◆ *connect-flash*: framework care gestionează mesajele de eroare
 - ◆ *method-override*: modul ce permite folosirea metodelor HTTP PUT (pentru a respecta formatul REST standard)
 - ◆ *formidable*: modul pentru parsarea fișierelor încărcate
 - ◆ *serve-favicon*: modul pentru încărcarea unui *favicon*³⁶

Încărcarea unui modul se face astfel:

- instalare: în directorul principal al aplicației se rulează o comandă de forma:


```
$ npm install express --save
```
- în aplicație:


```
var express = require("express");
```

³⁶ Un *favicon* este o pictogramă specifică unei aplicații web, afișată de obicei în titlul paginii

→ configurarea bazei de date³⁷:

```
mongoose.Promise = global.Promise;
var promise = mongoose.connect('mongodb://localhost/respir_info', {
  useMongoClient: true,
  /* other options */
});
```

→ rutele se vor defini în fișiere separate după funcționalitate, dar sunt încărcate toate în index (ordinea este importantă):

```
var indexRoutes      = require("./routes/index");
var userRoutes       = require("./routes/users");
var actionRoutes     = require("./routes/actions");
var articleRoutes   = require("./routes/articles");

app.use("/", indexRoutes);
app.use("/user", userRoutes);
app.use("/action", actionRoutes);
app.use("/articles", articleRoutes);
```

→ inițializarea aplicației și utilitarelor:

```
app.use(bodyParser.urlencoded({extended: true}));
app.set("view engine", "ejs");
app.use(express.static(__dirname + "/public"));
app.use(methodOverride("_method"));
app.use(flash());
```

→ inițializarea *framework*-ului Passport, pentru autentificare:

```
app.use(require("express-session")({
  secret: "Folosim respir.info pentru informatii privind calitatea aerului.
Masuram, calculam, informam. ASDF1234",
  resave: false,
  saveUninitialized: false
}));
app.use(passport.initialize());
app.use(passport.session());
passport.use(new LocalStrategy(User.authenticate()));
passport.serializeUser(User.serializeUser());
passport.deserializeUser(User.deserializeUser());
```

→ salvarea datelor în sesiunea locală (*middleware*):

```
app.use(function(req, res, next){
  res.locals.currentUser = req.user;
  res.locals.error = req.flash("error");
  res.locals.success = req.flash("success");
```

³⁷ Pentru dezvoltare și testare unitara se va folosi o bază de date locală. Pentru instalare se va folosi configurarea specificată în capitolul 3.2.2

```

        next();
    });

```

→ pornirea serverului pentru a asculta request-uri:

```

app.listen(process.env.PORT, process.env.IP, function(){
    console.log("respir.info server has started...");
});

```

Voi folosi modulul *bootstrap* pentru configurarea interfeței grafice. În directorul *views/partials* se adaugă câte un fișier pentru *header* și *footer*. Acestea vor fi încărcate în fiecare pagina. Astfel, în *header.ejs* încarc versiunile miniaturizate *bootstrap* și *jquery* și adaug un *navbar* ca meniu de navigație între pagini.

Se adaugă următoarele pagini în directorul *views*:

- *landing.ejs*, prima pagină, statică; va conține un *carousel* pentru afișarea câtorva imagini sugestive
- *register.ejs*: înregistrarea unui utilizator; conține un formular ce trimite o interogare HTTP POST pe ruta */register* cu un nume de utilizator și o parolă; câmpurile sunt validate
- *login.ejs*: conține un formular ce trimite o interogare HTTP POST pe ruta */login* cu un nume de utilizator și o parolă; câmpurile sunt validate

Se adaugă următoarele pagini în directorul *views/user*:

- *admin.ejs*: conține un formular ce trimite o interogare HTTP PUT pe ruta */user/admin* conținând un nume de utilizator și permisiunile acestuia
- *preferences.ejs*: conține un formular ce trimite o interogare HTTP PUT pe ruta */user/preferences* ce conține preferințele utilizatorului curent
- *update.ejs*: conține un formular ce trimite o interogare HTTP PUT pe ruta */user/profile* ce conține actualizări ale câmpurilor specifice profilului utilizatorului curent (nume, email, etc)

Deoarece interogările HTTP PUT³⁸ nu sunt suportate nativ, se folosește o instrucțiune de forma:

```
<form action="/user/profile?_method=PUT" method="POST">
```

În aplicația principală configurația aferentă este:

```
app.use(methodOverride("_method"));
```

Rutele vor fi implementate după cum urmează:

- *routes/index.js*:
 - HTTP GET "/" rediectează către pagina *landing.ejs*

³⁸ În aplicația curentă puteam folosi la fel de bine POST în loc de PUT. Dar am preferat să respect convenția REST, în care HTTP PUT este folosit pentru actualizarea unui obiect existent, iar POST pentru crearea unuia.

- HTTP GET “/register” rediectează către pagina *register.ejs*
 - HTTP POST “/register” creează un nou utilizator; în caz de succes rediectează către pagina principală(*map.ejs*, ce va fi creată într-o iterare ulterioară, momentan este doar un *stub*³⁹), altfel către *register.ejs*
 - HTTP GET “/login”, rediectează către pagina *login.ejs*
 - HTTP POST “/login”, autentifică utilizatorul și îi crează sesiunea folosind modulul *passport*:
- ```
router.post("/login", passport.authenticate("local",
 {
 successRedirect: "/map",
 failureRedirect: "/login"
 }), function(req, res){
});
```
- HTTP GET “/logout”, încheie sesiunea curentă și rediectează utilizatorul către pagina principală
- *routes/users.js*:
    - HTTP GET “/profile” încarcă pagina *user/update.ejs* cu datele utilizatorului curent; pagina poate fi accesată doar de un utilizator autentificat:

```
router.get("/profile", middleware.isLoggedIn, function(req, res){
 res.render("user/update", {user: req.user});
});
```

  - HTTP PUT “/profile” actualizează datele utilizatorului curent cu noile valori și rediectează la pagina principală:
- ```
router.put("/profile", middleware.isLoggedIn, function(req, res){
  User.findByIdAndUpdate(req.user.id, req.body.user, function(err,
  updatedUser){
    if(err){
      req.flash("error", err.message);
    } else {
      req.flash("success", "Datele tale personale au fost
actualizate.");
    }
    res.redirect("/map");
  });
});
```
- HTTP GET “/preferences” încarcă pagina *user/preferences.ejs* cu datele utilizatorului curent; pagina poate fi accesată doar de un utilizator autentificat
 - HTTP PUT “/preferences” actualizează datele utilizatorului curent cu noile valori și rediectează la pagina principală

³⁹ termen consacrat în domeniul pentru a desemna o entitate temporară, ce urmează să fie implementată

- HTTP GET “/admin” încarcă pagina *user/admin.ejs* cu datele utilizatorului curent; pagina poate fi accesată doar de un utilizator autentificat cu drepturi de admin:

```
router.get("/admin", middleware.isAdmin, function(req, res){
  res.render("user/admin");
});
```

În *middleware/index.js* se realizează validarea corespunzătoare:

```
middlewareObj.isAdmin = function(req, res, next){
  if(req.isAuthenticated()){
    if(res.locals.currentUser.isAdmin){
      return next();
    }
    req.flash("error", "Nu ai permisiuni pentru aceasta actiune!");
    res.redirect("/map");
  } else {
    req.flash("error", "Trebuie sa fii autentificat pentru aceasta actiune!");
    res.redirect("/login");
  }
}
```

- HTTP PUT “/admin” actualizează permisiunile utilizatorului cerut cu noile valori și redirecțiază la pagina principală; și aici se verifică permisiunile utilizatorului autentificat pentru a evita abuzurile

Pentru fiecare rută, se implementează mesaje de eroare temporare folosind modulul *flash*, astfel:

- În implementarea rutei se salvează mesajul corespunzător:

```
req.flash("error", err.message);
req.flash("success", "Salut, " + user.username + "! Bine ai venit în comunitatea
respir.info!");
```

- În middleware se salvează în sesiune mesajul aferent (modulul *flash* este responsabil să mențină mesajul pentru un singur apel):

```
res.locals.error = req.flash("error");
res.locals.success = req.flash("success");
```

- În pagină se afișează mesajul disponibil; implementarea este în *header.ejs*:

```
<div class="container">
  <% if(error && error.length > 0){ %>
    <div class="border-shadow alert alert-danger" role="alert">
      <%= error %>
    </div>
  <% } %>
  <% if(success && success.length > 0){ %>
    <div class="border-shadow alert alert-success" role="alert">
      <%= success %>
    </div>
  <% } %>
</div>
```

3.2.3.2 Încărcarea măsurătorilor

Se adaugă o pagină în directorul `views/actions`: `measurements_upload.ejs`, care încarcă un dialog de selectare fișiere și trimite o interogare HTTP POST cu fișierele selectate. Se implementează un request `jQuery ajax` pentru a sincroniza procentul de încărcare a fișierului:

```
$.ajax({
    url: '/action/measurements',
    type: 'POST',
    data: formData,
    processData: false,
    contentType: false,
    success: function(data){
        console.log('upload successful!\n' + data);
        $("#alertSuccess").attr("hidden", false);
    },
    xhr: function() {
        var xhr = new XMLHttpRequest();
        xhr.upload.addEventListener('progress', function(evt) {
            if (evt.lengthComputable) {
                // calculez procentul incarcat din fisier
                var percentComplete = evt.loaded / evt.total;
                percentComplete = parseInt(percentComplete * 100);
                // actualizarea corespunzatoare a progress-bar-ului Bootstrap
                $('.progress-bar').text(percentComplete + '%');
                $('.progress-bar').width(percentComplete + '%');
                if (percentComplete === 100) {
                    $('.progress-bar').html('Incarcat');
                }
            }
        }, false);
        return xhr;
    }
});
```

Se adaugă 2 rute în `routes/actions.js`:

- HTTP GET “/measurements/add” încarcă pagina `measurements_upload.ejs`, dacă utilizatorul are drepturi de contributor
- HTTP POST “/measurements” încarcă și parsează un fișier și adaugă măsurătorile aferente în baza de date - folosind modulul `formidable` (permisiunile de contributor ale utilizatorului curent sunt reverificate):
 - Încărcarea fișierului:

```
var form = new formidable.IncomingForm();
//configură incarcarea mai multor fisiere intr-un singur request
form.multiples = true;
//configură directorul unde sunt incarcate fisierele
form.uploadDir = path.join(__dirname, '../data/uploads');
```

```

form.on('file', function(field, file) {
    fs.rename(file.path, path.join(form.uploadDir, file.name));
    fileName = file.name;
});
// trimite un raspuns catre client o data ce fisierele sunt incarcate
form.on('end', function() {
    res.end('Successfully loaded on server');
    // parsez fiecare fisier si salvez datele in baza de date
    parseStoreMeasurements(path.join(form.uploadDir,fileName));
});

```

- parsarea fișierului: voi citi linie cu linie; pentru a fi valid, fișierul trebuie să aibă următorul format:
 - date inițiale, care sunt ignorate
 - separator header: “-----”
 - header, care conține tipul și ordinea câmpurilor, separate prin caractere *pipe* “|”; doar următoarele câmpuri sunt valide:

```

fieldsMapping = {
    'LAT': 'lat',
    'LONG': 'long',
    'MQ4' : 'mq4',
    'MQ9': 'mq9',
    'MQ135': 'mq135',
    'CO2_ppm': 'co2ppm',
    'DateTime': 'time'
}

```

- separator conținut: “-----”
- conținut, unde pe fiecare linie vor trebui să fie date de tipul și în ordinea specificată în header

Pentru parsarea datelor se va implementa o mașină de stări, astfel:

```

lineReader.on('line', function (line) {
    switch(nextStep){
        // astept identificarea separatorului pentru header: '-----'
        case 'SEP_HDR':
            if(line.indexOf('-----') !== -1){
                nextStep = 'HDR';
            }
            break;
        // astept identificarea ordinii campurilor:
        'S|HDP|LAT|LONG|DateTime|Age|MQ4|MQ9|MQ135|CO2_ppm'
        case 'HDR':
            if(line.indexOf('LONG') !== -1){
                linesplit = line.split("|");
                linesplit.forEach(function(element){
                    element = element.trim();
                    if(fieldsMapping[element]){
                        interpret.push(fieldsMapping[element]);

```

```

        } else {
            interpret.push('IGNORE');
        }
    });
nextStep = 'SEP_HDR_END';
}
break;
//astept identificarea separatorului ce marcheaza inceputul continutului:
'-----'
case 'SEP_HDR_END':
if(line.indexOf('-----') != -1){
    nextStep = 'DATA';
}
break;
//citesc campurile in ordinea specificata in header si le salvez in baza de date
case 'DATA':
if(line.indexOf('|') == -1){
    break;
}
linesplit = line.split("|");
var newMeasurement = {};
for (var i = 0; i < linesplit.length; i++) {
    if(interpret[i] != 'IGNORE'){
        console.log("INTERPRET:" + i + " " + interpret[i] + " " +
linesplit[i]);
        newMeasurement[interpret[i]] = linesplit[i];
    }
}
if(Object.keys(newMeasurement).length === 7){
    try{
        Measurement.create(newMeasurement, function(err, newlyCreated){}
    } catch(err) {
        console.log(err.message);
    }
}
break;
default:
break;
}
});

```

3.2.3.3 Afisare hartă

În pagina principală se afișează o hartă cu reprezentarea măsurătorilor făcute.

În directorul *views* se adaugă pagina *main.ejs* care va conține:

- harta: se implementează folosind Google Maps API, astfel:
 - se crează un cont de dezvoltator Google și se cere o cheie pentru folosirea Google Maps

- pentru adăugarea hărții se inserează codul de mai jos, cu cheia obținută în pasul anterior:

```
<div id="map"></div>
<script
src="https://maps.googleapis.com/maps/api/js?key=AIzaSyBJE8W-U7ncWeESEV6z1LbmLdloXC-ooS
U&callback=initMap" async defer></script>
```

- se inițializează harta cu centrul și ordinul de mărime predefinite, se adaugă evenimente la zoom și drag pentru interogarea serverului privind informațiile despre calitatea aerului, apoi se interoghează serverul pentru poziția curentă și se desenează cu diferite culori dreptunghiuri transparente conform valorilor medii ale măsurătorilor în zonele respective:

```
function initMap() {
    //initializez harta cu coordonatele curente
    map = new google.maps.Map(document.getElementById('map'), {
        center: {lat: <%=gmap.defaultCenter.lat%>, lng:
<%=gmap.defaultCenter.lng%>},
        zoom: <%=gmap.zoom%>,
        disableDoubleClickZoom: true
    });
    //adaug evenimente la schimbarea coordonatelor
    map.addListener('dragend', function() {
        getNewAirQData();
    });
    map.addListener('zoom_changed', function() {
        getNewAirQData();
    });

    //se deseneaza
    drawTiles();
}

function drawTiles(){
    //sterge desenul curent
    airqTiles.forEach(function(tile){
        tile.setMap(null);
    });
    airqTiles = [];

    //redesenez conform valorilor reincarcate
    <% gmap.zones.forEach(function(gzone){ %>
        var zoneColor = COLORS.GRAY;
        <% if(gzone.co2ppm > 2000) { %>
            zoneColor = COLORS.GRAY;
        <% } else if(gzone.co2ppm > 1500) { %>
            zoneColor = COLORS.DARK_PURPLE;
        <% } else if(gzone.co2ppm > 1000) { %>
            zoneColor = COLORS.PURPLE;
```

```

        <% } else if(gzone.co2ppm > 750) { %>
            zoneColor = COLORS.LIGHT_PURPLE;
        <% } else if (gzone.co2ppm > 600) { %>
            zoneColor = COLORS.DARK_RED;
        <% } else if (gzone.co2ppm > 550) { %>
            zoneColor = COLORS.RED;
        <% } else if (gzone.co2ppm > 500) { %>
            zoneColor = COLORS.ORANGE;
        <% } else if (gzone.co2ppm > 450) { %>
            zoneColor = COLORS.LIGHT_ORANGE;
        <% } else if (gzone.co2ppm > 350) { %>
            zoneColor = COLORS.YELLOW;
        <% } else if (gzone.co2ppm <= 350) { %>
            zoneColor = COLORS.GREEN;
        <% } %>
        airqTiles.push(new google.maps.Rectangle({
            strokeColor: zoneColor,
            strokeOpacity: 0.25,
            strokeWeight: 2,
            fillColor: zoneColor,
            fillOpacity: 0.25,
            map: map,
            bounds: {
                north: <%= gzone.latMax %>,
                south: <%= gzone.latMin %>,
                east: <%= gzone.lngMax %>,
                west: <%= gzone.lngMin %>
            }
        }));
    <% }); %>
}

```

La fiecare schimbare a coordonatelor (prin drag sau zoom) se face o interogare HTTP POST către server conținând un vector cu coordonatele fiecărei zone de pe harta (pentru utilizatorii neautentificați harta se împarte în 4x4 zone, la cei autentificați în mxn zone, conform preferințelor acestora).

- interpretarea hărții: se poziționează în stânga un *accordion* cu informații statice privind semnificația culorilor de pe hartă și recomandările aferente

Se adaugă 2 rute în *routes/index.js*:

- HTTP GET “/map” împarte harta în zone ($m \times n$ conform preferințelor utilizatorului dacă este autentificat, altfel 4x4), extrage din baza de date măsurătorile medii pentru coordonatele implicate pentru fiecare zonă și încarcă pagina *main.ejs*:

- pentru a agrega zonele în care este împărțită harta, în condițiile implementării non-blocante și fără stare a Node.js, implementarea este non-paralelă, recursivă, reținându-se într-o variabilă globală parametru ultimei măsurători⁴⁰
- HTTP POST “/map” face aceleași acțiuni precum GET, dar folosește coordonatele trimise în request (pentru mutarea coordonatelor hărții)

3.2.3.4 Header & footer. Stilizare

Stilizarea se realizează folosind în principal elemente *Bootstrap* care se modifică după gust. Un avantaj al bootstrap e că permite ca interfața să arate decent pe smartphone-uri fără prea multe bătăi de cap.

Se adaugă 2 pagini statice în views/articles:

- *static_mission.ejs*: cu informații despre scopul proiectului
- *static_pollution.ejs*: cu informații generale despre poluare

Se adaugă în footer legături către paginile de mai sus și un modul tip *modal* pentru contact:

```
<div class="modal fade" id="contact" role="dialog">
  [...]
```

3.2.3.5 Articole

Se adaugă elementele principale *CRUD* pentru gestiunea de articole pentru o funcționalitate tip *blog* (mai puțin *delete*).

Se adaugă următoarele pagini în directorul *views/articles*:

- *new.ejs*: conține un formular ce trimite o interogare HTTP POST pe ruta */articles* conținând informații pentru un articol (titlu, imagine coperta, descriere, conținut HTML⁴¹)
- *list.ejs*: afișează o listă de articole; fiecare include un link către pagina de vizualizare detaliu (view.ejs)
- *view.ejs*: afișează detaliile unui articol; dacă utilizatorul autentificat este autorul articoului, se afișează opțiunea de modificare a articoului:

```
<% if(currentUser && article.author.id.equals(currentUser._id)){ %>
  <a class="btn btn-xs btn-warning" href="/articles/<%= article._id
%>/edit">Editez</a>
<% }%>
```

- *edit.ejs*: conține un formular ce afișează informațiile curente din articol și permite modificarea acestora, trimițând o interogare HTTP PUT pe ruta */articles/<id_articol>*

⁴⁰ codul este disponibil în anexă, capitolul 7.2

⁴¹ implementarea unui editor HTML pentru conținut este în afara scopului acestui proiect

Rutele sunt implementate în *routes/articles.js* astfel:

- HTTP GET “/articles” extrage toate articolele existente în bază și încarcă pagina *views/articles/list.ejs*
- HTTP GET “/articles/new” rediectează către pagina *views/articles/new.ejs*
- HTTP POST “/articles” adaugă articolul din request, apoi rediectează către ruta GET “articles”
- HTTP GET “/articles/<id>” extrage din baza de date articolul identificat prin <id> și încarcă pagina *views/articles/view.ejs*
- HTTP GET “/articles/<id>/edit” extrage din baza de date articolul identificat prin <id>, validează că utilizatorul curent este autorul articolului sau administrator și încarcă pagina *views/articles/edit.ejs*
- HTTP PUT “/articles/<id>” actualizează articolul identificat prin <id> cu informațiile din request și rediectează către ruta HTTP GET “/articles/<id>”

Dezvoltările viitoare sunt subiect pentru noi iterații. După fiecare iterație, funcționalitatea anterioară trebuie păstrată sau îmbunătățită, apoi se face un nou *deployment*.

Testare

În scopul curent (PoC) se realizează iterativ doar testare nominală a interfeței grafice pentru fiecare funcționalitate adăugată. Testările funcțională, de performanță, de securitate și regresii nu-și au rostul la acest nivel.

4 Explotarea aplicației

În stadiul curent, aplicația are o procedură relativ greoie și înceată pentru exploatare, dar îmbunătățirile sunt în afara scopului lucrării de față (scopul PoC este demonstrarea posibilităților).

Astfel, după instalarea aplicației web pe serverul gratuit de găzduire, încărcarea informațiilor se realizează în cicluri ce presupun:

- efectuarea de măsurători folosind sistemul hardware
- încărcarea măsurătorilor în aplicația web

De asemenea, se pot adăuga articole pentru informarea utilizatorilor.

Explotarea aplicației constă, după cum văd lucrurile, în informarea și educarea utilizatorilor folosind aceste date.

Efectuarea măsurătorilor

În cadrul proiectării sistemului hardware, intenția a fost ca acesta să fie mobil, aşa că am avut în vedere folosirea unei baterii. Am omis însă faptul că senzorii de gaz au un consum consistent de energie, în consecință e necesară alimentarea permanentă.

Astfel, utilizatorul poate opta:

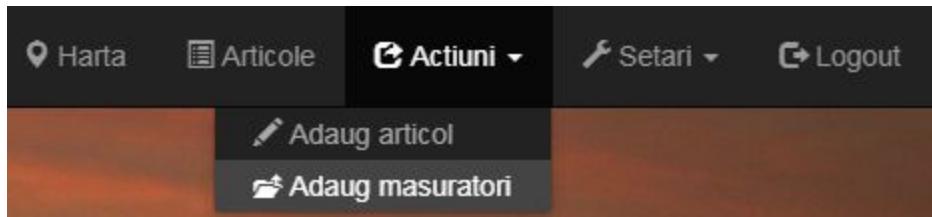
- să introducă un card SD în slotul dedicat și să folosească o sursă oarecare de alimentare, apoi să copieze datele de pe card în calculator
- să nu folosească un card SD, să conecteze dispozitivul direct la un computer și să copieze datele de pe serială (citite folosind aplicația Arduino) într-un fișier text (eu am folosit această variantă, pentru a urmări în timp real datele măsurate)

Utilizatorul va așeza dispozitivul pornit într-un mijloc de transport mobil cu acces imediat la aerul atmosferic (am folosit o mașină decapotată și cu geamurile deschise) și se va deplasa în zonele de interes.

Încărcarea măsurătorilor în aplicația web

Pentru încărcarea măsurătorilor, utilizatorul trebuie să aibă drepturi de contributor.

Folosind meniul *Acțiuni > Adaug măsurători din header* utilizatorul autorizat va accesa pagina de încărcare măsurători:



unde va încărca fișierele obținute anterior.

Trimite masuratori

Incarc fisier

Name	Date modified	Type
measurements_2017.11.30.txt	11/30/2017 1:52 PM	Text Document
measurements_2017.12.09.txt	12/9/2017 5:46 PM	Text Document
measurements_test - Copy (2).txt	11/30/2017 3:52 PM	Text Document
measurements_test - Copy.txt	11/30/2017 3:53 PM	Text Document
measurements_test.txt	11/30/2017 3:53 PM	Text Document
measurements1.txt	11/30/2017 1:52 PM	Text Document
measurements2.txt	11/30/2017 1:52 PM	Text Document

File name: measurements_2017.12.09.txt Open Cancel

Misiune Contact Despre poluare

Datele sunt actualizate automat în baza de date cu noile informații. Utilizatorii vor putea astfel vedea imediat actualizările în căutările lor pe hartă.

5 Ghid de utilizare

Acest scurt ghid își propune oferirea tuturor informațiilor necesare pentru a înțelege ce și cum se poate face folosind aplicația web. Pe scurt, scopul aplicației este informarea utilizatorilor în ce privește calitatea aerului în diferite zone:

- la nivel teoretic, prin știri și articole relevante
- la nivel practic, prin marcarea pe hartă a zonelor în funcție de risc și oferirea explicațiilor aferente

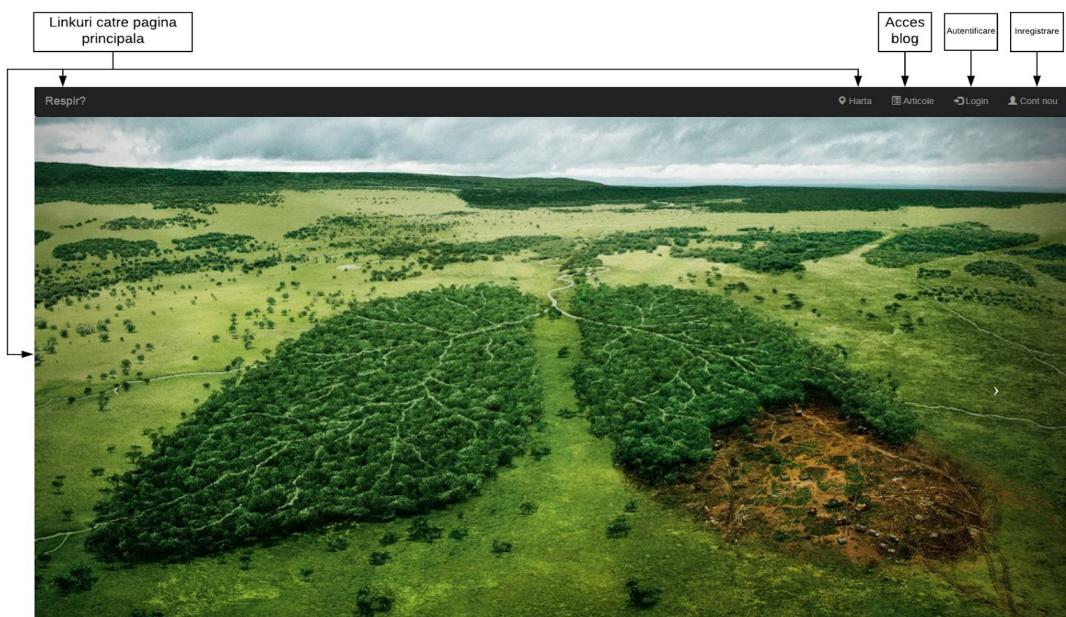
5.1 Administrare

5.1.1 Operații de bază

Accesarea aplicației

Aplicația poate fi accesată introducând adresa <http://respir.info>⁴² într-un browser compatibil⁴³.

Pagina initială include legături de acces pentru înregistrare și autentificare, precum și către pagina principală:

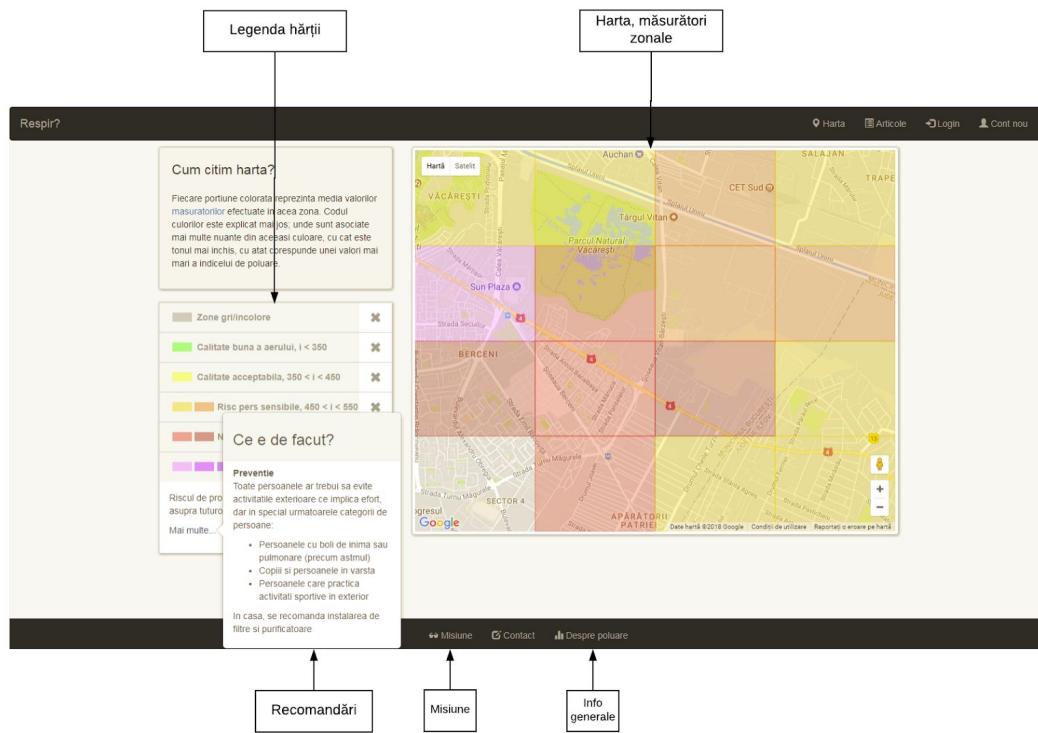


⁴² În momentul de față, deoarece aplicația este gazduită pe un server folosind un cont gratuit, domeniul este configurat să execute *frame redirect* către domeniul generat de serverul gazdă, i.e.

<https://enigmatic-ocean-67266.herokuapp.com/>

⁴³ Aplicația a fost testată folosind browser-ul Chrome versiunea 63

Pagina principală conține o hartă pe care sunt afișate zonele de interes, colorate transparent în funcție de media măsurătorilor efectuate în aria respectivă, precum și legenda culorilor și recomandări:



Înregistrare

Un utilizator se poate înregistra urmând următorii pași:

→ din meniul superior acceseză link-ul “Cont nou”:



→ introduce un nume de utilizator și o parolă:

The screenshot shows a registration form titled "Cont nou". It contains three input fields: "username", "parola", and "confirmare parola". Below the fields is a large blue button labeled "Inregistrare!".

◆ pentru a permite pasul următor, câmpurile sunt validate

The screenshot shows a registration form titled "Cont nou". The "username" field contains "Gigel". The "parola" field contains "....." and the "confirmare parola" field also contains ".....". A red error message "Parolele nu se potrivesc!" (Passwords do not match!) is displayed below the fields. Below the fields is a large blue button labeled "Inregistrare!".

→ dacă numele de utilizator nu este duplicat⁴⁴, se afișează un mesaj de succes și utilizatorul este autentificat automat:



Posibilitățile unui utilizator înregistrat sunt detaliate în capitolele următoare.

⁴⁴ numele de utilizatori sunt case sensitive

Autentificare

Un utilizator se poate autentifica urmând următorii pași:

- din meniul superior, accesează link-ul “Login”



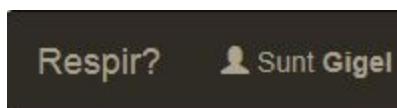
- introduce numele de utilizator și parola personală în câmpurile aferente:

The form has two input fields: one for 'username' containing 'Gigel' and one for 'parola' which is empty. Below the fields is a large blue button labeled 'Login!'. The entire form is set against a light gray background.

- ◆ pentru a permite pasul următor, câmpurile sunt validate

The form has two input fields: one for 'username' containing 'Gigel' and one for 'parola' which is empty. A red error message 'Parola nu e completata!' is displayed above the 'Login!' button. The 'Login!' button is blue. The entire form is set against a light gray background.

- dacă datele sunt valide, se afișează pagina principală:

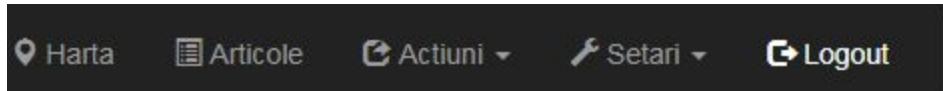


altfel, utilizatorul este redirectionat înapoi la pagina de autentificare

Un utilizator neautentificat poate folosi aplicația, însă nu o poate configura, adăuga măsurători sau articole.

Ieșire

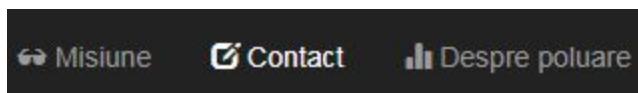
Un utilizator se poate deautentifica folosind link-ul "Logout" din meniu superior:



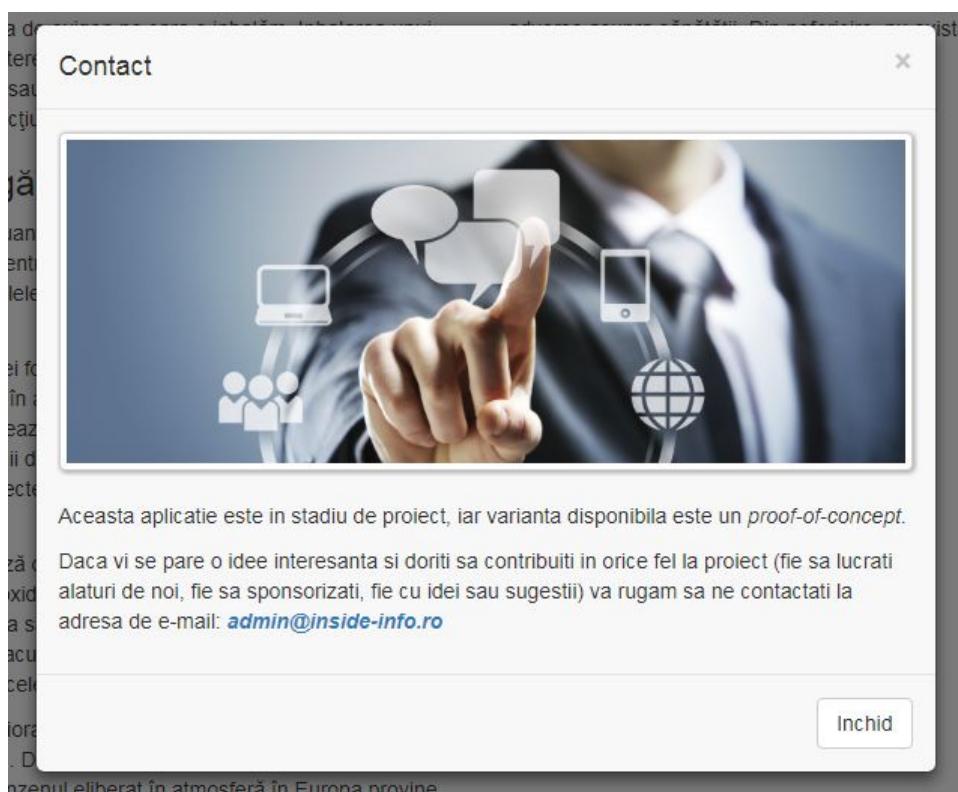
Se va afișa un mesaj de succes și utilizatorul va fi redirectionat la pagina principală.

Contact

Formularul de contact poate fi deschis accesând link-ul *Contact* din meniu inferior.



Utilizatorul poate contacta echipa proiectului via e-mail, accesând link-ul din formularul de contact.



Informații generale

Utilizatorul poate afla informații generale despre proiect accesând link-urile din meniul inferior “Misiune” și “Despre poluare”.

5.1.2 Utilizatori

Un utilizator înregistrat poate realiza, în plus față de unul neînregistrat, următoarele acțiuni:

- fără permisiuni specifice:
 - ◆ să își modifice datele personale
 - ◆ să își configureze preferințe în aplicație
- în funcție de permisiuni:
 - ◆ să adauge sau modifice articole (editor sau admin)
 - ◆ să adauge măsurători (contributor)
 - ◆ să modifice permisiunile altor utilizatori (admin)

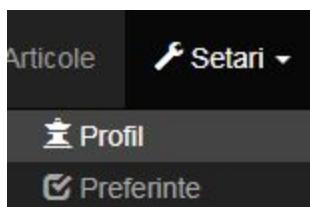
Modificarea datelor personale⁴⁵

Un utilizator înregistrat poate modifica următoarele date personale:

- prenume
- nume
- adresa e-mail
- telefon
- țara și orașul de reședință

Pentru a-și modifica datele personale, utilizatorul va trebui să se autentifice, apoi să aplice următorii pași:

- din meniul superior, accesează link-ul *Setări > Profil*



⁴⁵ În versiunea curentă, deși sunt salvate, datele nu sunt folosite în altă parte; vor fi necesare însă pentru dezvoltări ulterioare (trimitere notificări pe email, centrare pe orașul de reședință, adresare către utilizator, etc)

→ introduce sau actualizează datele⁴⁶ în formularul afișat

Imi actualizez datele

Vladimir

Ioana

vladimir.ioana@eservglobal.com

telefon

Romania

Popesti-Leordeni

Actualizez!

→ apasă butonul “Actualizez”

→ dacă datele sunt actualizate cu succes, se afișează un mesaj de succes, altfel unul de eroare; în ambele cazuri, utilizatorul este redirectionat către pagina principală

Configurarea preferințelor

Un utilizator înregistrat poate modifica următoarele preferințe:

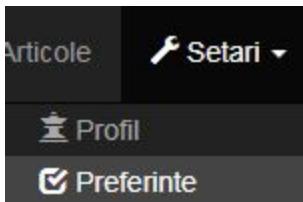
- “centrare pe orașul de reședință”: la prima încărcare a hărții, aceasta va afișa orașul de reședință configurat în datele personale ale utilizatorului⁴⁷
- “economie la transmisia de date”: va limita operațiunile efectuate, inclusiv granularitatea hărții (la 8x8), astfel încât răspunsul să fie mai rapid (recomandată în versiunea PoC)
- *linii x coloane*: determină granularitatea hărții, i.e. numărul de zone în care să fie împărțită pentru afișarea măsurătorilor medii

Pentru a-și modifica datele personale, utilizatorul va trebui să se autentifice, apoi să aplice următorii pași:

→ din meniul superior, accesează link-ul *Setări > Preferințe*:

⁴⁶ Niciun câmp nu este obligatoriu

⁴⁷ Funcționalitatea nu este încă implementată, deoarece momentan nu există măsurători efectuate decât în București și Popești-Leordeni



→ Își modifică preferințele:

Preferinte

Centrare pe orașul reședință

Economie la transmisia de date

12 x 12

Actualizez!

- apasă butonul "Actualizez"
- dacă datele sunt actualizate cu succes, se afișează un mesaj de succes, altfel unul de eroare; în ambele cazuri, utilizatorul este redirecționat către pagina principală

Administrarea permisiunilor

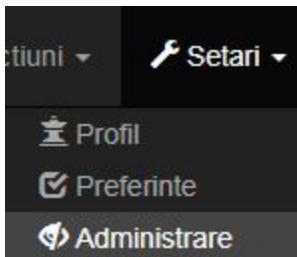
Un utilizator înregistrat, cu drepturi de administrare, poate modifica drepturile altor utilizatori:

- de a încărca măsurători
- de a adăuga articole
- de aprobare măsurători și articole⁴⁸
- de administrare

Pentru a modifica drepturile sale sau ale altor utilizatori, utilizatorul va trebui să se autentifice, apoi să aplice următorii pași:

- din meniul superior accesează link-ul *Setări > Administrare*

⁴⁸ această funcționalitate nu este implementată în PoC, fiind rezervată pentru dezvoltări ulterioare



- introduce un nume valid de utilizator și îl bifează drepturile de acces

Caut utilizator

Gigel

Drepturi incarcare masuratori
 Drepturi adaugare articole
 Drepturi aprobarare masuratori si articole
 Drepturi de administrare

Modific!

- apasă butonul "Modific"
- dacă utilizatorul nu a fost găsit, este afișat un mesaj de eroare și pagina este reîncărcată
- dacă utilizatorul a fost găsit, iar datele au fost actualizate cu succes este afișat un mesaj de succes; dacă a apărut o problemă la actualizarea datelor, este afișat un mesaj de eroare; în ambele cazuri, utilizatorul este redirecționat către pagina principală

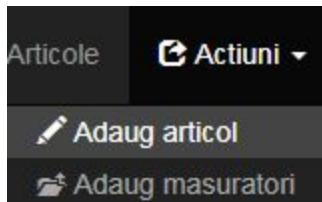
5.2 Continut

Adăugarea articolelor

Un utilizator înregistrat, cu drepturi de editor, poate încărca articole. Articolele vor fi afișate pe blogul aplicației.

Pentru a adăuga un articol, utilizatorul va trebui să se autentifice, apoi să aplice următorii pași:

- din meniul superior accesează link-ul *Acețiuni > Adaug articol*



- adaugă conținutul articolului în formularul afișat; pentru a fi afișat într-un format prietenos, conținutul trebuie adăugat în format HTML (un editor HTML va fi adăugat într-o iterare ulterioară)

Adaug articol

Topul oraselor cu cel mai poluat aer. Unde se situează Bucurestiul

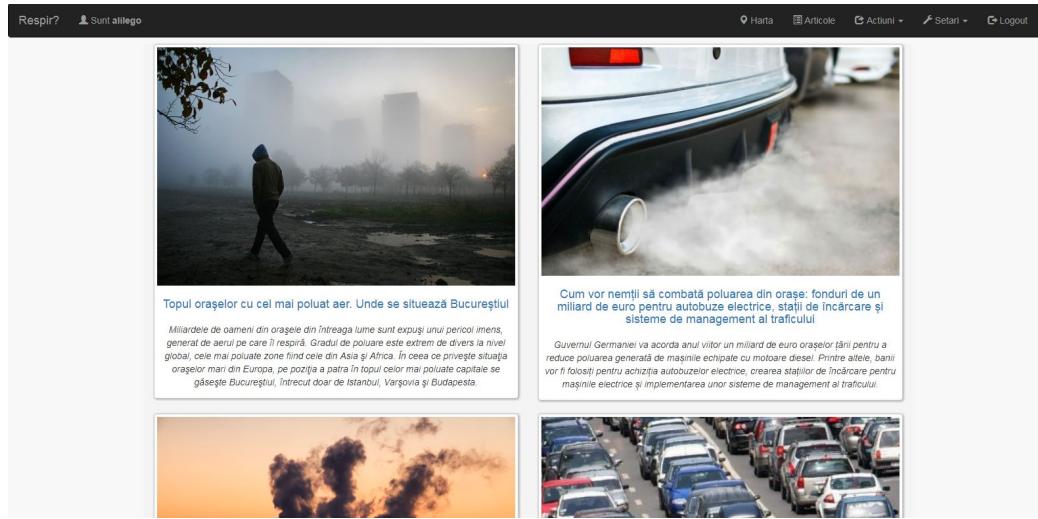
adresa imaginii principale

Miliardele de oameni din orașele din întreaga lume sunt expoși unui per

<p>Cele mai recente date ale Organizatiei Mondiale a Sanatatii (OMS) referitoare la concentratia de particule ultra-fine din aer au fost facute publice. Aceste date se refera la particule mai mici de 2,5 microni si au fost colectate din fiecare regiune a lumii. </p><p>Chiar si dupa implementarea masurilor pentru combaterea poluarii, aerul din Paris masoara un nivel de poluare a aerului dublu fata de limita recomandata de OMS, adica 18 micrograme pe metru cub, fata de 10 micrograme pe metru cub. Totusi, Parisul se inscrie in topul celor mai putin poluate orase. </p><p></p><p>La polul opus, cel mai mare pericol îl prezinta orasul iranian Zabol, cu nu mai putin de 217 micrograme pe metru cub. Topul celor mai poluate orase din lume este condus de state din India, China si Iran. </p><p>"></p>

Salvez!

- apasă butonul “Salvez”
- când articolul este salvat cu succes, utilizatorul este redirecționat la lista de articole (blog-ul aplicației)

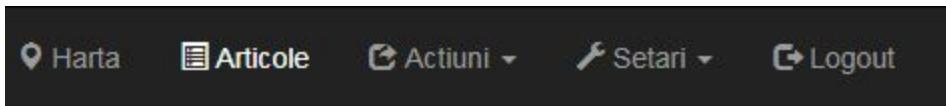


Modificarea unui articol

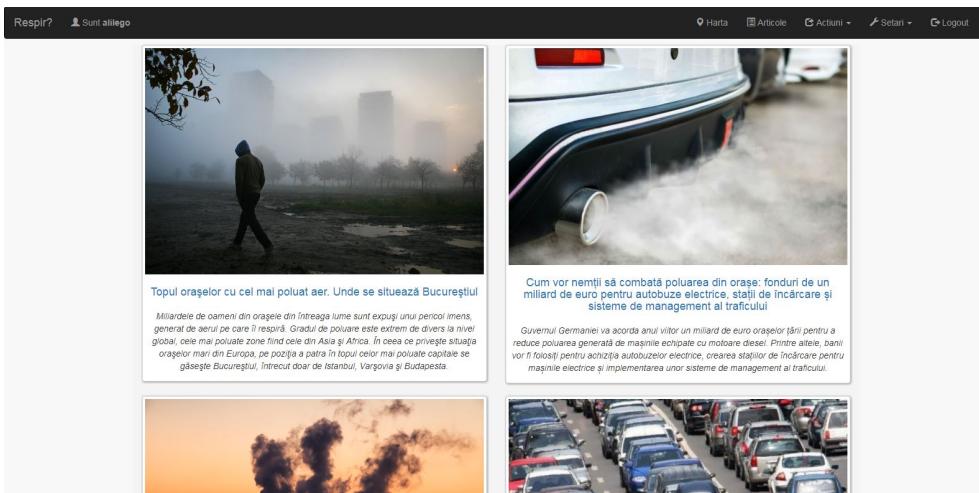
Un utilizator înregistrat poate modifica articolele al căror autor este.

Pentru a modifica un articol, utilizatorul va trebui să se autentifice, apoi să aplică următorii pași:

- din meniul superior accesează link-ul “Articole”



- din lista de articole afișată apasă pe titlul sau imaginea unui articol al cărui autor este



- în pagina articolului apasă pe butonul “Editez”

Respir?  Sunt aliego

Topul orașelor cu cel mai poluat aer. Unde se situează Bucureștiul

Miliardele de oameni din orașele din întreaga lume sunt expuși unui pericol imens, generat de aerul pe care îl respiră. Gradul de poluare este extrem la nivel global, cele mai poluate zone fiind cele din Asia și Africa. În ceea ce privește situația orașelor mari din Europa, pe pozitia a patra în topul celor mai poluate capitale se găsește Bucureștiul. Înrevă doar de Istanbul, Varsavia și Budapesta.

Autor: aliego



Cele mai recente date ale Organizației Mondiale a Sănătății (OMS) referitoare la concentrația de particule ultra-fine din aer au fost facute publice. Aceste date se referă la particule mai mici de 2,5 micrometri și au fost colectate din fiecare regiune a lumii.

Chiar și după implementarea masurilor pentru combaterea poluării, aerul din Paris masoara un nivel de poluare a aerului dublu fata de limita recomandată de OMS, adică 18 micrograme pe metru cub, fata de 10 micrograme pe metru cub. Totuși, Parisul se înscrie în topul celor mai puțin poluate orașe.

La polul opus, cel mai mare pericol îl prezintă orașul iranian Zabol, cu nu mai puțin de 217 micrograme pe metru cub. Topul celor mai poluate orașe din lume este condus de state din India, China și Iran.

	217	176	170	200
Zabol Iran	217			
Gwalior India	176			
Allahabad India	170			
Riyadh Saudi Arabia	156			
Al Jubail Saudi Arabia	152			
Patna India	149			
Rajpur India	144			

→ În pagina de editare, actualizează datele articoului și apasă pe butonul "Salvez"

Edit

Topul orașelor cu cel mai poluat aer. Unde se situează Bucureștiul

<http://storage0.dms.mpinteractiv.ro/media/1/186/3929/16160514/1/3949>

Miliardele de oameni din orașele din întreaga lume sunt expuși unui pericol imens, generat de aerul pe care îl respiră. Chiar și după implementarea masurilor pentru combaterea poluării, aerul din Paris masoara un nivel de poluare a aerului dublu fata de limita recomandată de OMS, adică 18 micrograme pe metru cub, fata de 10 micrograme pe metru cub. Totuși, Parisul se înscrie în topul celor mai puțin poluate orașe. La polul opus, cel mai mare pericol îl prezintă orașul iranian Zabol, cu nu mai puțin de 217 micrograme pe metru cub. Topul celor mai poluate orașe din lume este condus de state din India, China și Iran.

<p>Cele mai recente date ale Organizației Mondiale a Sănătății (OMS) referitoare la concentrația de particule ultra-fine din aer au fost facute publice. Aceste date se referă la particule mai mici de 2,5 micrometri și au fost colectate din fiecare regiune a lumii. </p>
<p>Chiar și după implementarea masurilor pentru combaterea poluării, aerul din Paris masoara un nivel de poluare a aerului dublu fata de limita recomandată de OMS, adică 18 micrograme pe metru cub, fata de 10 micrograme pe metru cub. Totuși, Parisul se înscrie în topul celor mai puțin poluate orașe. </p>
<p>La polul opus, cel mai mare pericol îl prezintă orașul iranian Zabol, cu nu mai puțin de 217 micrograme pe metru cub. Topul celor mai poluate orașe din lume este condus de state din India, China și Iran. </p>


Salvez!

Renunt

→ datele articoului sunt actualizate, iar utilizatorul este redirecționat către pagina de vizualizare articol

Adăugarea măsurătorilor

Un utilizator înregistrat, cu drepturi de contributor, poate încărca măsurători. Toate măsurătorile validate vor fi luate în considerare pentru afișarea datelor statistice.

Măsurătorile sunt încărcate ca fișiere text cu următorul format:

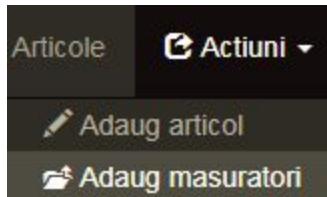
- HEADER: separat de restul fisierului prin 2 linii formate din cratima ‘-’
 - conține formatul în care sunt introduse măsurătorile (fiecare linie trebuie să aiba același format) și poate conține orice câmpuri separate prin *pipe* (‘|’), în orice ordine, dar vor fi luate în considerare doar următoarele:
 - LAT
 - LONG
 - DateTime
 - MQ4
 - MQ9
 - MQ135
 - CO2_ppm
- CONȚINUT: câte o măsurătoare pe fiecare linie, după formatul specificat în header

Un exemplu de fișier valid este următorul:

S	HDP	LAT	LONG	DateTime		Age	MQ4	MQ9	MQ135	CO2_ppm
8	99	44.392818	26.122447	11/26/2017 12:59:11	855	100	247	60	534.45	
8	99	44.392921	26.122352	11/26/2017 12:59:13	857	100	249	61	559.78	
8	99	44.393016	26.122266	11/26/2017 12:59:15	931	102	258	67	728.24	
8	99	44.393108	26.122175	11/26/2017 12:59:18	113	104	262	68	759.15	
8	99	44.393112	26.122129	11/26/2017 12:59:20	186	104	264	68	759.15	
7	126	44.393104	26.122112	11/26/2017 12:59:22	145	105	267	69	790.90	
9	109	44.393077	26.122093	11/26/2017 12:59:24	197	106	269	70	823.50	

Pentru a adăuga un fișier cu măsurători, utilizatorul va trebui să se autentifice, apoi să aplice următorii pași:

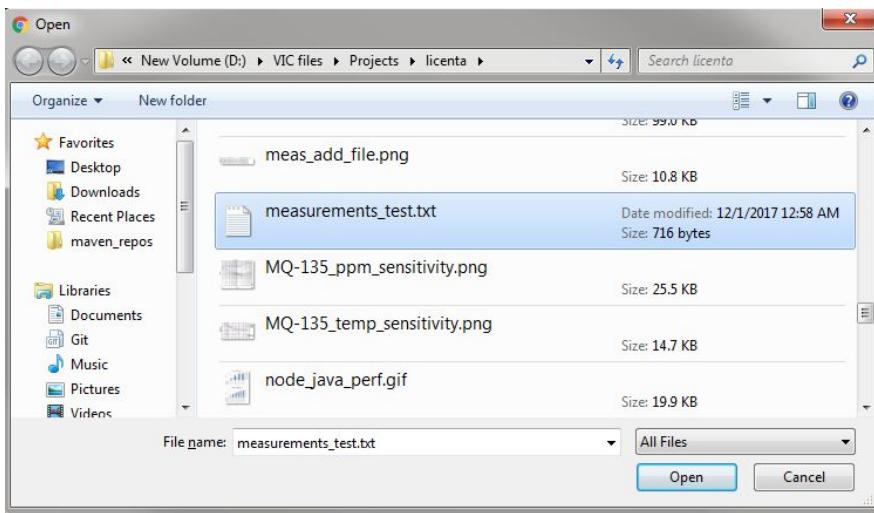
→ din meniul superior accesează link-ul *Acțiuni > Adaug măsurători*



→ apasă butonul “Încarc fișier”



→ caută și selectează fișierul, apoi apasă “Open”



→ dacă fișierul este încărcat cu succes se afișează un mesaj de succes



Fișierul este validat pe server. În caz că formatul este invalid, utilizatorul nu este notificat.

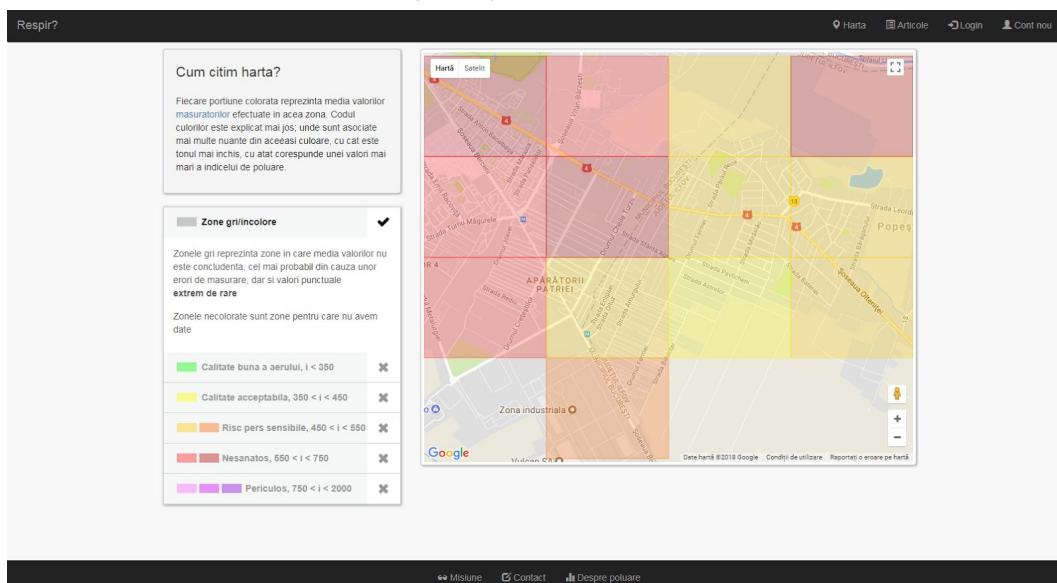
De asemenea, pentru a permite ca mai mulți utilizatori să-și încarce măsurătorile, presupunând că 2 măsurători în condiții similare pot fi identice, nu se elimină duplicatele.

Afișarea hărții

Orice utilizator poate accesa harta, unde sunt indicate date cu privire la poluarea zonală.

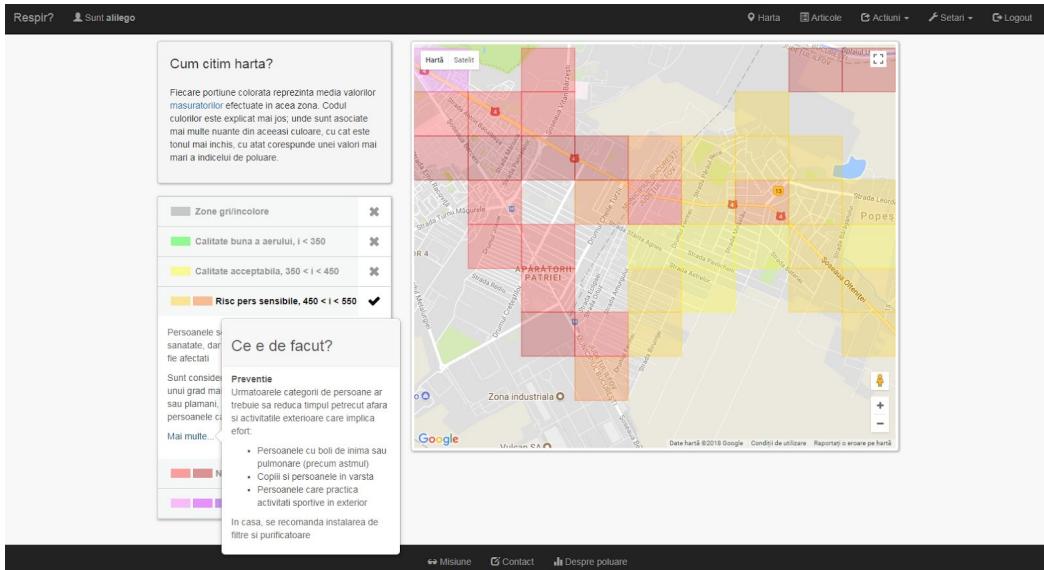
Pentru a accesa harta, un utilizator apasă pe link-ul “Harta” din meniul superior. Va fi afișată pagina principală, ce conține:

- harta, cu zone hașurate în diferite culori în funcție de media măsurătorilor înregistrate în zonele respective
- explicații privind ceea ce reprezintă zonele de pe hartă
- legenda culorilor, cu informații adiționale și recomandări



Pentru un utilizator neînregistrat, harta va avea o granularitate de 4x4. Utilizatorii înregistrați își vor putea defini orice valori ale granularității⁴⁹:

⁴⁹ Totuși, cu cât este granularitatea mai mare cu atât harta va fi actualizată mai încet.

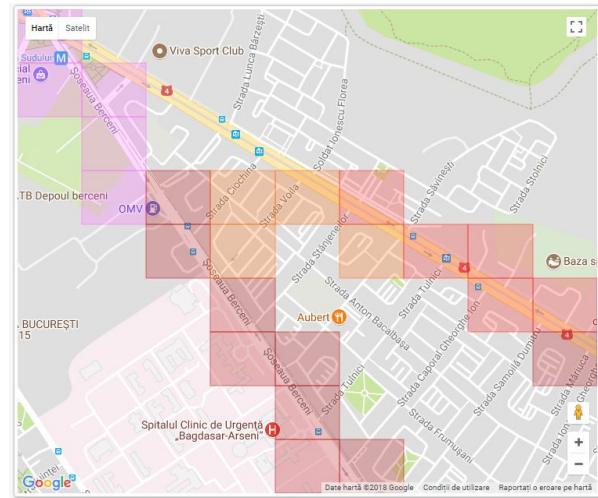
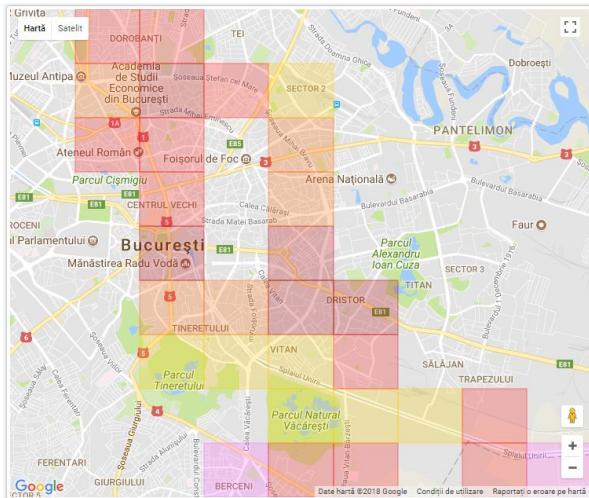


Utilizatorii pot schimba zona de vizualizare prin interacțiuni cu harta:

- micșorare/magnificare
- tragere

Astfel, se pot identifica:

- valori medii într-o zonă mai largă, realizând operații de micșorare:
- masurători precise, realizând operații de magnificare:



Blog

Orice utilizator poate accesa portalul de articole și știri, pentru informații din domeniu.

Pentru a accesa blog-ul aplicației, un utilizator va urma pașii de mai jos:

- pentru a vizualiza lista de articole, va apăsa butonul “Articole” din meniul superior al aplicației:

The screenshot shows a website with a dark header containing the logo "Respir?" and navigation links for "Harta", "Articole", "Login", and "Cont nou". Below the header is a grid of four images. The top-left image shows a person walking in a field covered in fog. The top-right image shows the rear of a car with visible exhaust fumes. The bottom-left image shows a dramatic sunset or sunrise with large, billowing smoke or steam plumes. The bottom-right image shows a dense traffic jam of many cars on a road. At the bottom of the page, there is a footer with links for "Misiune", "Contact", and "Despre poluare".

- pentru a citi un articol, va apăsa pe imaginea sau titlul acestuia, din lista afișată; se deschide o pagină nouă cu detaliile articoului:

The screenshot shows a detailed view of an article. The title is "Topul orașelor cu cel mai poluat aer. Unde se situează Bucureștiul". Below the title is a small image of a person walking in a foggy field. The text discusses air pollution levels and includes a quote from the World Health Organization (OMS) about particulate matter. It also mentions Paris as an example of a city taking measures against air pollution. At the bottom of the article section is a horizontal bar chart comparing PM2.5 levels across various cities. The chart has a scale from 0 to 200. The data is as follows:

Oraș	PM2.5 (µg/m³)
Zabol Iran	217
Gawalior India	176
Allahabad India	170
Riyadh Saudi Arabia	156
Al Jabbah Saudi Arabia	152
Patna India	149
Rajpur India	144
Bamenda Cameroun	132
Xingtai China	128
Haodong China	126

6 Concluzii

6.1 Un punct de vedere asupra tehnologiilor utilizate

Pot confirma că tehnologiile folosite sunt potrivite pentru dezvoltări rapide, tip PoC sau MVP:

- cu minime cunoștințe de electronică și cunoștințe de bază de programare în C am proiectat un prototip de dispozitiv IoT, capabil să înregistreze măsurători din mediul exterior, cu următoarele minusuri:
 - lipsa cunoștințelor de electronică a determinat o proiectare deficitară (în sensul că nu am reușit să folosesc interfața UEXT a plăcuței, iar funcționarea pe baterie nu este aplicabilă din cauza consumului mare al senzorilor)
 - funcționalitatea acestuia este mai mult demonstrativă, deoarece nici costurile, nici performanța dispozitivului nu sunt competitive; în consecință, pentru lansarea pe piață trebuie reproiectat un dispozitiv competitiv
- folosind doar cunoștințe JavaScript și Node.js, am proiectat rapid o aplicație full-stack JavaScript: prima schiță a aplicației web a fost gata în câteva ore, apoi am putut să adaug funcționalitate iterativ fără să-mi fac griji pentru schema bazei de date sau *framework*, cu funcționalitatea dispusă în următoarele iterații:
 - index
 - utilizatori
 - măsurători
 - hartă
 - stilizare
 - articole

Cantitatea mare de informații disponibile pe Internet m-a ferit de orice blocaj - am găsit soluții relativ rapid pentru orice problemă întâmpinată.

6.2 Concluzii la nivel practic

Având în vedere scopul de facto al lucrării, analizând datele obținute reies câteva concluzii:

- stațiile fixe folosite în acest moment de ANPM sunt irelevante pentru interesul cetățenilor cu privire la calitatea aerului, deoarece din datele *respir.info* se poate observa variația consistentă a calității aerului pe arii restrânse (ANPM detine

doar 5 stații fixe în București, iar din pozele de pe site pare că sunt așezate în spații verzi)

- rezultă că o analiză statistică cu suficiente date (cu un număr suficient de dispozitive, mai multe măsurători efectuate în aceleași locuri la momente diferite) ar conduce la o definire precisă a problemelor, pas esențial în identificarea corectă a soluțiilor
- facând o analiză comparativă cu noul site ANPM (bazat însă tot pe câteva stații fixe), calitateaer.ro:
 - aplicația *respir.info* oferă avantajul informării cetățenilor pe baza datelor afișate (momentan, afișează potențialele riscuri, categoriile de persoane impactate și recomandări pentru acestea)
 - valorile medii diferă (posibil că algoritmul folosit în aplicație pentru calcularea indicelui trebuie îmbunătățit)

6.3 Dezvoltări ulterioare ale aplicației

Pentru acoperirea unei suprafețe vaste și măsurări frecvente cu constituirea unei statistici există 3 opțiuni:

1. Atragerea unor investitori/suștinători. Platforma *Patreon* este o variantă. Ministerul public ar fi varianta ideală
2. Orientarea aplicației web către realizarea de profit (fie din drepturi publicitare, fie prin implementarea SaaS⁵⁰)
3. Implicarea utilizatorilor

Oricare dintre aceste opțiuni ar fi aleasă, prototipul va trebui dezvoltat pentru a avea autonomie și a rezista la condiții meteorologice proaste (umiditate, frig). De asemenea, va trebui calibrat pentru măsurători absolute mai precise.

Pentru primele 2 opțiuni, dezvoltarea fiind internă (în organizație), abordarea este mai ușoară.

Opțiunea a 3-a prezintă mai multe condiții de satisfăcut:

- Pentru a fi achiziționat, dispozitivul trebuie să ofere și altă funcționalitate de interes general
- Dispozitivul trebuie să fie relativ ieftin
- Conectarea și transmiterea datelor trebuie să fie facilă
- Utilizatorii trebuie motivați să întreprindă această acțiune

⁵⁰ Software as a Service

O variantă relativ simplă este un accesoriu auto pentru măsurarea calității aerului (există o asemenea funcționalitate integrată deja în unele mașini Renault). Datele pot fi transmise automat via wi-fi pe *smartphone* și încărcate în aplicația dedicată, iar folosirea *smartphone-ului* pentru identificarea locației scutește costuri pentru modulul de măsurare.

Bibliografie

1. Boicea, Alexandru, Radulescu, F. & Agapin, L. I. - *MongoDB vs Oracle*
2. Buckler, Craig - *SQL vs NoSQL: The Differences*
3. Correa, Gustavo - *Node.js JavaScript Server Side*
4. Doberstein, Dan - *Fundamentals of GPS Receivers. A Hardware Approach*
5. Harell, Jeff - *Node.js at PayPal*
6. Hart, Mikal - *TinyGPS Library*
7. Iorga, Gabriela - *Air Pollution Monitoring: A Case Study from Romania*
8. Miller, T. A., Bakrania, S. D., Perez, C. & Wooldridge, M. S. - *Nanostructured Tin Dioxide Materials for Gas Sensor Applications*
9. Mualem, Melkar - *Investigation into Full-Stack JavaScript as a web server*

Resurse online

1. www.altexsoft.com/blog - *The Good and the Bad of JavaScript Full Stack Development*
2. www.arduino.cc - *What is Arduino?*
3. davidegironi.blogspot.ro - *Cheap CO₂ meter using the MQ135 sensor with AVR ATmega*
4. learn.sparkfun.com - *GPS Basics*
5. www.mongodb.com - *NoSQL Explained*

ANEXE

1 Dicționar de termeni și acronime

Pe parcursul lucrării sunt folosiți uneori termeni și acronime care nu există în limba română. Mi-am permis totuși să îi folosesc deoarece au fost consacrați în engleză, aşa că în traducere ar fi fost dificil de recunoscut pentru cititor. Tabelul de mai jos conține scurte definiții și explicații pentru unii dintre aceștia. Pentru ceilalți am considerat că sensul este evident.

PoC	<i>Proof of Concept</i> - proiect pilot, care demonstrează în general fezabilitatea unui proiect
MVP	<i>Minimum Viable Product</i> - proiect pilot, cu un număr minim de cerințe, care poate fi lansat în producție
GPS	<i>Global Positioning System</i> , un sistem global de navigație prin satelit, administrat de SUA, care oferă informații despre timp și locația geografică unui receptor GPS oriunde pe sau aproape de Pământ unde există vedere la 4 sau mai mulți sateliți GPS. Transmisia se realizează via unde radio.
NMEA	<i>National Marine Electronics Association</i> . Au definit și reglementat <i>NMEA 0183</i> , un standard ce definește un protocol de comunicație între diverse electronice folosite în marină (sonar, pilot automat, etc) printre care și receptoare GPS
WEB	O rețea/un spațiu informațional unde diferitele resurse (documente, imagini, etc) sunt identificate de URL-uri (<i>Uniform Resource Locators</i>), interconectate prin <i>link-uri</i> și pot fi accesate via INTERNET; deseori folosit în contextul "pagină web"
AQI	<i>Air Quality Index</i>
ANPM	Agenția Națională pentru Protecția Mediului
SD [card]	<i>Secure Digital</i> - un tip standard de card de memorie non-volatile folosit pentru stocarea datelor în dispozitive portabile

IoT	<i>Internet of Things</i> - rețeaua de obiecte, vehicule, locuințe, și.a. în care au fost integrate dispozitive electronice inteligente, senzori, elemente de comandă și elemente de conectare la INTERNET, astfel încât obiectele sunt interconectate și pot schimba date între ele
ppm	<i>Part-Per-Milion</i> . Unitate de măsură folosită pentru concentrația gazelor
baud	Unitate de măsură pentru viteza de modulație în transmisia datelor printr-un canal de comunicație. Frecvența baud reprezintă numărul de schimbări ale semnalului (fie el voltaj, frecvență, fază) pe secundă. În cazul transmisiei de date binare, 1 baud = 1 bit. De pildă în acest proiect am configurat viteze de transmisie de 9600 baud = 9600 bits/secundă
VCS	<i>Version Control System</i> , un sistem de gestiune al diferitelor versiuni ale unui același produs; exemple: CVS, SVN, Git
npm	<i>node package manager</i> , manager de pachete și cel mai mare registru de software din lume; folosit pentru încărcarea și integrarea bibliotecilor și <i>framework-urilor</i> JavaScript
JSON	<i>JavaScript Object Notation</i> , un format pentru transmisia datelor bazat pe o colecție de perechi nume - valoare, și o listă ordonată de valori. Ușor de parsat și generat de către computer, ușor de citit de către oameni, fără surplus mare de date.
REST	<i>REpresentational State Transfer</i> . Contextual, un model de implementare a serviciilor web folosind un set uniform predefinit de operații.
HTTP	<i>HyperText Transfer Protocol</i> . Un protocol de comunicație de nivel aplicație pentru transferul de informație între sisteme. Stă la baza <i>World Wide Web</i> . Conceptul de <i>hypertext</i> se referă la un text structurat ce conține legături logice (<i>hyperlinks</i>) între noduri ce conțin text. HTTP este protocolul prin care se interschimbă/transferă <i>hypertext</i> .
CRUD	<i>Create, Read, Update, Delete</i> . Cele 4 operații de bază pentru stocarea informației. Când e folosit în relație cu interfața cu

	utilizatorul, se referă la adăugarea, căutarea, vizualizarea și actualizarea informațiilor.
ORM	<i>Object Relational Mapping</i> . O tehnică de programare pentru convertirea datelor între sisteme cu tipuri de date diferite folosind limbaje orientate pe obiecte.
MVC	<i>Model-View-Controller</i> . Un model arhitectural folosit în general la aplicațiile web ce separă aplicația în cele 3 componente logice: <ul style="list-style-type: none"> - <i>model</i> = totalitatea datelor transferate între <i>view</i> și <i>controller</i> - <i>view</i> = implementarea interfeței cu utilizatorul - <i>controller</i> = interfața între <i>view</i> și <i>model</i> ce include toată partea de manipulare și procesare a datelor

2 Agregarea recursivă a măsurătorilor în Node.js

```
var getMeasurements = function(gmap, callback){
    Measurement.aggregate([
        {$match: { long: { $gte: gmap.lngMin, $lte: gmap.lngMax },
                  lat: { $gte: gmap.latMin, $lte: gmap.latMax }}},
        {$group: { _id: "$__v",
                   averageMq4: { $avg: "$mq4" },
                   averageMq9: { $avg: "$mq9" },
                   averageMq135: { $avg: "$mq135" },
                   averageCo2ppm: { $avg: "$co2ppm" }}})
    ], function (err, measurement) {
        if(err){
            console.log(err);
        } else {
            callback(measurement);
        }
    });
}

getMeasurements(req.session.gmap, function retrieveMeasurement(measurement){
    //compute lat&long for storage
    var lngMin = req.session.gmap.boundaries.lngMin + (req.session.gmap.b *
    req.session.gmap.lngDelta);
    var lngMax = lngMin + req.session.gmap.lngDelta;
    var latMin = req.session.gmap.boundaries.latMin + (req.session.gmap.a *
    req.session.gmap.latDelta);
    var latMax = latMin + req.session.gmap.latDelta;

    if(measurement.length > 0){
        req.session.gmap.zones.push({
            latMin: latMin,
            latMax: latMax,
            lngMin: lngMin,
            lngMax: lngMax,
            mq4: measurement[0].averageMq4,
            mq9: measurement[0].averageMq9,
            mq135: measurement[0].averageMq135,
            co2ppm: measurement[0].averageCo2ppm,
        });
        //console.log(gmap);
    }
    //compute lat&long for next calculus
    if(req.session.gmap.a < (req.session.gmap.maxA - 1) && req.session.gmap.b <
    (req.session.gmap.maxB - 1)){ req.session.gmap.b++; }
    else if(req.session.gmap.a < (req.session.gmap.maxA - 1) && req.session.gmap.b ==
    (req.session.gmap.maxB - 1)){ req.session.gmap.a++; req.session.gmap.b = 0; }
    else if(req.session.gmap.a == (req.session.gmap.maxA - 1) && req.session.gmap.b <
    (req.session.gmap.maxB - 1)){ req.session.gmap.b++; }
}
```

```
        else {
            renderMain(req, res);
            return;
        }
        lngMin = req.session.gmap.boundaries.lngMin + (req.session.gmap.b * req.session.gmap.lngDelta);
        lngMax = lngMin + req.session.gmap.lngDelta;
        latMin = req.session.gmap.boundaries.latMin + (req.session.gmap.a * req.session.gmap.latDelta);
        latMax = latMin + req.session.gmap.latDelta;

        Measurement.aggregate([
            {$match: { long: { $gte: lngMin, $lte: lngMax },
                      lat: { $gte: latMin, $lte: latMax }}},
            {$group: { _id: "$__v",
                      averageMq4: { $avg: "$mq4" },
                      averageMq9: { $avg: "$mq9" },
                      averageMq135: { $avg: "$mq135" },
                      averageCo2ppm: { $avg: "$co2ppm" }}}
        ], function (err, measurement) {
            if(err){
                console.log(err);
            } else {
                retrieveMeasurement(measurement);
            }
        });
    });
}
```