

UNIVERSITATEA POLITEHNICA BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL AUTOMATICĂ ȘI INFORMATICĂ
INDUSTRIALĂ



PROIECT DE DIPLOMĂ

Implementarea unui sistem în vederea monitorizării traficului
feroviar

Stejar Nicolai

Coordonator științific:

Conf. dr. ing. Sacală Ioan

BUCUREȘTI

2018

UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
AUTOMATION AND APPLIED INFORMATICS DEPARTMENT



DIPLOMA PROJECT

Implementing a rail-traffic monitoring system

Stejar Nicolai

Thesis advisor:

Conf. dr. ing. Ioan Sacală

BUCHAREST

2018

CUPRINS

2018	2
CUPRINS	1
Sinopsis	2
Abstract	2
Mulțumiri	3
1 Introducere	4
1.1 Context	4
1.2 Problema	5
1.3 Obiective	5
1.4 Structura lucrării.....	6
2 Studiu de piață / Metode existente.....	7
3 Soluția propusă	12
3.1 Arhitectura soluției.....	12
3.2 Implementare hardware / software	17
3.3 Fluxuri aplicație	30
3.4 Studiu de caz – configurare și monitorizare tren.....	37
3.5 Tehnologii folosite.....	40
4 Evaluare	47
5 Concluzii.....	49
6 Dezvoltări viitoare	50
7 Bibliografie.....	52
8 Anexe	54

SINOPSIS

Transportul feroviar este vital pentru infrastructura oricărei țări, iar managementul unui astfel de sistem este esențial pentru deplasarea la timp și în siguranță a călătorilor sau a mărfurilor.

Rețeaua română numără aproximativ 990 de stații și peste 10.000 de km de cale ferată pe care circulă zilnic 1200 de trenuri, operate de 10 companii de transport feroviar. Astfel se justifică necesitatea administrării și monitorizării digitale a infrastructurii feroviare și urmărirea și înregistrarea în timp real a materialului rulant care circulă în țară.

În prezenta lucrare am implementat o soluție completă – atât la nivel de infrastructură națională, cât și la nivel de operator de transport feroviar – care asigură integrarea managementului entităților feroviare cu o rețea de senzori IoT.

ABSTRACT

Railroad transportation is vital for every country's infrastructure, and the management for this kind of system is essential for the safe and in-time movement of people and goods.

Romanian rail network has approximately 990 stations and over 10.000 km of tracks that hold 1200 daily trains operated by 10 different companies. Thus, the necessity of digital administrating and monitoring of the infrastructure and the real-time tracking and recording of the rolling stock is obvious.

I have developed a complete solution – for both the national infrastructure and the operating companies – that combines the management of the rail entities with a network of IoT sensors.

MULȚUMIRI

Parcursul pentru a ajunge în poziția de a redacta această lucrare a fost suficient de solicitant cât să fiu dator cu câteva mulțumiri. Am un merit deosebit angajații de la CFR Călători care au modelat un hobby într-un rezultat academic, colectivul Essensys Software pentru experiența profesională și competențele tehnice și, în special, domnul profesor Ioan Sacală pentru răbdarea și încrederea acordată dezvoltării unei astfel de soluții.

1 INTRODUCERE

1.1 Context

Prima atestare documentară a unui transport pe șine datează din 1515 – un furnicular utilizat pentru accesul la castelul Hohensalzburg din Austria¹. Doua secole mai târziu, în 1748, James Watt brevetează locomotiva² cu aburi și astfel dă startul unei revoluții în materie de transport.

În prezent, 149 de țări au o rețea feroviară utilizabilă, iar statele dezvoltate investesc masiv în acest domeniu construind căi de mare viteză sau oferind facilități moderne călătorilor. Printre avantajele utilizării acestui mijloc de transport se numără:

- costurile scăzute (în special pentru transportul de marfă)
- grad mic de poluare
- timpii de parcurs care concurează chiar cu cel al avioanelor pe rute domestice
- confortul

Deși Romania nu se află în topurile ce vizează sectorul feroviar, acest tip de transport este foarte utilizat și vital pentru multe domenii de activitate. Uzinele Dacia, de exemplu, își transportă 50% din autovehiculele produse și 80% din piese și materie primă cu trenul. În prezent se modernizează Coridorul IV Paneuropean între Nădlac – Brașov – București – Constanța, fiind proiectat pentru o viteză comercială de 160km/h. După decizia de liberarizare a pieței de transport feroviar au apărut numeroase companii de transport de călători sau de marfă, iar unele sectii de circulație închise au fost redeschise și modernizate de aceste companii.

¹ <https://www.austria.info/us/austria/1000-years-of-imperial-history/hohensalzburg-fortress>

² Dickinson, Henry Winram (1939). *A Short History of the Steam Engine*. Cambridge University Press. p. 87. [ISBN 978-1-108-01228-7](#).

1.2 Problema

Întârzierile frecvente și defecțiunile apărute constant (vizibile și mediatizate cel mai des în cazul trenurilor de pasageri) fac transportul feroviar în România să nu fie eficient și nici atractiv pentru potențialii clienți care se reorientează către alte mijloace. În cazul transportului de marfă se poate ajunge la situații când, pentru a traversa țara de la Arad la Portul Agigea, un transport durează și 2-3 zile.

Soluția pe care o propun se axează pe probleme care țin de materialul rulant utilizat la tracțiunea trenurilor: locomotivele (indiferent de vechimea lor) se pot defecta în exploatare, mecanicii de tren nu respectă întotdeauna indicațiile din cale (semnalele, semafoarele, indicatoarele) și obligațiile semnalizării, de multe ori locomotivele utilizate sunt supra-dimensionate pentru trenurile remorcate și total ineficiente. În acest caz, dacă unii parametri fizici ai locomotivelor ar fi monitorizați, ar putea indica viitoare probleme.

Problema pentru care am dezvoltat o soluție în cele ce urmează pornește de la lipsa de monitorizare în timp real a traficului feroviar, și, în special, a echipamentelor de tracțiune (locomotivele).

1.3 Obiective

Obiectivul principal propus este realizarea unei platforme prin care fiecare operator de transport feroviar înregistrat să își poată monitoriza în timp real locomotivele proprii, putând să urmărească o serie de parametri relevanți pentru tipul de material rulant pe care îl deține și pentru propriile interese de eficientizare a transportului. De exemplu, un operator de transport care deține un parc de locomotive electrice este interesat să vadă în timp real: poziția fiecăreia, viteza de deplasare, tensiunea electrică din linia de contact, curenții pe motoarele de tracțiune, presiunile în instalația de frână a trenului și respectarea indicațiilor de semnalizare de către mecanicul de serviciu.

Un obiectiv secundar îl reprezintă extinderea soluției astfel încât mai mulți operatori de transport feroviar să poată accesa individual platforma menționată și să existe coerență între datele comune ale acestora.

În final, este în interesul călătorilor să aibă informații în timp real asupra trenului cu care călătoresc sau pe care îl așteaptă – întârzieri, opriri programate etc.

Soluția trebuie să fie scalabilă și extensibilă. Se pot înregistra mai mulți operatori, fiecare cu propriul material rulant și propriile necesități tehnice. De asemenea, senzorii trebuie să poată fi configurați, iar toată soluția trebuie să reziste încărcării la care este supus un sistem IoT (zeci – sute de mii de mesaje pe zi).

1.4 Structura lucrării

Capitolele următoare tratează abordări existente ale problemelor remarcate în domeniul feroviar, a soluțiilor propuse pentru rezolvarea acestora și o modalitate de implementare tehnică.

Capitolul 2 urmărește să prezinte stadiul actual al domeniului, prin expunerea principalelor aplicații software existente în România: software de comandă și monitorizare pentru locomotive și software pentru managementul materialului rulant.

Capitolul 3 vine cu o abordare nouă, prezentarea arhitecturii sistemului dezvoltat și detalii legate de implementarea acestuia. Sunt detaliate componentele principale: software-ul de management și rețeaua de senzori IoT pentru monitorizarea în timp real și modul cum sunt integrate într-o manieră scalabilă și extensibilă. Limbajul folosit este unul specific domeniului tehnic, cu multe tehnologii prezentate.

Capitolele 4, 5 și 6 aduc evaluări calitative și cantitative pentru componentele web și IoT, concluziile lucrării și direcțiile propuse pentru viitor.

Ultima parte prezintă bibliografia citată.

2 STUDIU DE PIAȚĂ / METODE EXISTENTE

Momentan, pe piața din România există doar câteva produse care se apropie de problema observată, dar acestea nu o abordează într-un mod eficient deoarece nu sunt complete , accesibile sau integrate.

Prima dintre ele este destinată doar către operatorul CFR Călători și CFR Infrastructură și este aplicația informatică utilizată intern: xSellData³. Este o aplicație software dezvoltată de către compania de Informatică Feroviară – Infofer, destinată administrării și versionării tuturor datelor operatorului național: mers de tren, compuneri, tarife etc. Această aplicație și modulele ei conexe sunt folosite de toate casieriile CFR din țară, de reviziile de vagoane, de automatele de vânzare de bilete din gări și de serviciul on-line de rezervare și plată a CFR Călători. Față de soluția pe care o propun în cele ce urmează, această aplicație se ocupă doar cu administrarea trenurilor care circulă constant, dar este constrânsă de operatorul de transport feroviar. Deși prin acest software se administrează rutele și compunerile, componenta lui majoră o reprezintă rezervarea și cumpărarea biletelor de tren. Cu toate acestea, se poate observa destul de ușor că nici aceasta nu este completă (nu se pot cumpăra online sau la automate bilete cu reduceri, nu există posibilitatea utilizării de POS la multe agenții, modalitățile de rezervare a locului sunt limitate, chiar și la casierii).

Aplicația xSellData expune un modul online disponibil pentru călătorii CFR prin care aceștia pot consulta mersul de tren și își pot calcula călătoriile. Este disponibil pe internet, dar conține date doar pentru operatorul național. Pentru a căuta trenuri operate de alte companii, un utilizator este nevoit să viziteze site-urile acestor companii.

Tot compania Informatică Feroviară a realizat sistemul prin care stațiile CFR dotate cu PC și conexiune la Internet oferă detalii despre trenurile care tranzitează respectivele stații. Aplicația Iris este folosită de către impiegații de mișcare din stații astfel: prin integrarea cu sistemul menționat anterior (xSellData), impiegații au acces la lista cu trenurile care sunt programate să tranziteze stația unde ei sunt de serviciu; în momentul în care un tren a oprit sau a tranzitat respectiva stație, ei raportează manual în aplicație data și ora la care a sosit și

³ <http://www.infofer.ro/ct-menu-item-19>

la care a plecat. Această metodă este total ineficientă deoarece sunt foarte multe stații fără impiegat dispozitor de mișcare (toate haltele de călători) și sunt suficiente stații care, chiar dacă au impiegat de serviciu, nu au PC sau conexiune la internet. De asemenea, datele sunt trecute de impiegat manual și astfel sunt predispuse la erori și, mai grav, la neconcordanță cu realitatea.

Software-ul Iris oferă și el un modul prin care călătorii au acces la datele în timp real cu privire la parcursul unui tren. Aplicația TrenulMeu⁴ este accesibilă online pentru oricine. Este însă dificil de folosit deoarece nu este responsive pentru mobil și, pentru a căuta un tren, trebuie știut în prealabil numărul sub care circulă. Pe lângă acestea, informațiile în timp real pe care le oferă sunt, de fapt, cele introduse manual de către impiegatii dispozitori de mișcare, deci este foarte posibil să nu fie de actualitate sau să fie eronate. De asemenea, TrenulMeu oferă date doar despre trenurile operate de CFR Călători.



Toate aceste sisteme software sunt realizate folosind tehnologii stabile, dar vechi: Visual Basic.NET, VB-Script, ASP.NET WebForms. În prezent există soluții tehnice mai eficiente și elegante din perspectiva rezultatului obținut.

⁴ <http://appiris.infofer.ro/MyTrainRO.aspx>

Totodată, compania care produce și modernizează locomotive, Softronic Craiova, a dezvoltat un computer de bord pentru locomotivele reparate de ei - ICOL⁵. Acest computer doar îi afișează mecanicului de tren date pe un monitor (curentul pe motoare, tensiune în catenară, curenți pe bateria de acumulatori și treapta curentă a graduatorului – pentru locomotivele electrice care au controlul tracțiunii în trepte). În poza 1 se observă, în stânga-jos, computerul de bord.



Poza 1 – Post conducere al unei locomotive modernizate cu instalație ICOL

O altă companie care efectuează reparații locomotivelor - INDA⁶, oferă o soluție similară cu cea propusă de Softronic. Deși implementarea tehnică diferă, computerul de bord are aceleași funcționalități: îi afișează digital mecanicului de tren o serie de parametri. Nici această soluție nu este integrată cu alte soluții și nu trimite date în timp real. În poza 2 se observă, în stânga-jos, o altă variantă de computer de bord de la INDA.

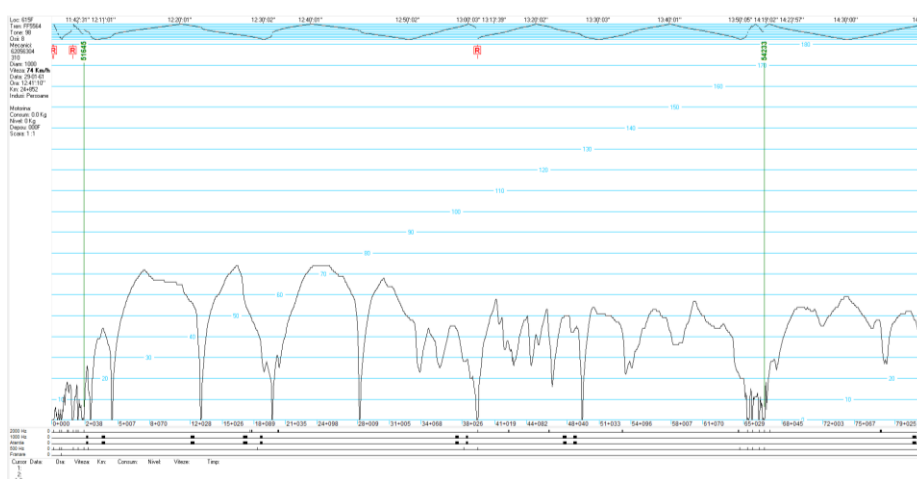
⁵ <https://www.softronic.ro/modernizari.html>

⁶ <http://www.indar.ro/noutati.html>



Poza 2 – Post de conducere al unei locomotive modernizate cu instalație INDА

Majoritatea locomotivelor din România care nu au trecut prin modernizări, sunt dotate cu vitezometru IVMS⁷. Acesta permite înregistrarea pe un suport magnetic a vitezei trenului, a datelor despre mecanicul și cazuri în care nu au fost respectate indicațiile semnalelor din cale (depășire de viteză, depășire de semnale etc.) În departamentele de tracțiune de la CFR Călători există un software cu care se pot citi și interpreta aceste înregistrări. O captură de ecran cu o astfel de înregistrare interpretată se observă în poza 3.



Poza 3 – citire date IVMS

⁷ <https://www.softronic.ro/>

Soluțiile implementate în momentul de față acoperă doar anumite părți din problema expusă și, după cum am observat, nu sunt integrate într-o abordare completă, nu sunt scalabile și nici extensibile. În prezent, cu software-urile disponibile fiese pot administra trenuri, fie doar se pot emit bilete, fie poate vedea mecanicul de serviciu unii parametri elctro-mecanici ai trenului remorcat.

3 SOLUȚIA PROPUȘĂ

„If builders built houses the way programmers built programs, the first woodpecker to come along would destroy civilization.”

- Gerald M. Weinberg, Mastering Software Quality Assurance

Acest capitol conține o privire de ansamblu a soluției propuse pentru problema monitorizării sistemului feroviar. În prima parte este vorba despre arhitectura sistemului construit: rețeaua de senzori, serviciile IoT și Cloud, aplicațiile și serviciile web de prezentare a datelor către operatori și călători.

În partea a doua sunt prezentate detaliile de implementare cu exemple de cod și configurări.

Subcapitolul al treilea prezintă funcționalitățile dezvoltate cu exemple din aplicații.

Ultima parte este destinată prezentării tehnologiilor folosite: limbajele și tehnologiile de programare, serviciile cloud.

3.1 Arhitectura soluției

Pentru ca problema discutată să poată fi rezolvată, în primul rând corect și apoi eficient, cu un produs final care respectă obiectivele propuse și care să poată fi folosit ușor de către utilizatorii cărora le este adresat, este necesară analiza în detaliu a problemei și dezvoltarea unei imagini de ansamblu pentru tot sistemul informatic ce urmează a fi construit.

Aici intră domeniul arhitecturii de sistem. Acesta se referă la structurile de nivel înalt (unde nu contează detalii de implementare) și la legăturile dintre acestea. Pentru început, trebuie proiectate componentele, definite modelele și marcarea structurilor fundamentale a sistemului.

Ulterior, trebuie luați în calcul factorii care rămân constanți pe toată durata de viață a proiectului: grupurile de utilizatori care vor folosi sistemul și rolurile lor, organizarea rețelei de senzori și modul de comunicație cu ei, platforma în care senzorii sunt agregați și

administrați, aplicațiile prin care utilizatorii comunică cu sistemul, mediile de stocare necesare și integrarea sistemului cu servicii externe.

Cerințele funcționale pe care sistemul trebuie să le îndeplinească sunt:

- pe fiecare locomotivă urmărită trebuie instalat un senzor capabil să trimită date de telemetrie sau mesaje speciale;
- senzorii trebuie agregați și administrați din perspectiva tehnică;
- senzorii trebuie agregați și administrați din perspectiva urmăririi de către operatorii de transport feroviar;
- operatorii de transport feroviar se pot înscrie în platformă și pot utiliza senzorii puși la dispoziție și conturile pentru angajații săi;
- operatorii de transport feroviar pot să își administreze propriile date care țin de domeniul de business – feroviar: rute cu opriri itinerarice, material rulant de tracțiune și senzorii aferenți, trenuri zilnice;
- fiecare operator are acces la o bază de date comună cu informații referitoare la domeniul de business – feroviar: stațiile existente pe infrastructura din România, tipurile de material rulant de tracțiune și categoriile de motoare și echipamente electro-mecanice;
- această bază de date și regulile de acces pentru operator trebuie să fie administrabile;
- călătorii pot vedea în timp real stațiile din România cu opririle programate și trenurile pe care operatorii de transport feroviar le-au făcut publice.

Fiind vorba atât de un software enterprise, cât și de o rețea de senzori IoT, cerințele non-funcționale sunt diverse:

- clienții (angajați ai operatorilor de transport sau călătorii) trebuie să poată accesa platforma într-un mod simplu, disponibil pe orice device (laptop, desktop, smartphone), într-un mod securizat și conform drepturilor de acces ale fiecăruia;
- datele fiecărui operator trebuie să fie independente și să nu poată fi accesate de utilizatorii fără drepturile necesare;

- platforma trebuie să fie modulară, extensibilă (să suporte mai mulți operatori, fiecare cu potențialele reguli proprii de business, mai multe tipuri de senzori etc.) și mai ales scalabilă;
- componentele IoT trebuie să fie eficiente din punct de vedere al consumului (curent electric, trafic de date, calcul computațional delegat către servere), iar infrastructura completă să suporte un flux de date ridicat (mii de dispozitive senzori, milioane de mesaje/secundă);
- mediul de stocare trebuie să fie de încredere - indicat este să aibă asigurarea replicării datelor și distribuirea încărcării pe mai multe centre;
- mediul de stocare trebuie să fie ales în funcție de tipul de date: datele de business sunt relaționate natural, iar datele de telemetrie sunt variate, se transmit cu o viteză foarte mare și vor ocupa un volum mult mai mare;
- integrarea cu serviciile externe trebuie să aibă loc cu cel mai mic grad de dificultate.

În figura 1 este prezentată diagrama de nivel înalt (arhitectura high-level) a sistemului.

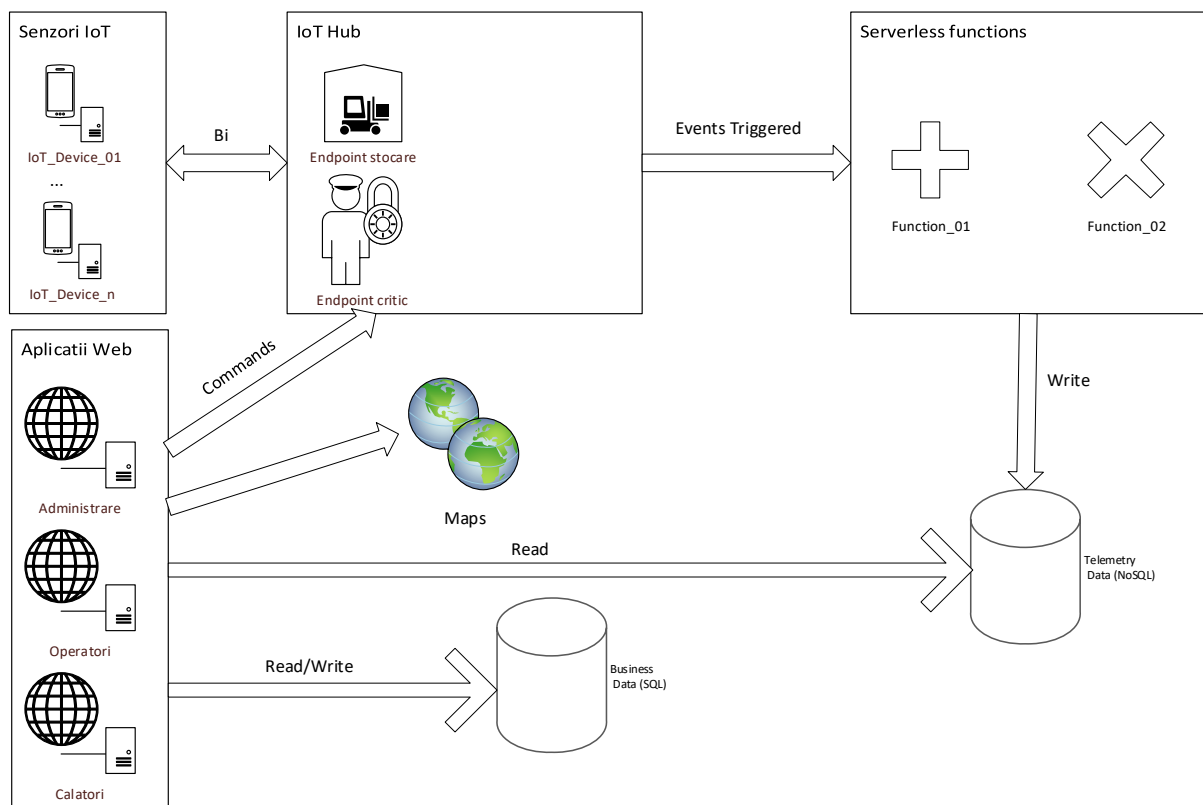


Figura 1

Arhitectura este organizată cu următoarele componente majore:

Device-uri IoT: acestea sunt mici calculatoare care vor fi instalate pe locomotivele monitorizate; fiecare device este legat la circuitele locomotivei (de exemplu la ampermetrele de pe motoarele de tracțiune și la voltmetrul pantografului) și, prin conexiunea la date mobile, are acces la coordonate de geolocație (latitudine, longitudine, altitudine și viteză). Tot prin internet se trimit mesaje (telemetrie, alerte etc.) către Hub-ul configurat. Device-urile rulează un sistem de operare și au instalat un software prin care se citesc datele de la senzori și de unde se transmit mai departe. Aceste device-uri măsoară la intervale regulate și configurabile de timp datele de la senzorii instalați.

IoT Hub: reprezintă locul de comunicare între senzori și restul aplicației. În Hub ajung mesajele de la senzori, aici sunt înregistrate și urmărite device-urile, prin acest canal se pot realiza operații de mentenanță remote a senzorilor (update, interogări de stare etc.) și tot prin intermediul Hub-ului se comunică de la aplicații către device-uri. Se pot configura mai multe rute pentru mesajele primite, în funcție de destinația lor (datele de telemetrie sunt stocate, datele critice precum o depășire de semnal sunt semnalizate special etc.). Rutele sunt configurabile și se pot adăuga pe viitor în funcție de nevoile clienților.

Serverless Functions: sunt componente software care încep execuția automat în funcție de evenimentul declanșator. Se leagă de IoT Hub astfel: în funcție de tipul de mesaj trimis de senzori (telemetrie, alertă), acestea sunt rutate diferit de Hub. În funcție de ruta pe care este trimis un mesaj, se declanșează evenimentul care pornește execuția funcției aferente: fiecare tip de mesaj va declanșa funcția aferentă și aceasta va rula codul din interior (stocare date de telemetrie, trimitere notificare etc.). Funcțiile sunt configurabile și extensibile după cum apar necesități viitoare.

Storage NoSQL: mesajele de la senzori vin în număr foarte mare (înseamnă automat viteză și volum). Pe lângă aceasta, soluția este proiectată să reziste schimbărilor potențiale (pentru un anumit tip de locomotivă să poată fi măsurat în viitor și alt parametru fizic – de exemplu temperatura din baia de ulei a Transformatorului Principal sau senzorii folosiți să trimită date în alt format). Acest lucru înseamnă că mediul de stocare trebuie să fie adaptat pentru viteză, volum, dar și varietate, iar cea mai bună implementare pentru astfel de criterii este un storage NoSQL de tip cheie – valoare.

Storage SQL: domeniul de business ales – cel feroviar – este compus dintr-un număr mare de entități relaționate între ele. În acest sistem sunt operatori care au mai multe rute, rute care trec prin mai multe stații, fiecare rută are câte un tren care circulă în respectiva trasă zilnic, fiecare tren are în compunere una sau mai multe locomotive și aceste locomotive sunt legate și la câte un Device IoT. Cel mai natural mod de stocare pentru un astfel de sistem îl reprezintă o bază de date relaționată. Aceasta, pe lângă tabele, are utilizatori tehnici, politici de securitate și reguli de acces.

Aplicație Web Administrare: toată platforma (indiferent că vorbim despre rețeaua de senzori sau de datele specifice domeniului feroviar) trebuie administrată într-un mod unitar și agnostic la tehnologiile folosite. Aplicația de administrare este o aplicație web (accesibilă din internet) prin care provider-ul platformei poate să înregistreze operatori de transport feroviar noi, poate să asigneze acestora conturi de utilizator pentru angajați, poate să definească stațiile acoperite din rețeaua feroviară națională și poate să gestioneze tipurile de locomotive și de senzori pentru care oferă suport la un anumit moment. Tot din această componentă, provider-ul platformei poate să înregistreze în Hub device-uri noi, să le administreze tehnic și să le aloce operatorilor de transport feroviar care le-au cerut.

Aplicație Web Operatori: angajații unui operator de transport feroviar (fie că vorbim de operatori de date, mecanici, șefi de tură în depouri sau utilizatori cu rol de administrator de sistem) își desfășoară activitatea legată de urmărirea trenurilor și a locomotivelor din acest site (tot o aplicație web disponibilă în internet și cu aceleași cerințe non-funcționale ca ale celei de administrare). Aplicația oferă autentificare și autorizare și este folosită de către angajații operatorului pentru a defini trase și trenuri curente, pentru a administra parcul de material rulant, pentru a urmări în timp real locomotivele și parametrii acestora și pentru gestionarea utilizatorilor web ai aplicației

Aplicație Web Călători: este o hartă real-time în care persoanele care călătoresc folosind trenul în România au acces la informații de interes pentru ei. Pot vedea în locația geografică corespunzătoare orice stație de cale ferată din țară, pot urmări opririle programate pentru respectiva stație. Mai mult, pot vedea în timp real, pe hartă, locomotivele pentru care operatorul de transport feroviar care le deține și le gestionează a lăsat posibilitatea să fie urmărite și de publicul extern.

Maps: aplicațiile web au nevoie de poziționarea datelor geospațiale primite de la senzori într-un mod intuitiv pentru utilizatori. Este nevoie de integrarea cu un sistem extern care să funcționeze ca un serviciu care oferă diverse tipuri de hărți. Sunt necesare date despre zona României și informații legate de căile ferate naționale. Serviciul trebuie să fie extensibil și configurabil, deoarece în viitor putem monitoriza și elemente de infrastructură feroviară.

3.2 Implementare hardware / software

Dispozitive IoT

Am folosit computere Raspberry Pi (2 și 3)⁸ ca end-device-uri pentru senzorii de pe locomotive. Deși aceste modele sunt construite mai mult în scop didactic, pentru realizarea unui produs funcțional minimal sunt suficiente, iar pentru lansarea în mediul de producție se poate trece, fără dezvoltări noi, la alternativa Compute Module⁹ de la același producător.

Pe dispozitivele Raspberry Pi am instalat sistemul de operare Windows 10 Core¹⁰. Acesta este special conceput pentru dispozitivele de acest gen, pentru roboți industriali și, în general, pentru device-uri care controlează sisteme electro-mecanice sau care înregistrează date. O altă variantă ar fi fost utilizarea unui microcontroller și programarea acestuia să măsoare și să transmită date spre cloud. Însă, o soluție formată dintr-un Raspberry Pi și un sistem de operare propriu se încadrează în aceleași costuri și este mult mai scalabilă: se poate conecta la un display și, fiind instalat deja un sistem de operare, utilizarea lui în producție este mult mai facilă (în prezent, locomotivele dotate cu instalație ICOL au instalat pe computerul de bord sistemul de operare Windows CE).

Pe sistemul de operare rulează o aplicație Universal Windows Platform, scrisă în C#, folosind tehnologii .NET Core, care gestionează citirea datelor și comunicația cu back-end-ul cloud. Transmisia de date se face prin Internet prin protocolul MQTT¹¹. Aplicația, prin API-urile expuse de modelul UWP, poate citi orice pin de intrare (astfel, înregistrează senzorii conectați) sau poate scrie orice pin de ieșire (așadar, controlează echipamentele electrice).

⁸ <https://www.raspberrypi.org/>

⁹ <https://www.raspberrypi.org/products/compute-module-1/>

¹⁰ <https://developer.microsoft.com/en-us/windows/iot>

¹¹ <http://mqtt.org/>

Senzorii sunt legați prin circuite de adaptare la pinii de I/O ai computer-ului (sunt doar pini digitali). Pentru că majoritatea senzorilor transmit totuși semnale analogice, am folosit un convertor analog-numeric, MCP3008¹².

Prin realizarea unui sistem integrat, aceste componente funcționează astfel: la intervale predefinite se citesc date pe pinii la care sunt legați senzori. Aceste date de telemetrie sunt împachetate și trimise către end-point-ul din Hub-ul de IoT. Dacă senzorii care înregistrează date critice (de exemplu, instalația INDUȘI a locomotivei care semnalizează depășirea unui semnal care ordona oprirea) se activează, atunci se declanșează un eveniment care trimite un mesaj de alertă către end-point-ul din Hub-ul de IoT. Aplicația este configurată ca un task de background: pornește automat în momentul în care sistemul bootează.

În figura 1 este prezentată o schemă electrică pentru prototipul instalației. Se pot citi senzori digitali (direct) și analogici (prin intermediul unui circuit ADC) și se pot controla consumatori digitali.

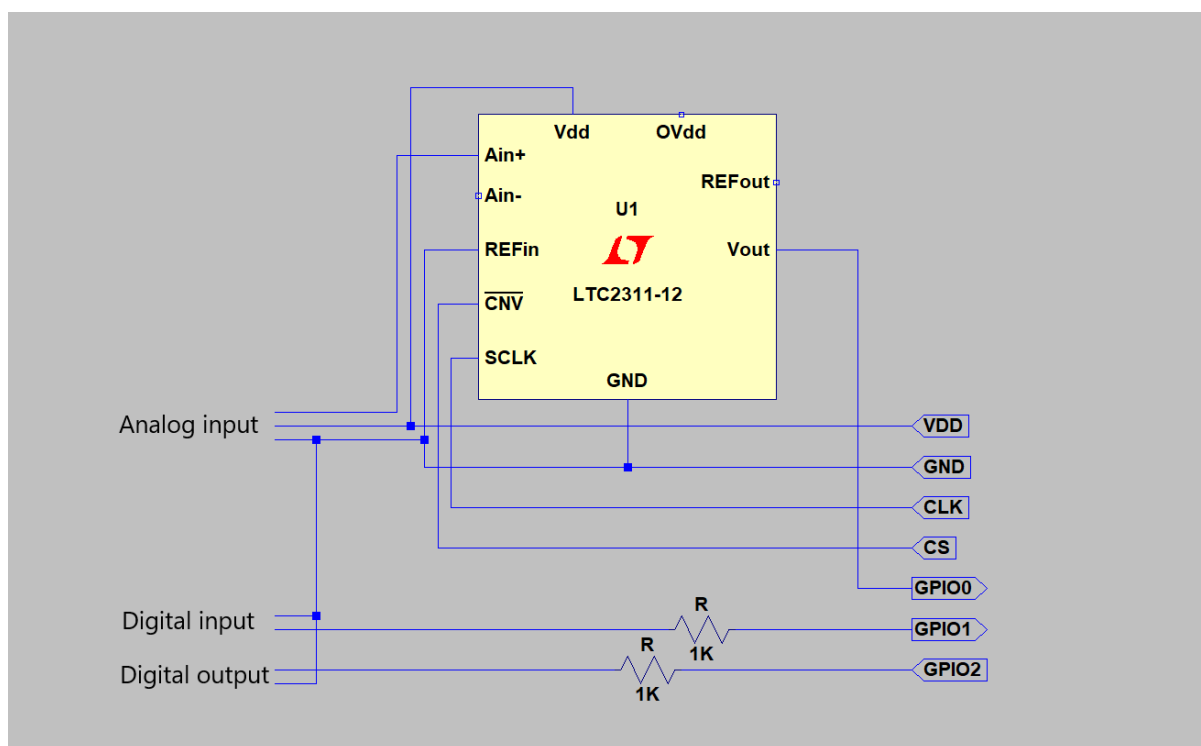


Figura 1

¹² <https://www.microchip.com/wwwproducts/en/MCP3008>

Unul din scenariile avute în vedere la implementarea soluției a fost modul în care datele colectate ar putea ajuta la eficientizarea transportului. În prezent, locomotivele nu sunt rulate la capacitatea lor: există trenuri de călători care sunt tractate de locomotive de 6800CP și care transportă 2 sau 3 vagoane de călători (deci până în 75 de tone). Folosind următoarele formule de calcul și având date despre vitezele trenurilor monitorizate, putem alocă locomotive din clase mai puternice pentru trenurile care au tonaje ridicate.

Variația tonajului remorcat în funcție de viteză $M(v)$:

$$M_V = \frac{F_{ac} - R_{Lo} - m_L g i}{g(r_{ve} + i_{er})} [t] \text{ în care:}$$

F_{ac} – forța de tracțiune la obadă (de pe caracteristica de tracțiune) $[kN]$

R_{Lo} – forța rezistentă a locomotivei $[N]$

$$R_{Lo} = g \left[296 + 7.068 \left(\frac{v}{10} \right)^2 \right] [N]$$

v – viteza $[km/h]$

m_L – masa locomotivei (din cartea tehnică) $[T]$

g – accelerația gravitațională, $9.81 [m/s^2]$

i – declivitatea echivalentă rezultantă pe rampa caracteristică (din livrete) $[\%]$

r_{ve} – rezistența totală a vagoanelor $[N/kN]$

$$r_{ve} = \alpha_i r_{vi} + \alpha_g r_{vg}$$

$$\alpha_i = 0.25 + \frac{N_s}{N_{sg}} 0.5 [\%] \text{ (procentul de vagoane încărcate)}$$

$$\alpha_g = 0.25 - \frac{N_s}{N_{sg}} 0.5 [\%] \text{ (procentul de vagoane goale)}$$

$$r_{vi} = 2 + \frac{v^2}{1950} [N/kN] \text{ (rezistența specifică a vagoanelor încărcate pe 2 și 4 osii)}$$

$$r_{vg} = 2 + \frac{v^2}{850} [N/kN] \text{ (rezistența specifică a vagoanelor goale pe 2 și 4 osii)}$$

IoT Hub

Implementarea folosită este soluția IoT Hub de la Microsoft Azure. Aceasta permite înregistrarea securizată și administrarea dispozitivelor end-device (atât prin portalul Azure, cât și programatic, prin SDK-urile puse la dispoziție), interogarea dispozitivelor și realizarea de update-uri remote, dar mai ales primirea și trimiterea de mesaje (device-to-cloud și cloud-to-device).

În figura 2 este prezentat traseul mesajelor de la device până la procesare.

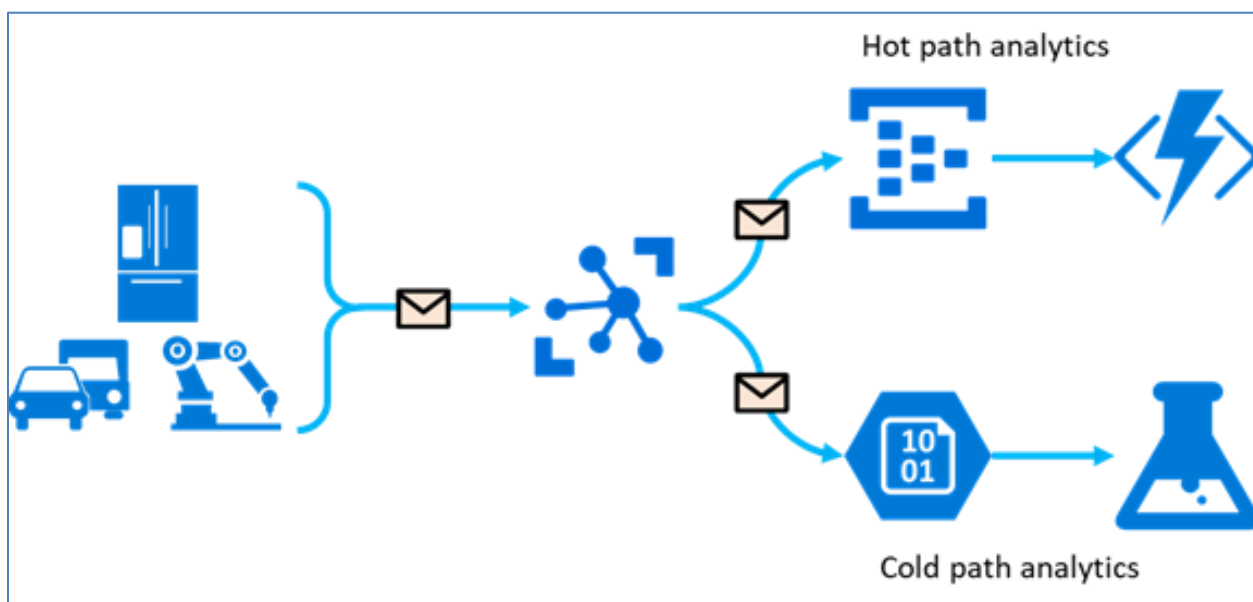
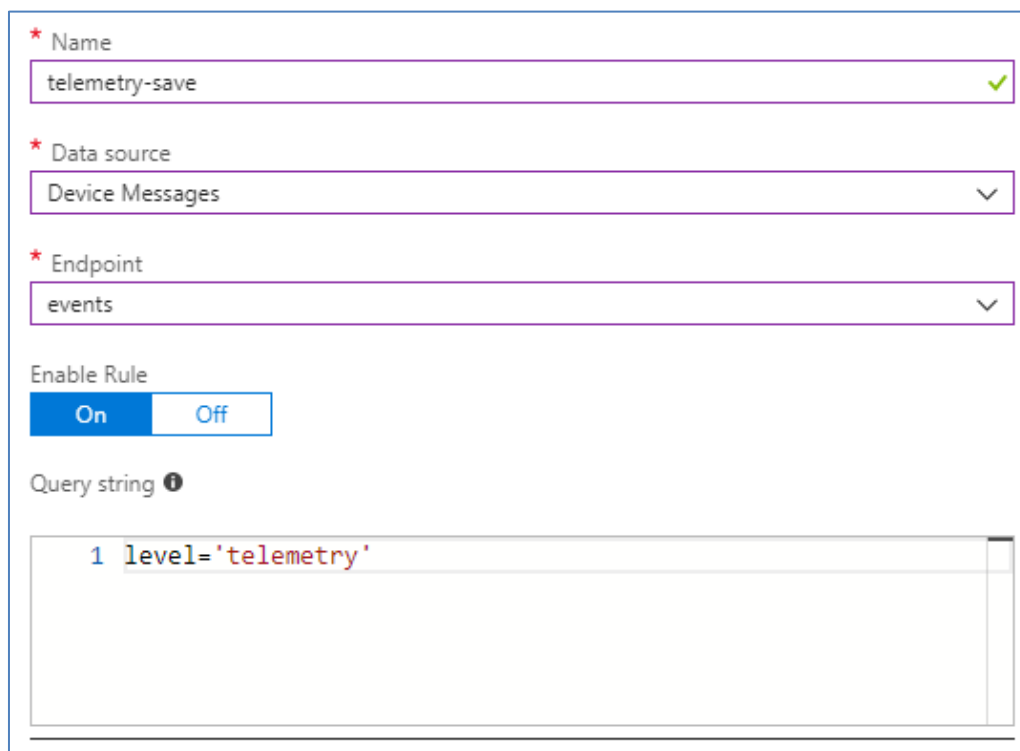


Figura 2 - rutare mesaje¹³

Mesajele primite pot fi rutate eficient către un end-point în funcție de tipul de mesaj trimis (telemetrie, alertă etc.). Rutare se poate face după multe filtre, inclusiv date din corpul mesajului. Mai departe, este responsabilitatea end-point-ului în care ajung mesajele ce se întâmplă cu acestea (de exemplu unde se stochează).

¹³ N., Berdy (21.09.2017), *Route IoT device messages to Azure Storage with Azure IoT Hub*, <https://azure.microsoft.com/en-us/blog/route-iot-device-messages-to-azure-storage-with-azure-iot-hub/>

În figura 4 este prezentată configurarea unei rute către endpoint-ul events pentru mesajele care conțin câmpul „level” cu valoarea „telemetry”.



The screenshot shows the configuration for a new rule in the Azure IoT Hub console. The rule is named "telemetry-save" and is enabled. It uses "Device Messages" as the data source and routes to the "events" endpoint. The query string is set to "1 level='telemetry'", which filters for messages where the level is telemetry.

Field	Value
Name	telemetry-save
Data source	Device Messages
Endpoint	events
Enable Rule	On
Query string	1 level='telemetry'

Figura 4

Hub-ul, fiind una din componentele critice ale sistemului propus, este necesară și o analiză de performanță. În România circulă zilnic aproximativ 1000 de trenuri. La o medie de 5 ore/tren și un mesaj trimis de pe fiecare locomotivă o dată la 5 secunde, rezultă aproximativ 3.600.000 de mesaje zilnice, în condițiile în care fiecare operator și-ar dota parcul de locomotive cu astfel de senzori. Configurația utilizată în prezent pe Hub-ul implementat suportă 400.000 de mesaje pe zi (deci un număr mediu de 250 de trenuri/zi), dar poate scala fără efort până la 300.000.000 de mesaje zilnice.

Funcții Serverless

Mesajele primite de Hub-ul IoT trebuie tratate diferit în funcție de scopul lor. Dacă Hub-ul se ocupă cu gestionarea mesajelor și rutarea lor către diferite end-point-uri, din momentul în care mesajele ajung pe un end-point specific, acestea trebuie procesate corespunzător.

Funcțiile serverless sunt mici aplicații care rulează într-un mediu container și care încep execuția automat în funcție de evenimentele declanșatoare. În soluția propusă, o funcție serverless este implementată pentru a se ocupa de stocarea mesajelor de telemetrie, iar o funcție tratează mesajele critice .

Funcțiile sunt configurate să se declanșeze automat când un mesaj este rutat spre un end-point. Acestea primesc ca parametri mesajul cu toate datele trimise de la device-ul IoT și, după ce îl deserializează și îl pre-procesează, pot face diferite acțiuni: îl salvează în mediul de stocare configurat (în cazul mesajelor de telemetrie) sau trimite un mail utilizatorilor care au nevoie să primească această informație (de exemplu, când un device IoT a înregistrat o depășire de semnal, se declanșează o întrerupere care este tratată special și care declanșează trimiterea unui e-mail către personalul autorizat).

Cod C# pentru funcția care primește mesajele de la end-point-ul events și le persistă în baza de date:

```
public static void Run(string myIoTHubMessage, TraceWriter log)
{
    var storageAccount = CloudStorageAccount.Parse("connectionString");
    var tableClient = storageAccount.CreateCloudTableClient();
    var table = tableClient.GetTableReference("TableName");
    table.CreateIfNotExists();

    var telemetryData = JsonConvert.DeserializeObject<dynamic>(myIoTHubMessage);

    var partitionKey = telemetryData.DeviceId.ToString();
    var rowKey = string.Format("{0:D19}",
        DateTime.MaxValue.Ticks - DateTime.UtcNow.Ticks);

    var telemetryDataEntry = new TelemetryDataEntity(partitionKey, rowKey)
    {
        DeviceId = telemetryData.DeviceId,
        OperatorId = telemetryData.OperatorId,
        Date = DateTime.Now,
        Telemetry = myIoTHubMessage
    };

    var insertOperation = TableOperation.Insert(telemetryDataEntry);
    table.Execute(insertOperation);
}
```

Asemănător, pot fi implementate funcții serverless care să execute diferite procese logice.

Storage NoSQL

Anterior, am detaliat motivele pentru care soluția cea mai bună de stocare a mesajelor de telemetrie o reprezintă o bază de date NoSQL de tip cheie – valoare: date cu un volum mare, foarte variate și care apar încontinuu.

Implementarea abordată este Azure Tables Storage. Am definit un tabel pentru stocare în formă cheie-valoare, iar datele de telemetrie sunt introduse după următorul principiu: cheia de partiționare (partition key) trebuie aleasă astfel încât să nu fragmenteze prea mult datele și să permită interogări eficiente, iar pentru cheia de rând (row key) trebuie luat în calcul faptul ca ordonarea implicită se face după ordinea lexicografică a acestora.

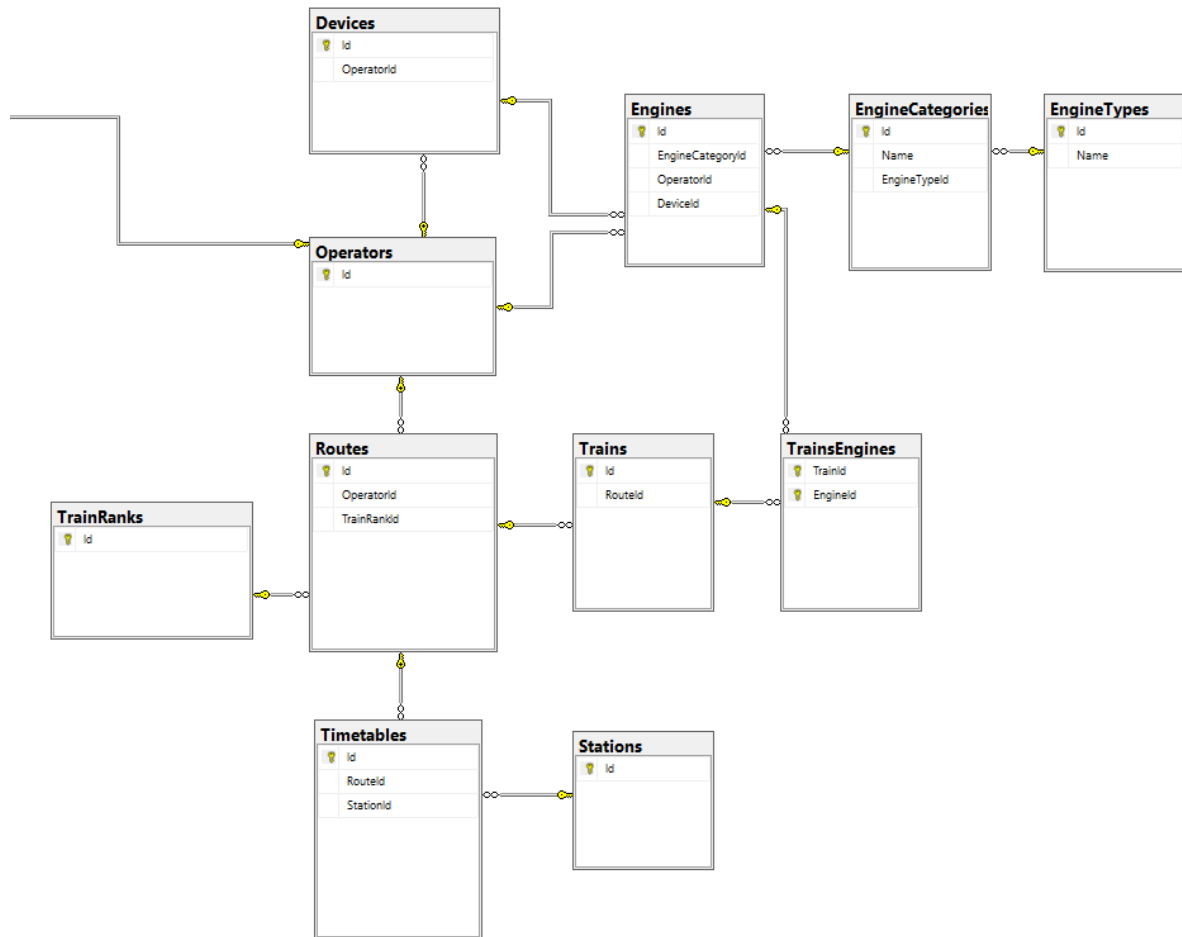
Astfel, am ales să partiționez tabelul după id-ul unic al device-ului care trimite mesaje (în felul acesta va exista câte o partiție pentru fiecare device) și cheia de rând va fi calculată la insert ca fiind diferența dintre ora UTC maximă care se poate reprezenta și ora UTC la care s-a primit mesajul (asigurând astfel o stocare naturală, în ordine descrescătoare a mesajelor în funcție de data la care au fost primite).

Storage SQL

Mediul de stocare relațional este folosit pentru a persista entități din domeniul de business feroviar. Pentru dezvoltarea bazei de date la nivel local am folosit Microsoft SQL Server 2016, iar în momentul în care făcut deploy aplicațiilor am trecut și la o stocare în cloud – tot un server SQL.

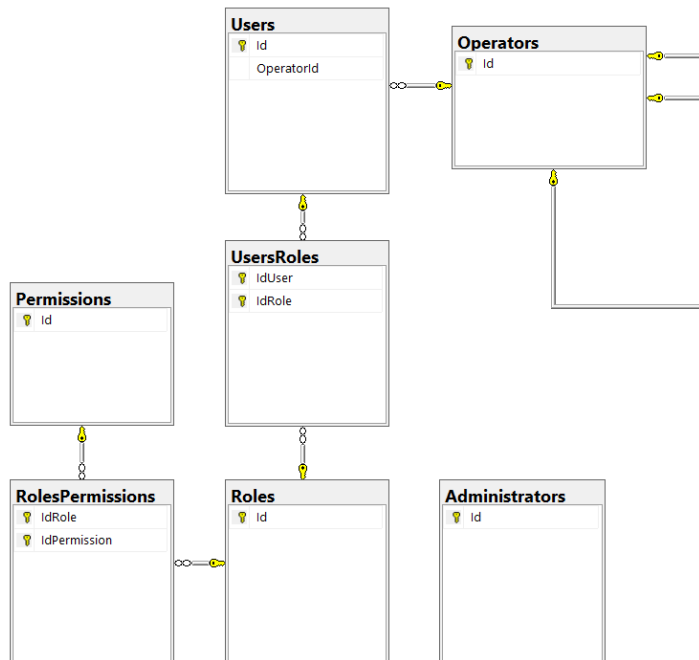
Atât serverul, cât și principala bază de date conținută sunt securizate folosind conturi de utilizatori tehnici, printr-o politică de acces care elimină permisiunile complet pentru IP-uri client care sunt din exteriorul infrastructurii Azure.

Pentru a modela domeniul de business, am implementat următoarele tabele, cu relații între ele, baza de date fiind normalizată:



- Un operator de transport (*Operators*) are unul sau mai multe dispozitive (*Devices*);
- Un dispozitiv (*Devices*) este asignat la o locomotivă (*Engines*);
- O locomotivă (*Engines*) se află într-o categorie (*EngineCategories*), care are un singur tip (*EngineTypes*);
- Una sau mai multe locomotive compun un tren (*Trains*, prin tabelul de legătură *TrainsEngines*);
- Trenurile (*Trains*) circulă în baza unei trase (*Routes*);
- Un operator de transport (*Operators*) operează mai multe trase (*Routes*);
- Fiecare trasă (*Routes*) are un rang (*TrainRanks*);
- Fiecare trasă (*Routes*) are una sau mai multe opriri (*Timetables*);
- Fiecare oprire (*Timetables*) se petrece în fix o stație (*Stations*).

Pentru a persista informațiile critice pentru autentificarea utilizatorilor și definirea resurselor pe care le pot accesa și a permisiunilor din aplicațiile client, am extins baza de date cu următoarele tabele:



- Un operator de transport (*Operators*) are unul sau mai mulți utilizatori (*Users*);
- Un utilizator (*Users*) face parte dintr-unul sau mai multe roluri (*Roles*, prin tabelul de legătură *UsersRoles*);
- Un rol (*Roles*) are una sau mai multe permisiuni (*Permissions*, prin tabelul de legătură *RolesPermissions*);
- Separat de utilizatorii standard (angajații operatorului de transport), sunt administratorii platformei (*Administrators*).

Aplicații web

Toate aplicațiile web sunt realizate folosind același stack de tehnologii (.NET Core 2) și implementează o arhitectură de tip “Clean” sau “Onion”¹⁴.

O arhitectură standard, multi-layer, ar fi fost suficientă într-o primă instanță, dar evoluția ulterioară a aplicațiilor și schimbările din domeniul de business ar fi fost mai dificile de implementat. În plus, organizarea codului sub forma “Onion” permite o tranziție mult mai lejeră către microservicii și virtualizare sub formă de containere (Docker¹⁵).

Această arhitectură respectă următoarea diagramă din figura 5, astfel: nivelul de aplicație (UI) referențiază direct nivelul de logică de business, care referențiază direct nivelul de acces la date. Deși este respectat principiul “Separation of concerns”, straturile aplicației depind direct unele de altele. Nivelul logicii de business, deși este centrul aplicației, are dependențe directe către servicii care țin de infrastructură (accesul la date). Deși, în teorie, se pot reutiliza componente, în cazul de față ar fi dificil de schimbat cu alte implementări. De exemplu, deoarece nivelul de acces la date este referențiat direct și nu prin interfețe, în codul din logica de business vor exista dependențe către clase concrete din proiectul de date. Mici schimbări în cod sau înlocuirea unor layere cu altele duce la refactorizări și rescrieri masive.

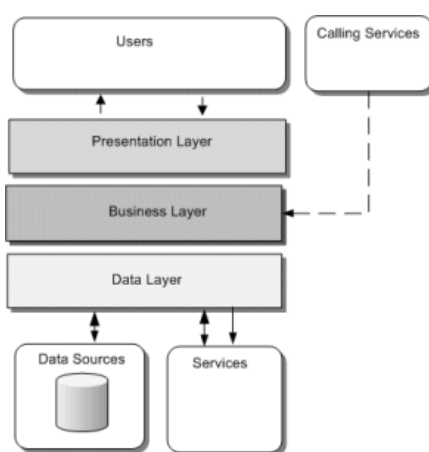


Figura 5 - Arhitectura n-layer¹⁶

¹⁴ <https://www.martinfowler.com/bliki/PresentationDomainDataLayering.html>

¹⁵ <https://www.docker.com/>

¹⁶ J.D., Meier (05.11.2008), *Layers and Tiers*, <https://blogs.msdn.microsoft.com/jmeier/2008/09/05/layers-and-tiers/>

După cum am menționat, am preferat să evit acest model arhitectural (oarecum clasic) și să trec la o variantă proprie de arhitectură Onion (figura 6).

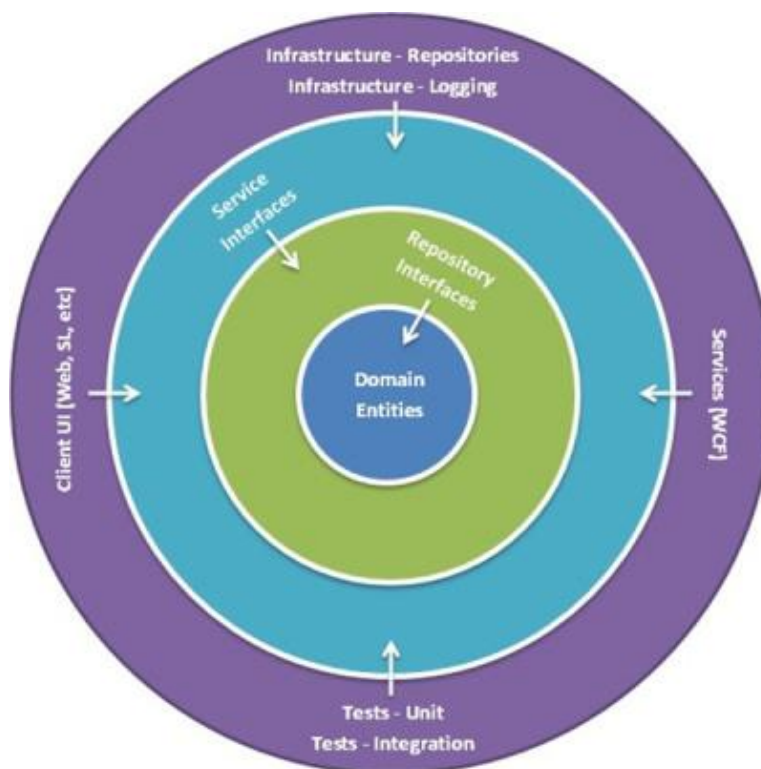


Figura 6 - Arhitectură Onion¹⁷

Față de varianta precedentă, îmbunătățirile aduse sunt următoarele: în centrul aplicației stă, domeniul de business – adică problema pe care vrem să o modelăm. Există un proiect cu entitățile de business cu care aplicația lucrează (de exemplu Rute, Operatori, Utilizatori, Device-uri, Stații sau Opriri).

Peste aceste entități există un proiect cu servicii pentru fiecare entitate în parte: clase care definesc funcționalitățile posibile (de exemplu adăugarea unei noi rute, înregistrarea unui utilizator, căutarea unui utilizator după credențialele furnizate sau înregistrarea unui device IoT). Serviciile, în mod evident, trebuie să folosească infrastructură externă: tabele din baza de date, servicii cloud etc., dar toate sunt reprezentate prin interfețe, nu prin clase concrete.

¹⁷ S. Grech (31.08.2014), *Onion-izing your multi-tier architecture*, <https://www.incredible-web.com/blog/the-onion-architecture/>

În zona de infrastructură există implementări concrete pentru clasele așteptate de proiectul de servicii de business. Este folosit EntityFramework Core 2.0 pentru acces la baza SQL și SDK-ul Azure pentru servicii pentru Storage-ul NoSQL sau pentru conexiunea cu IoT Hub.

La nivelul de aplicație sunt construite, folosind ASP MVC Core 2.0, cele trei aplicații web: pentru Administratori, Operatori și Călători. În acest nivel se configurează implementările concrete pentru interfețele cu care lucrează serviciile de business, setările de aplicație și datele de conectare la diversele servicii.

Structura în detaliu a arhitecturii aplicațiilor se observă în arborele de proiecte din figura 7.

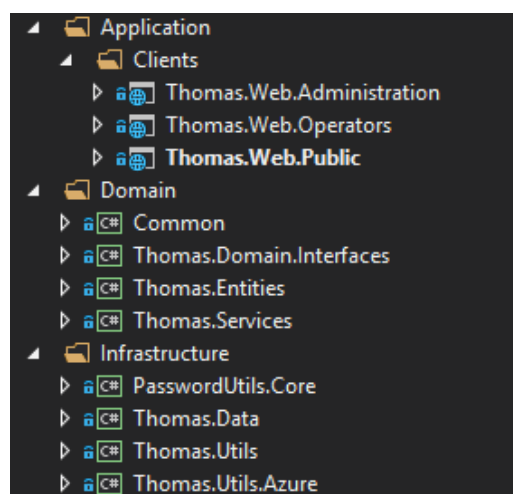


Figura 7

La nivel de domeniu există: proiectul de entități (*Thomas.Entities*) în care sunt definite tipurile de date de business cu care lucrează aplicația (Rute, Trenuri, Locomotive, Stații etc), proiectul de servicii cu logica de business utilizabilă (*Thomas.Services*), iar acestea depind de exterior prin interfețele specificate (*Thomas.Domain.Interfaces*).

La nivel de infrastructura există în primul rând implementări: pentru acces la baza de date (*Thomas.Data* – folosind Entity Framework Core 2.0) și pentru servicii cloud (*Thomas.Utils.Azure*). Proiectul *Thomas.Utils* conține funcționalități utile în mai multe zone (metode de extensie, helpere etc.), iar *PasswordUtils.Core* conține o bucată extrasă din framework-ul Microsoft.Identity.Core, folosită pentru hashing-ul și verificarea parolelor

(acestea nu sunt salvate în clar, ci sub formă de șir de caractere securizat prin algoritmi de criptare – SHA 256¹⁸).

În final, la nivel de aplicație sunt proiectele ASP MVC Core corespunzătoare celor trei aplicații client. Acestea respectă o arhitectură internă ușor de extins și implementat: fiecare resursă (Tren, Stație, Rută etc.) are un model propriu, diferit de clasele din proiectul de entități. Maparea dintre ele se face automat folosind Automapper¹⁹, iar validarea modelelor trimise de utilizatori se realizează prin clase de FluentValidation²⁰. Autentificarea se face folosind claim-uri²¹ transportate prin cookies. Fiecare resursă are politici de autorizare proprii, astfel încât un utilizator care nu are drepturi, nu poate accesa sau modifica respectiva resursă (de exemplu, un utilizator al unui operator nu poate accesa deloc resurse ale altui operator). Pe client sunt trimise View-uri HTML care sunt decorate cu elemente de CSS. Framework-ul utilizat pe client este Bootstrap pentru design, cu elemente de JavaScript și jQuery pentru o experiență cursivă.

Toate layer-ele sunt configurate la nivel de aplicație, în momentul în care aceasta pornește. Este implementat pattern-ul arhitectural Inversion of Control, prin configurarea container-ului de Dependency Injection²² pus la dispoziție de ASP astfel: pentru fiecare interfață sau serviciu care ar putea fi utilizat pe durata de viață a aplicațiilor este specificată ce implementare concretă trebuie folosită și care este durata de viață a obiectului care va fi injectat (tranzitiv, per-request sau singleton²³).

¹⁸ <https://tools.ietf.org/html/rfc4634>

¹⁹ <https://automapper.org/>

²⁰ <https://github.com/JeremySkinner/FluentValidation>

²¹ <https://docs.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-2.1>

²² <https://www.martinfowler.com/articles/injection.html>

²³ <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-2.1>

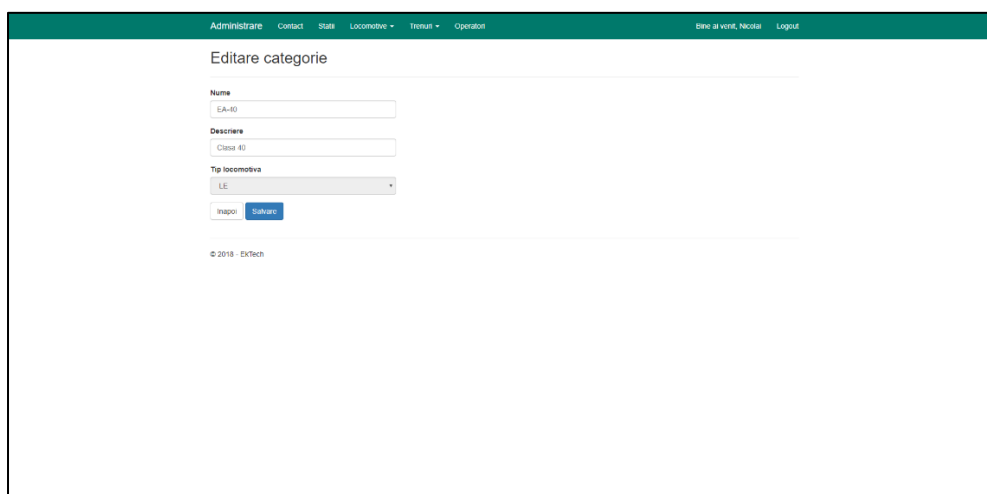
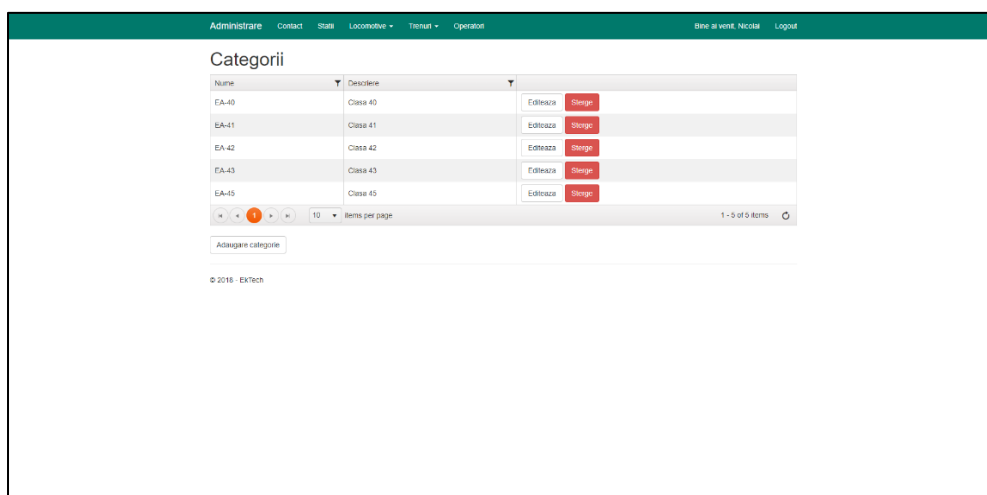
3.3 Fluxuri aplicație

Aplicație de administrare

Din aplicația de administrare se pot configura entități din domeniul feroviar care pot fi folosite de orice operator de transport: categorii de locomotive și tipuri de motoare, stații de cale ferată, asignare de utilizatori și senzori către un operator etc.

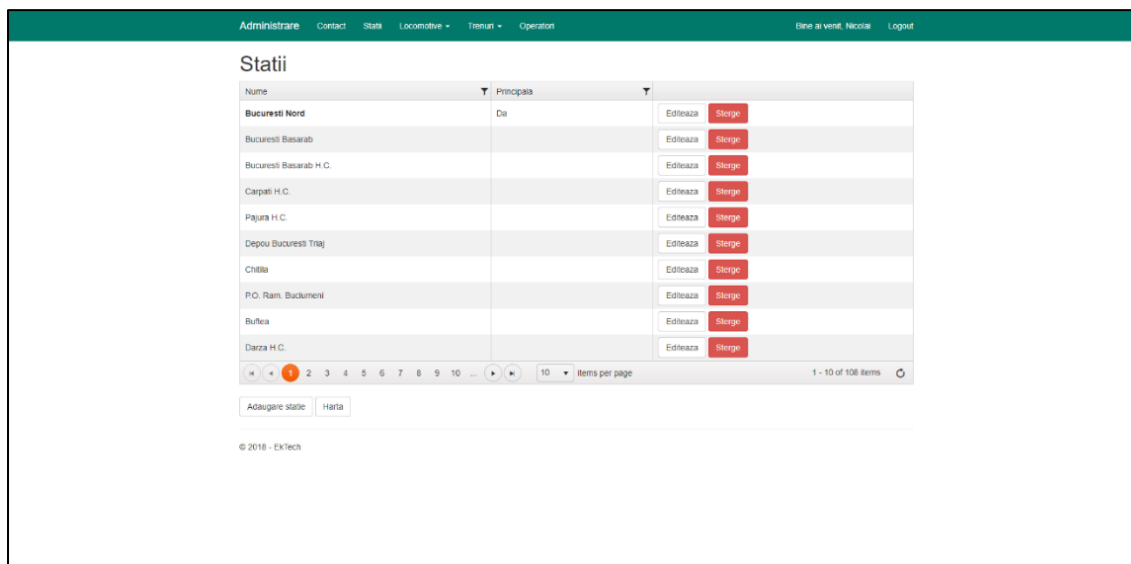
1. Administrare categorii de locomotive

Categoriile de locomotive și tipurile de motoare pe care aceste locomotive le au sunt nomenclatoare editabile. Din lista cu elemente se poate alege să se creeze unul nou, să se editeze un element existent sau să se șteargă. Pe server se verifică dacă operația poate fi efectuată – există permisiuni pentru utilizator și elementul nu este referențiat de o altă entitate. Modificările sunt persistate în baza de date SQL.

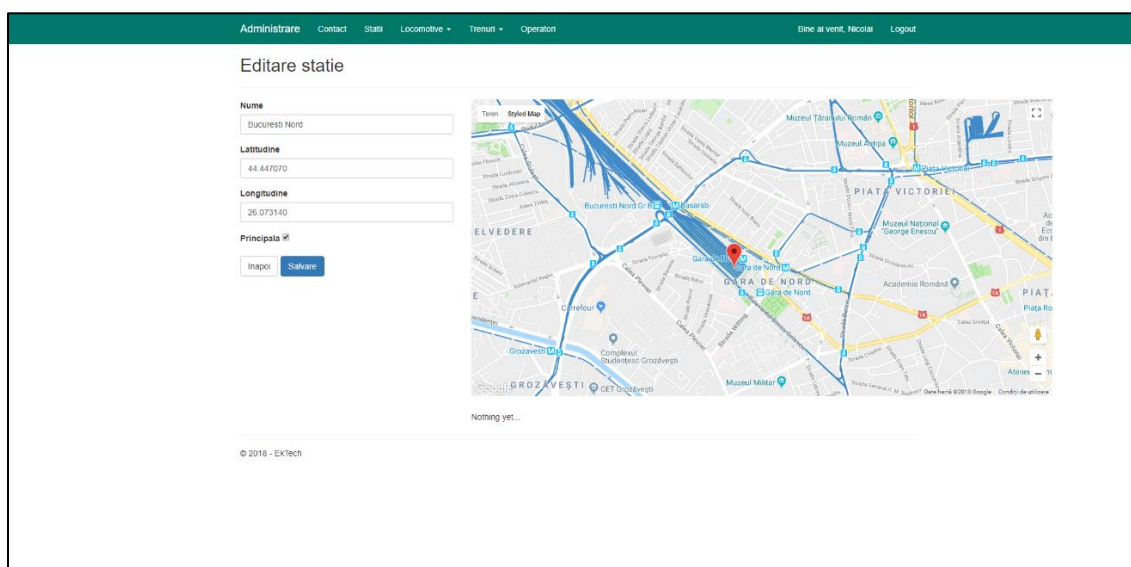


2. Administrare stații C.F.

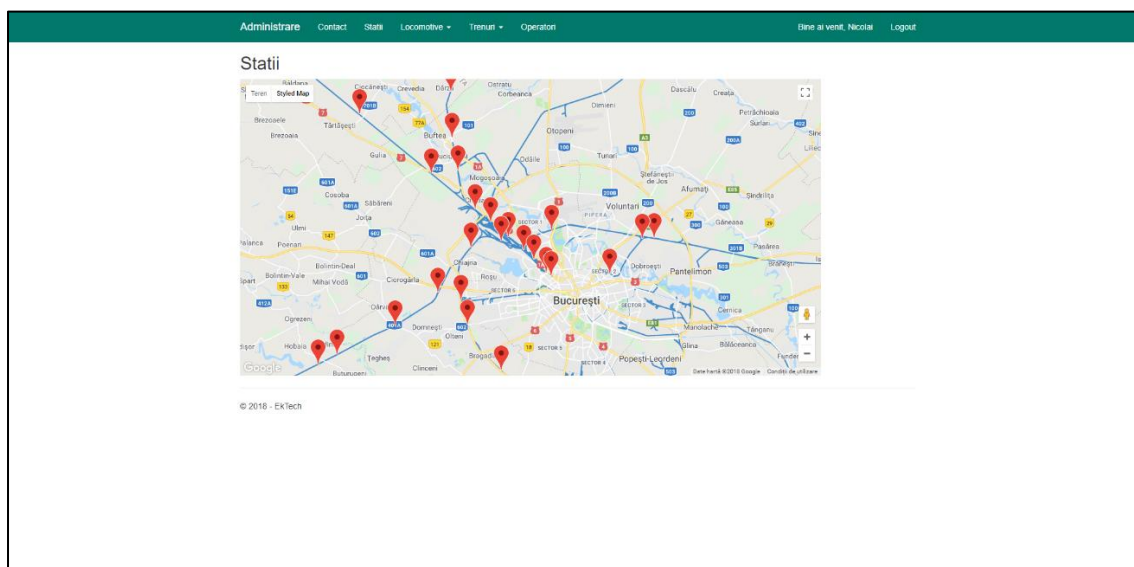
Stațiile de cale ferată sunt obiecte care pot fi folosite de orice operator de transport. Similar cu celelalte tipuri de entități, stațiile pot fi adăugate, șterse și modificate, iar modalitatea de lucru este unitară.



Pentru a ușura sarcina utilizatorului, administrarea stațiilor utilizează serviciul de hărți: se localizează automat coordonatele geografice în funcție de locul unde este amplasat pin-ul corespunzător stației.

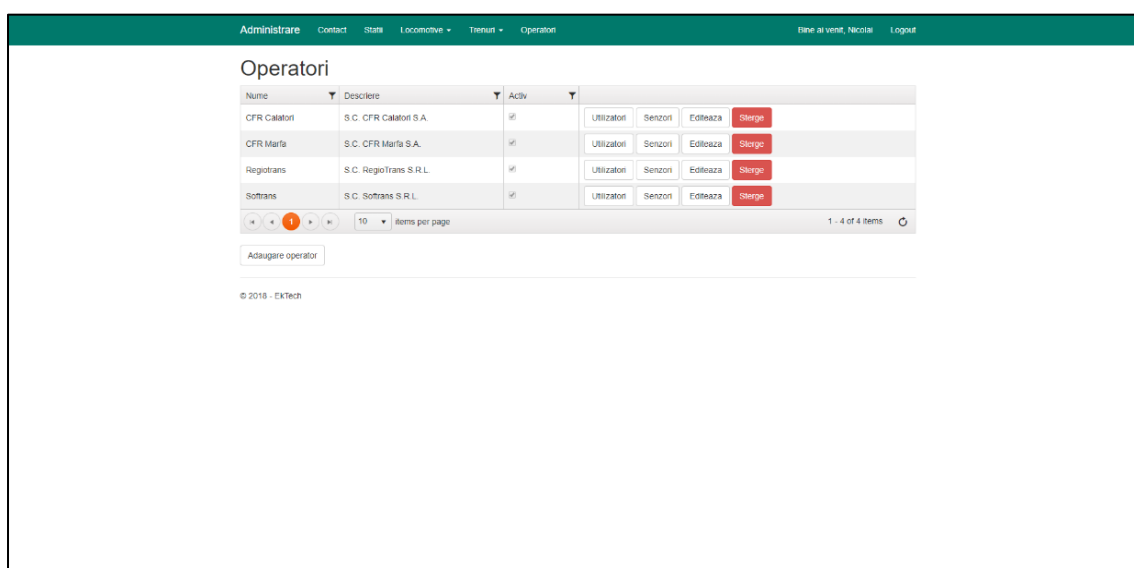


Există posibilitatea de a vedea pe hartă toate stațiile existente în aplicație, dar și liniile de cale ferată de pe teritoriul României.



3. Administrare operatori în platformă

Operatorii înregistrați din platformă pot fi administrați individual: se pot adăuga operatori noi, iar pentru fiecare în parte se pot alocă device-uri IoT pe care să le monitorizeze, și conturi de utilizator pentru aplicația web.

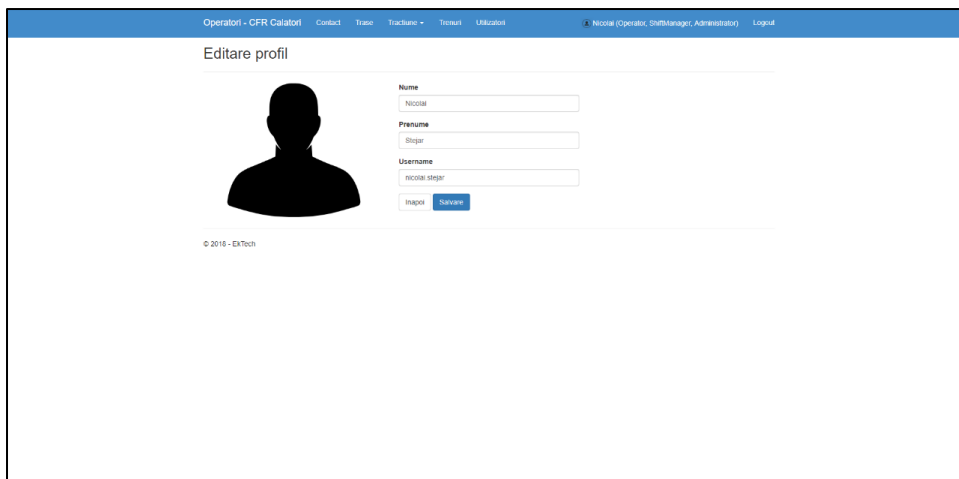


Aplicație pentru operatori

Aplicația destinată operatorilor este o aplicație web separată de cea de administrare. Aici se pot loga (cu username, parolă și id de operator) utilizatorii operatorilor de transport feroviar înregistrați în platformă.

1. Setări profil utilizator (oricine)

Utilizatorii își pot modifica datele de profil.



The screenshot displays the 'Editare profil' page. The header includes navigation links: Operatori - CFR Calatori, Contact, Trece, Tractare, Trece, Utilizatori. The user is logged in as Nicolai (Operator, ShiftManager, Administrator). The form contains the following fields:

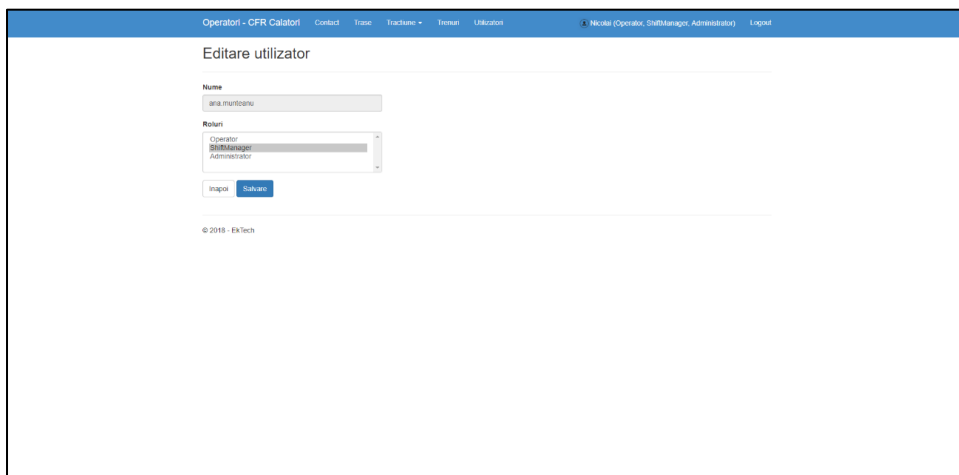
- Name: Nicolai
- Prenume: Stefan
- Username: nicolai.stefan

Buttons: Inapoi, Salvare

© 2018 - EkTech

2. Administrare roluri pentru conturile primite (doar de utilizatorii care sunt „Administrator”)

Utilizatorii cu rol de administrator pot alocă alți utilizatori în diverse roluri. Fiecare rol are o serie de permisiuni posibile în aplicație (editare trenuri, editare utilizatori, monitorizare senzori etc.).



The screenshot displays the 'Editare utilizator' page. The header is the same as the previous page. The form contains the following fields:

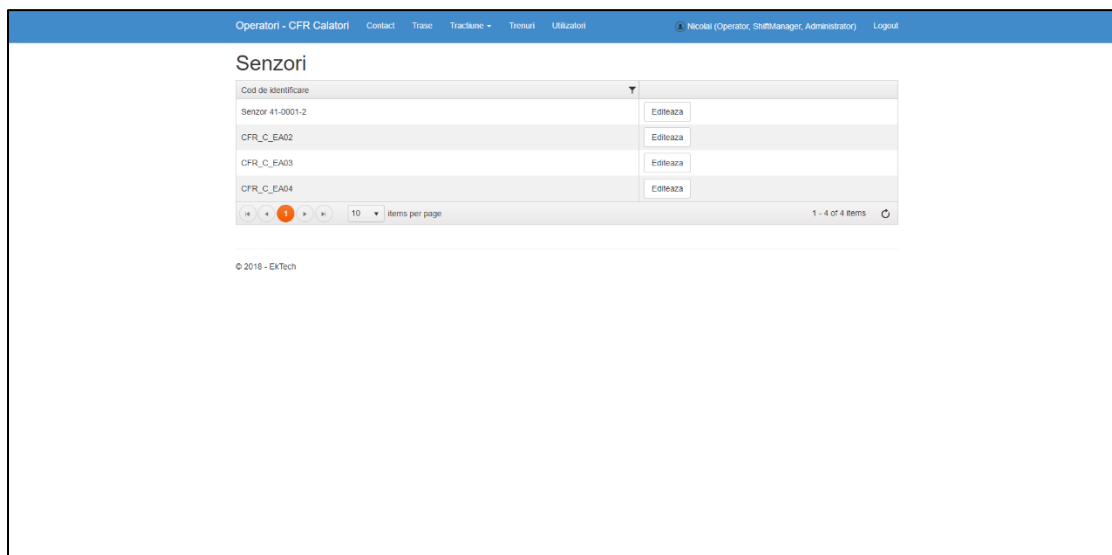
- Name: aria.murtoanu
- Roluri: Operator, ShiftManager, Administrator

Buttons: Inapoi, Salvare

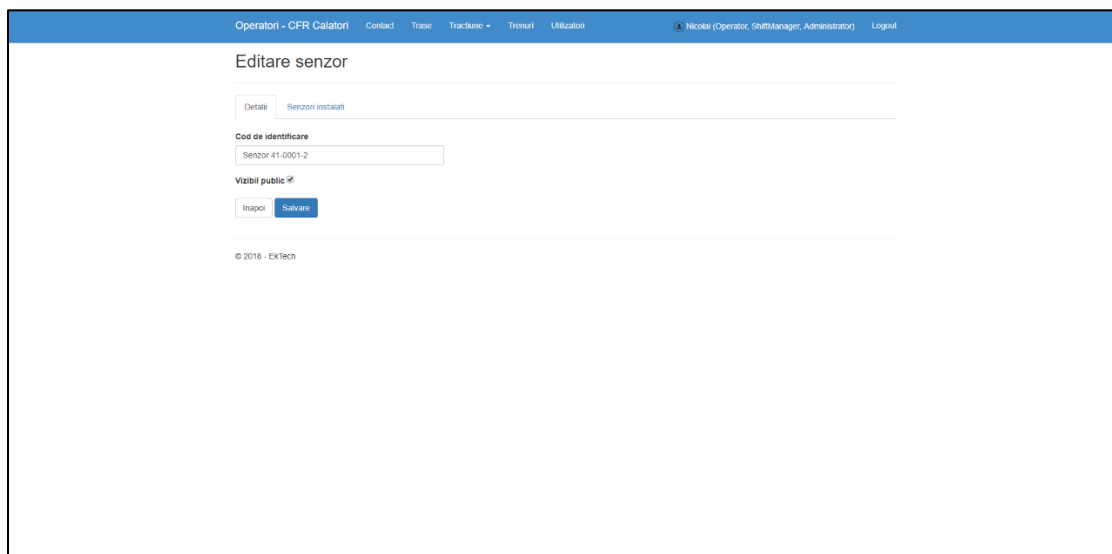
© 2018 - EkTech

3. Administrare senzori primiți (doar de utilizatorii care sunt „Operator”)

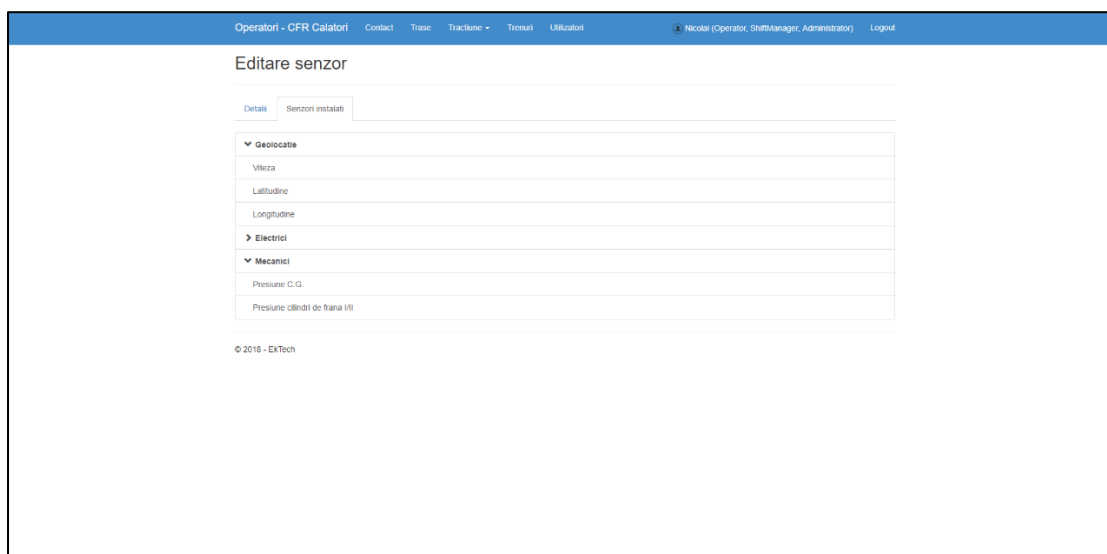
Senzorii atribuiți din platforma de administrare unui operator pot fi configurați și la nivel de operator. Datele la care utilizatorii unui operator au acces sunt doar date de business, neavând niciun impact asupra modului în care senzorii comunică cu Hub-ul de IoT.



Pentru un senzor poate fi configurat numele sub care este găsit și setările de accesibilitate: dacă poate fi urmărit și de alte servicii externe sau doar de operatorul curent (cel care îl deține).

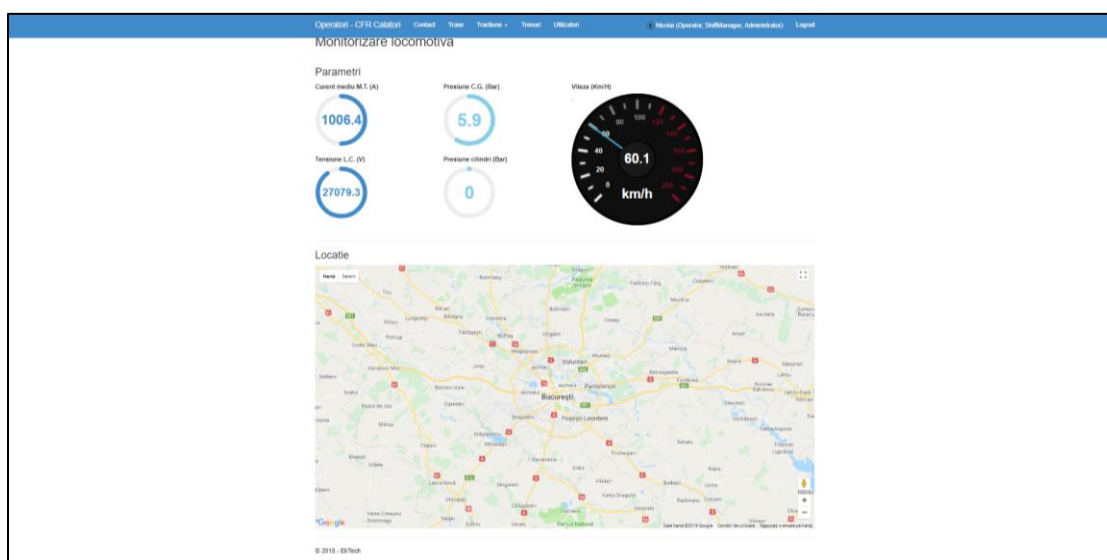


Pentru fiecare senzor asignat unui operator, acesta poate vedea și semnalele pe care senzorul le poate măsura de pe locomotivă: ce instalații electrice sau mecanice sunt conectate direct la senzor.



4. Monitorizare locomotivă în timp real (doar de utilizatorii care sunt „Operator”)

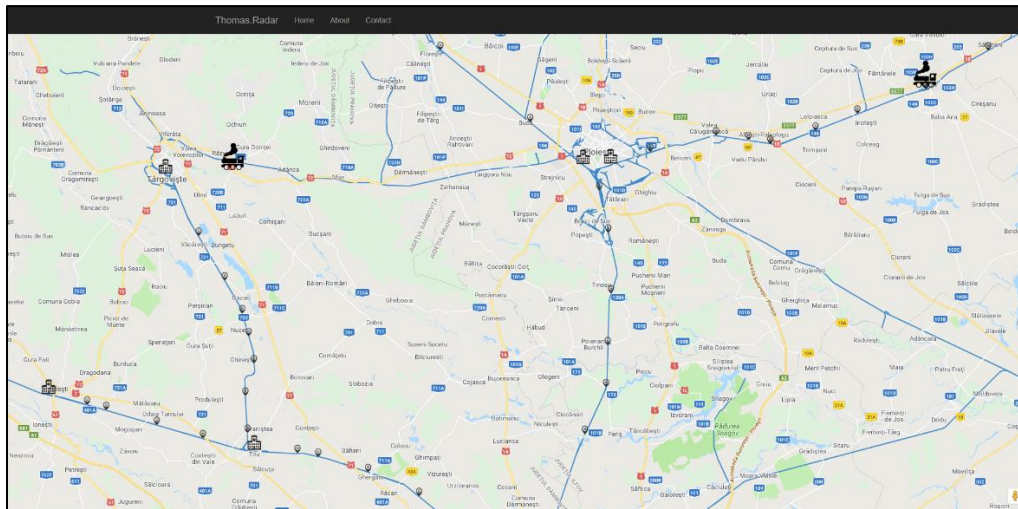
Utilizatorii unui operator pot urmări în timp real o locomotivă, dacă aceasta are un senzor alocat în aplicație (și instalat fizic). Parametrii mășurați din teren sunt afișați astfel încât semnificația lor să fie evidentă. Locomotiva este plasată pe harta la coordonatele trimise de senzor.



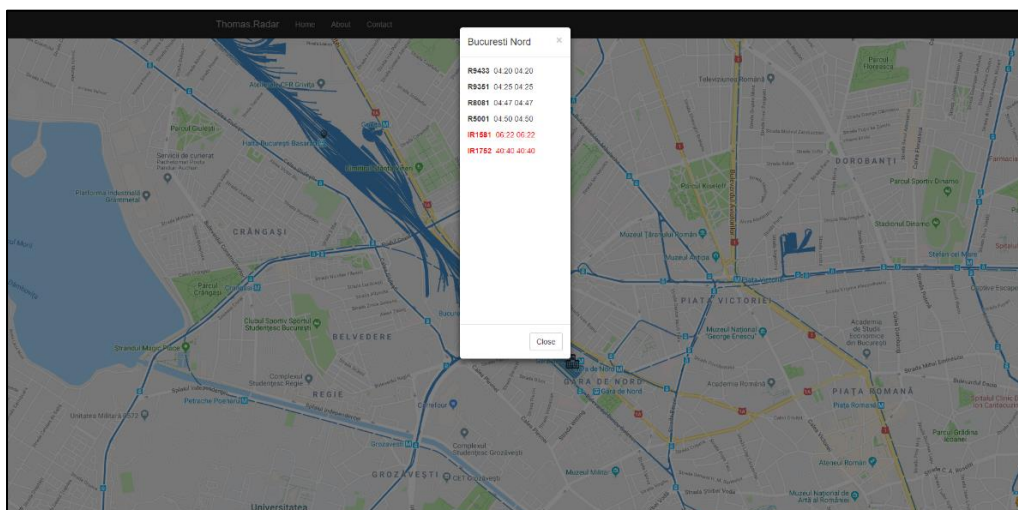
Aplicație pentru călători

1. Monitorizare în timp real trenuri și stații

Aplicația destinată călătorilor este o aplicație independentă de celelalte două. Din acest modul, un călător poate vedea informații despre trenurile care au locomotive unde au fost instalate device-uri IoT și care au fost configurate ca fiind publice. Harta conține informații în timp real despre trenuri și despre stațiile administrate din platformă.



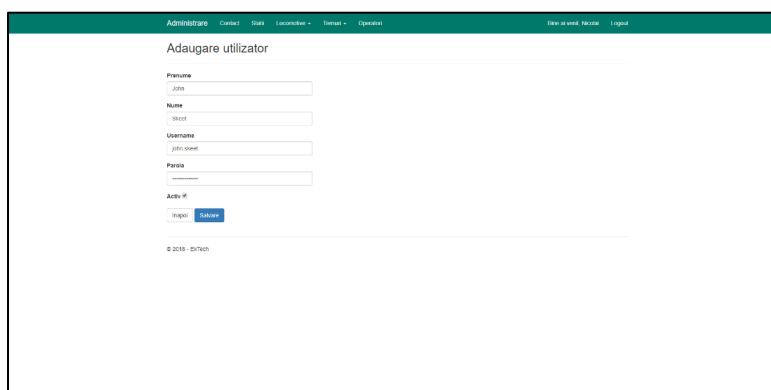
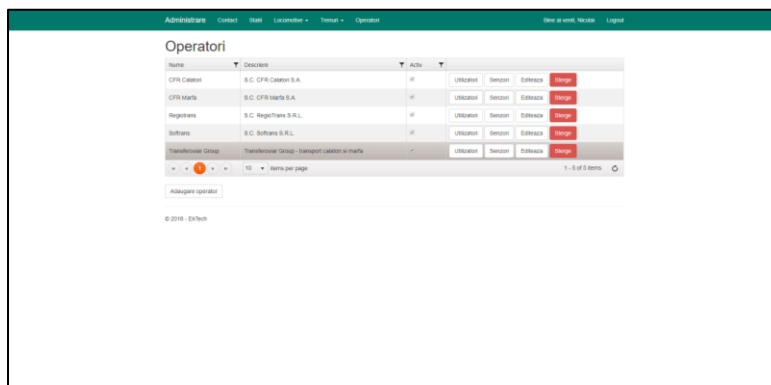
Pentru fiecare stație, un călător poate vedea ce trenuri de călători sunt programate în ziua respectivă să o tranziteze.



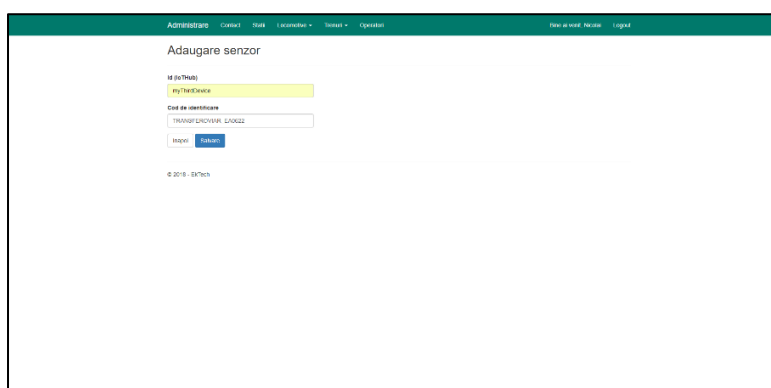
3.4 Studiu de caz – configurare și monitorizare tren

Din platforma de administrare, se alocă unui operator de transport feroviar un cont de utilizator și un device IoT. Device-ul va fi montat fizic pe locomotivă și legat logic în aplicație la o locomotivă. Din aplicația operatorilor este configurat dispozitivul.

1. Din aplicația de administrare este adăugat contul de utilizator. Implicit, conturile noi alocate nu fac parte din niciun grup, rămânând la latitudinea operatorului să adauge utilizatorul în rolurile dorite.



2. Din aplicația de administrare este adăugat un dispozitiv IoT. Acesta se înregistrează în Hub-ul din cloud și este asignat operatorului de transport.



3. Din aplicația operatorilor este necesară autentificarea cu contul creat anterior.

The screenshot shows the 'Login' page of the 'Operator' application. The page has a blue header with 'Operator' and 'Contact' links, and a 'Login' button on the right. The main content area contains a login form with the following fields: 'OTF' (a dropdown menu showing 'Transferoviar Group'), 'Utilizator' (a text input field with 'john.doe' entered), and 'Parola' (a password input field with dots). Below the password field is a 'Login' button. To the right of the form is a small illustration of a train engine. At the bottom left, there is a copyright notice: '© 2019 - ERTech'.

După ce personalul operatorului a adăugat utilizatorul nou în grupurile dorite, acesta are posibilitatea să execute acțiuni în aplicație. În meniul de navigație apar și rolurile curente.

The screenshot shows the dashboard of the 'Operator' application. The header is blue and contains the following links: 'Operator - Transferoviar Group', 'Contact', 'Trase', 'Tracțiune', 'Tensiuni', 'Utilizatori', and a user profile 'John (Operator, ShiftManager, Administrator)' with a 'Logout' button. The main content area features a large image of a train on tracks with the text 'Informații în timp real' and a loading spinner. Below the image are four sections: 'Funcționalități' (with links to 'Definirea rutei', 'Administrarea grupurilor', 'Monitorizarea locomotivei', 'Controlul locomotivei', and 'Notificarea deranjamentelor'), 'Servicii' (with links to 'Informația feroviară' and 'Softing'), 'OTF' (with links to 'OTF Cabluri', 'CPIS Logare', and 'Regiotrans'), and 'Tehnologii' (with links to 'Integrare de date', 'ASFINET Core', 'Entity Framework Core', and 'Raspberry Pi'). At the bottom left, there is a copyright notice: '© 2019 - ERTech'.

4. Utilizatorul poate să adauge o locomotivă nouă sau poate să o editeze una existentă. Se pot edita date legate de locomotivă sau se poate lega la senzorul primit.

The screenshot shows the 'Adaugare locomotiva' (Add locomotive) page in the 'Operator' application. The page has a blue header with the same navigation links as the dashboard. The main content area contains a form with the following fields: 'Numar' (a text input field with '40-9822-7' entered), 'Numar LIC' (a text input field), 'Clasa' (a dropdown menu showing 'CA-40'), and 'Id Senzor' (a dropdown menu). Below the 'Id Senzor' field are two buttons: 'Inapoi' and 'Salvare'. At the bottom left, there is a copyright notice: '© 2019 - ERTech'.

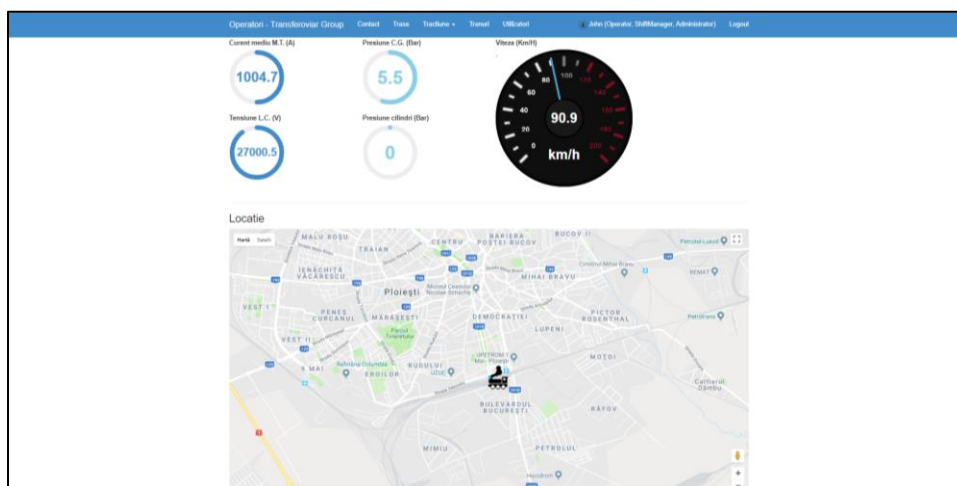
Device-ul IoT poate fi ales doar dintre cei alocați operatorului și disponibili (dintre cei care nu au fost asignați altei locomotive ale aceluiași operator).

The screenshot shows a web application interface for editing a locomotive. The title is 'Editare locomotiva'. The form contains the following fields:

- Numar**: A text input field containing '40-0522-7'.
- Numar UIC**: An empty text input field.
- Categorie**: A dropdown menu with 'EA-40' selected.
- Id Senzor**: A dropdown menu with 'TRANSFEROVAR_EA0022' selected. Below it, a blue button with the text 'TRANSFEROVAR_EA0022' is visible.

At the bottom left of the form, there is a small copyright notice: '© 2018 - ExTech'.

Din momentul în care locomotiva începe să circule, device-ul pornește și începe să transmită date. Date sunt monitorizate în timp real din aplicația operatorilor, doar de către utilizatorii operatorului care deține locomotiva și device-ul IoT.



3.5 Tehnologii folosite

.NET Core 2.0

.NET Core este o implementare cross-platform a framework-ului .NET produs de Microsoft. Față de versiunile anterioare (până la .NET Framework 4.x), acesta vine cu o serie de noutăți pentru dezvoltatorii de aplicații și utilizatori: este cross-platform, open-source și are un design mult mai modular.

.NET Core are două componente principale: o colecție de biblioteci de bază (Base Class Library) și o mașină virtuală (Common Language Runtime). Mașina virtuală se ocupă cu memory-management, thread-management, securitate la nivel de cod și asigură faptul că software-ul scris în diferite limbaje de programare (C#, F#, VB.NET etc.) - după ce compilează în prealabil în limbaj intermediar (Microsoft Intermediate Language) -, poate fi compilat Just-In-Time pe platforma pe care rulează aplicația. Acest lucru înseamnă că o componentă software scrisă folosind tehnologii .NET Core poate fi rulată pe orice sistem care are instalat runtime environment-ul (există implementări stabile pentru Windows, Linux și Mac OS).

.NET Core vine cu o serie de modele de aplicații pentru care sunt dezvoltate biblioteci specifice: ASP.NET Core (pentru aplicații web și servicii web), Class Library (pentru biblioteci reutilizabile), Console Application (pentru aplicații de consolă) și Universal Windows Platform (pentru aplicații și servicii Windows moderne).

Limbajul standard folosit este C# 7.

C# 7.2

Cel mai răspândit limbaj de programare pentru familia .NET este C#. Acesta este un limbaj care suportă mai multe paradigme de programare: imperativă, declarativă, funcțională și orientată pe obiecte. Este un limbaj strongly-typed, cu suport și pentru obiecte fără un tip predefinit. Suportă tipurile generice (sunt implementate până la nivel de IL), inferența tipurilor (prin keyword-ul *var*) și este proiectat cu suport nativ atât pentru date *reference-type* (clase, interfețe și delegați), cât și pentru date *value-type* (struct, enum).

Deși paradigma de bază este cea orientată pe obiecte, o parte importantă o are programarea funcțională în cadrul limbajului. Standardul C#²⁴ vine cu o serie de construcții foarte utile pentru programatorii care vor să utilizeze această paradigmă în software-ul scris de ei: delegați (echivalentul pointerilor la funcții din C) și event-uri, delegații strongly-typed Action și Func, lambda-expressions, expression-trees și integrarea acestor construcții ca first-class citizens în cadrul limbajului.

În limbaj, există suport nativ pentru programare asincronă.

Entity Framework Core 2.0

Primul pas pentru implementarea unei aplicații data-driven este dezvoltarea tehnologiei de acces la date. Entity Framework este colecția de biblioteci utilizată în acest scop în cadrul familiei de tehnologii .NET.

Modelul arhitectural după care funcționează Entity Framework presupune o legare loosely-coupled între clasele din cod și tabele din baza de date: pentru fiecare tabel disponibil, există un echivalent în cod sub forma unei clase care nu conține altceva decât proprietăți care reprezintă coloanele tabelului respectiv. Legătura dintre tabel și clasă (denumită Entitate sau POCO – Plain Old CLR Object) se face printr-o clasă de mapare unde fiecărei proprietăți din clasă îi este specificată coloana aferentă, iar fiecărei clase îi este specificat tabelul aferent. Entity Framework știe să mapeze indecși, constraint-uri și relații între tabele (acestea din urmă sunt modelate ca proprietăți de navigare – o clasă poate avea ca proprietate o referință către un obiect din altă clasă sau către o colecție de obiecte).

Entity Framework elimină obligația programatorului de a scrie interogări sau de a compune tranzacții pentru baza de date. Folosindu-se un driver specific pentru serverul de baze de date folosit (există suport deja pentru SQL Server, MySQL, Oracle, Postgres, Maria DB, dar programatorii au libertatea să-și implementeze propriul driver), query-urile se construiesc automat astfel: programatorul scrie query-uri folosind LINQ (Language INtegrated Query), query-uri care sunt peste colecții in-memory (liste generice, array-uri, hash set-uri etc.).

²⁴ <https://www.ecma-international.org/publications/standards/Ecma-334.htm>

Serverul de baze de date folosit este configurat și driver-ul instalat, iar în momentul în care query-ul peste obiectele in-memory este concretizat, se generează automat interogări specifice.

ASP.NET Core

Active Server Pages este tehnologia Microsoft pentru realizarea API-urilor și a paginilor Web. În versiunea Core, framework-ul lasă posibilitatea programatorilor să utilizeze două modele de prezentare: fie MVC (pentru aplicații web standard, care nu sunt destinate a fi Single Page Applications), fie Web API (pentru a expune API-uri clienților terți – aplicații JavaScript Single Page Application, clienți mobili, clienți desktop sau chiar alte sisteme software care depind de serviciile expuse).

ASP Core, fiind un model de aplicație construit peste framework-ul .NET Core, este tot cross-platform și open-source. Aplicațiile astfel scrise pot rula pe mai multe sisteme de operare și pot fi livrate în două modalități: *framework-dependent* (în care este necesar runtime environment-ul instalat în prealabil) sau *self-hosted* (în care aplicația instalată la client conține fișiere necesare pentru lansarea aplicației în execuție fără să mai fie nevoie de altceva).

ASP Core se ocupă cu toate elementele ce țin de procesarea request-urilor HTTP: rutarea acestora către Controllere și Acțiuni corespunzătoare, autentificarea și autorizarea (fie prin headere, fie prin cookies – există implementări built-in), binding-ul parametrilor request-ului la parametrii așteptați în cod, validarea modelelor conform regulilor impuse de programator, procesarea efectivă a request-ului. Framework-ul oferă implementări native pentru componente software care folosesc la nivel de aplicație protocolul HTTP.

Modelul Web API este foarte recomandat pentru arhitecturi REST²⁵ (REpresentational State Transfer) pentru a expune servicii web.

²⁵ Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". *Architectural Styles and the Design of Network-based Software Architectures* (Ph.D.). University of California, Irvine.

JavaScript

JavaScript este un limbaj interpretat, dinamic, weakly-typed și de nivel înalt. Suportă mai multe paradigme de programare: imperativă, event-driven, funcțională și orientată pe obiecte (dar nu în sensul clasic, ci bazându-se pe prototipuri). Scopul principal a fost interacțiunea cu DOM-ul²⁶ paginilor web, dar a evoluat și e folosit în toate mediile posibile (aplicații client în browsere, desktop, servere web, jocuri și chiar device-uri IoT).

API-ul nativ nu suportă interacțiuni I/O (lucrul cu fișiere, rețea, acces la periferice), dar mediul în care rulează (mașini virtuale în browser sau peste sistemul de operare) oferă astfel de posibilități. Deși implicit nu software-ul scris în JavaScript rulează într-un singur thread, există suport nativ pentru programare asincronă.

Bootstrap este un framework pentru front-end folosit în principal pentru aplicații web. Conține elemente de HTML și CSS predefinite pentru a ușura design-ul paginilor și pentru a oferi un rezultat uniform și responsive indiferent de rezoluția ecranului clientului. Sunt componente implementate pentru formulare, butoane, elemente de navigare, fonturi și tipografie, liste, input-uri etc.

Universal Windows Platform

UWP este o colecție de API-uri introduse pentru sistemul de operare Windows 10. Acest sistem de operare are versiuni pentru mai multe platforme hardware: PC, Xbox, HoloLens sau orice end-device pe care a fost instalat Windows 10 sau Windows 10 Core, astfel o aplicație scrisă folosind UWP poate rula automat pe oricare din platformele menționate.

API-ul este dezvoltat folosind C++ și există suport de utilizare în C++, C#, F# și JavaScript. Astfel, se poate dezvolta software client-side care să ruleze nativ pe platforme foarte diferite, fără să fie necesară nicio modificare a codului/executabilului.

²⁶ https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

În contextul în care se pot configura senzori complecși care să ruleze pe device-uri mici, dar cu sistem de operare instalat, UWP are o serie de API-uri care ajută la dezvoltarea de aplicații pentru acest tip de sistem: geolocație, GPIO, protocoale de comunicație specifice (de exemplu SPI).

Azure

Azure este platforma cloud de la Microsoft. Aceasta oferă servicii cloud în toate cele 3 forme cunoscute: Software as a Service (pentru servicii ca Outlook, Office 365), Platform as a Service (business analytics, unelte de dezvoltare, sisteme de operare) și Infrastructure as a Service (servere web și de stocare, configurare rețele, mașini virtuale).

Azure IoT Hub este un serviciu utilizat pentru a conecta, monitoriza și administra milioane de device-uri IoT. Se pot stabili canale de comunicație bidirecționale între Hub și device-uri pentru a transmite date și comenzi. Device-urile se conectează independent și securizat la hub, fiecare având identitate și credențiale unice. Din Hub se pot înregistra programatic device-uri noi și se poate administra fiecare în parte, oferindu-se inclusiv posibilitatea realizării de update-uri de software pe device, din cloud. Mesajele primite de Hub de la dispozitivele din teren pot fi rutate după reguli scrise de programatori, către diferite alte servicii: stocare, analiză în timp real, trimitere de notificări etc.

Azure Functions sunt o soluție serverless pentru a rula aplicații doar când este nevoie și fără a fi influențat de mediu sau de interacțiunea cu alte module. Funcțiile astfel scrise sunt bucăți de cod care sunt găzduite individual și care sunt legate la diferite evenimente declanșatoare (un request HTTP, o înregistrare nouă într-un tabel, un mesaj primit de un IoT Hub etc.). Când un eveniment se declanșează, funcția legată la acesta începe să ruleze automat. Avanzatele principale sunt faptul că programatorul nu mai trebuie să se ocupe de infrastructura care leagă logica relevantă (funcția) de evenimentul declanșator și că soluția poate scala automat foarte ușor.

Azure Storage reprezintă o soluție de stocare foarte scalabilă și eficientă. Sunt oferite mai multe modele de stocare, în funcție de necesitățile aplicației: BLOB (pentru date

nestructurate), Queue (pentru cozi de mesaje), Archive (pentru arhivare de date) și Table (pentru baze de date NoSQL, sub formă de stocare key-value pair).

Azure SQL oferă servicii de stocare de date sub formă relaționată. Este un serviciu bazat pe Microsoft SQL Server, complet configurabil: baze de date, utilizatori și grupuri, permisiuni, IP-uri client deschise și acces prin firewall etc. Serviciul se poate utiliza cu unul din cele 3 moduri de auto-replicare și failover automat.

Azure App service permite construirea și host-area aplicațiilor web direct în cloud, fără a mai administra infrastructura adiacentă (mașini virtuale, sisteme de operare, runtime environment sau servere web). Oferă suport atât pentru Linux, cât și pentru Windows, deploy automat din GitHub sau TFS și suportă nativ cele mai folosite tehnologii web: Java, PHP, .NET (atât Framework, cât și Core), Python, Node.js și HTML. Se pot administra domenii, certificate SSL și oferă integrare cu servicii de identitate pentru utilizatorii finali (cum ar fi integrare cu Facebook și Google pentru OAuth²⁷ sau Azure Active Directory).

Toate serviciile menționate sunt construite special pentru a fi scalabile, extensibile și au garanția unui up-time mult mai ridicat decât o soluție on-premises. Faptul că sunt în cloud înseamnă și că dispăre grija infrastructurii, a framework-urilor dependente și a posibilităților reduse de scalare verticală. Serviciile sunt autonome, deci funcționarea defectuoasă a unuia dintre ele nu va opri complet aplicația dezvoltată.

Google Maps

Google Maps, prin platforma on-line, oferă soluții software pentru ridesharing, gaming sau urmărire de bunuri și vehicule. Produsele oferite sunt: hărți (statice sau dinamice, Street View, 360 Views), rute (cu date în timp real despre trafic și optimizare traseu) și locuri (date agregate de la utilizatori pentru mai mult de 100 de milioane de locații din toată lumea).

API-urile puse la dispoziție sunt implementate pentru Android, iOS și JavaScript (suport pentru toate platformele care pot rula cod JavaScript – web, UWP, aplicații desktop sau mobile hibride etc.).

²⁷ <https://oauth.net/>

Raspberry PI

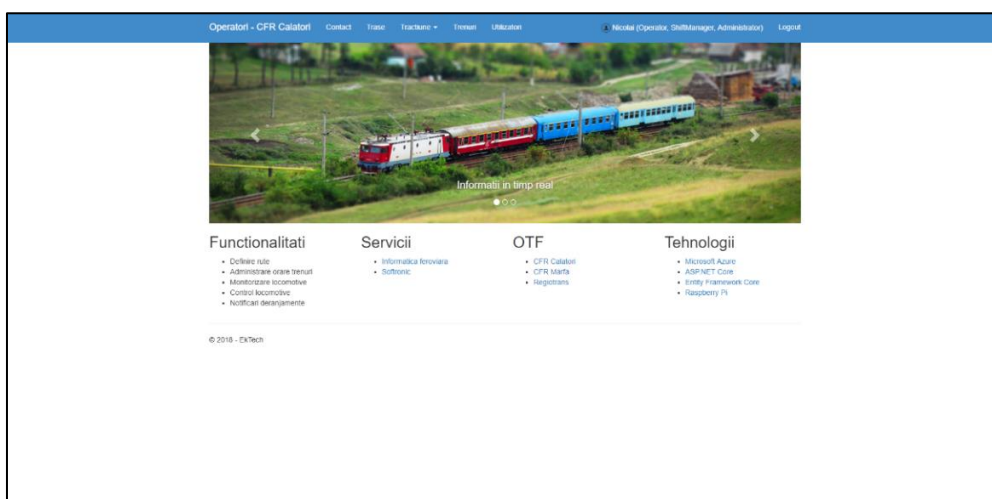
Dispozitivie finale care trimit date ar putea să fie microcontrollere, PLC-uri, dar varianta optimă sunt Raspberry PI. Acestea sunt mini-calculatoare, care au putere de calcul mare (cel mai slab din cele folosite are un procesor ARM Cortex care lucrează la 900MHz, are 1GB RAM, 4 porturi USB, 40 de porturi GPIO pentru senzori/control și port HDMI). În funcție de model se poate conecta la Internet direct prin Wi-Fi sau Ethernet, dar porturile de GPIO pot fi folosite pentru conexiunea la un modul GSM extern.

Senzorii se conectează plug-and-play la device și pot fi proiectați și legați independent la locomotivă față de calculator.

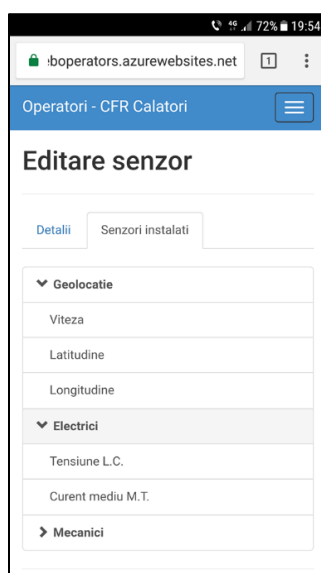
4 EVALUARE

Calitativ, soluția se comportă conform așteptărilor: sunt respectate cerințele funcționale impuse, senzorii transmit date care sunt persistate și urmărite în timp real, Hub-ul IoT din cloud primește mesaje în mod securizat, iar utilizatorii din grupuri diferite au acces separat (administratorii platformei, angajații operatorilor de transport și călătorii).

Aplicațiile pentru clienții finali (utilizatorii menționați anterior) sunt accesibile, sunt securizate, oferă acces bazat pe politici de autorizare și sunt adaptate automat la diverse tipuri de dispozitive. În imaginile de mai jos se poate observa aceeași aplicație pe două dispozitive diferite (laptop cu diagonala de 17” și smartphone cu diagonala de 5.1”)



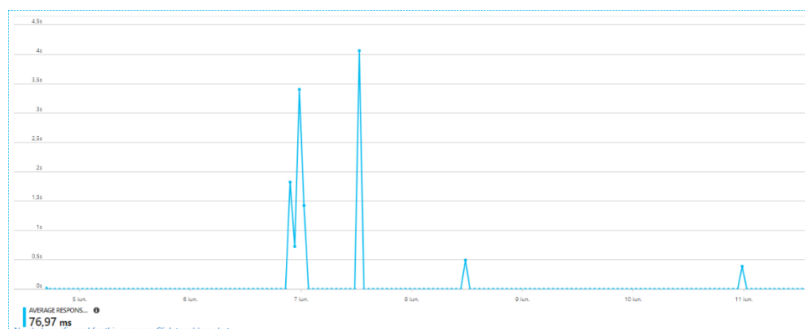
Display de 17”



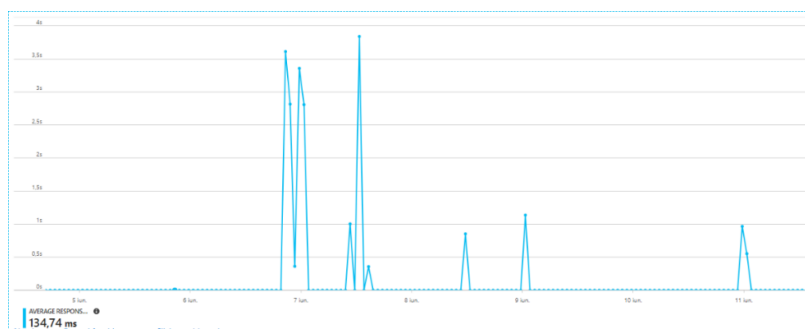
Display de 5.1”

Cantitativ, pentru a măsura performanțele soluției, am apelat la o serie de metrice disponibile în portalul de administrare al platformei cloud de la Azure. Deși nu am testat performanțele simulând o încărcare de producție, metricile oferite de Azure sunt suficient de sugestive pentru a oferi o imagine de ansamblu asupra posibilităților aplicației.

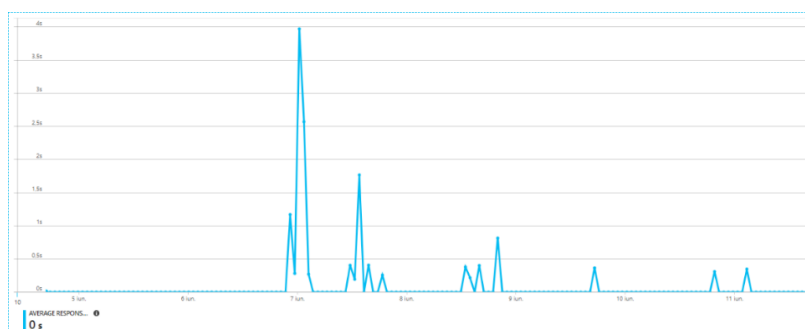
Pentru aplicația web de administrare se observă valori între 0.5 secunde (pentru request-uri de tip GET simple) și 4 secunde (pentru request-uri de tip POST).



Pentru aplicația web pentru operatori se observă valori între 0.5 secunde (pentru request-uri de tip GET simple) și 3.5 secunde (pentru request-uri GET în serie – la vizualizarea datelor în timp real).



Pentru aplicația web pentru călători există un spike de 4 secunde la primul request GET (când se încarcă stațiile și trenurile), dar revine valori de 0.5 – 1 secunde pentru fiecare request GET ulterior, folosit pentru a afișa pozițiile modificate ale trenurilor.



5 CONCLUZII

În această lucrare am tratat una dintre problemele sistemului feroviar – lipsa monitorizării materialului rulant de tracțiune și efectele pe care acest lucru îl produce asupra calității transportului: întârzieri, lipsa viziunii asupra stării tehnice a locomotivelor, utilizarea materialului rulant în mod ineficient și pierderile cauzate de stagnarea în timp a tehnologiilor folosite.

Soluția pe care am dezvoltat-o presupune instalarea unor senzori wireless pe locomotivele operatorilor de transport feroviar care doresc acest lucru. Acești senzori sunt configurabili și pot măsura în timp real orice parametru electric sau mecanic relevant pentru operatorii de transport. În implementarea de față am propus măsurarea coordonatelor spațiale, a vitezei de deplasare și a unor parametri tehnici: curenții motoarelor de tracțiune, tensiunea din linia de contact, presiunile din instalația de frânare.

Întreaga soluție e construită în jurul unei arhitecturi concepute special pentru acest tip de problemă: infrastructură cloud pentru servicii IoT, deosebit de scalabilă și configurabilă, software enterprise loosely-coupled, independent și extensibil, medii de stocare de încredere, alese pentru nevoi specifice (relațional acolo unde este cazul și non-relațional pentru date care se potrivesc principiului celor trei V²⁸).

Tehnologiile folosite sunt stabile, fac parte din aceeași familie și astfel se integrează ușor, sunt open-source, în mare parte, și cross-platform. Alegând soluții cloud (IoT Hub, Serverless Functions, Storage, Service Apps), am rezolvat problemele ce țin de infrastructură și de gestiunea acestora – timpul de dezvoltare este consumat în principal doar pentru logica de rezolvare a problemei, nu și pentru găzduirea soluției.

În final, prin arhitectura propusă și serviciile implementate am atins obiectivele dorite: monitorizarea în timp real a materialului rulant printr-o serie de aplicații ușor de folosit, accesibile rapid și securizat, extensibile la cerințe ulterioare și scalabile pentru solicitări de producție. Costul total este mic și total justificat de serviciile oferite.

²⁸ <https://www.gartner.com/it-glossary/big-data>

6 DEZVOLTĂRI VIITOARE

Cunoscând domeniul feroviar din România și tendințele la nivel european, pot găsi o serie de funcționalități care vor trebui implementate în viitor. Senzorii sunt momentan concepuți pentru a măsura date de telemetrie de la materialul rulant de tracțiune (locomotive și automotore). În viitor putem implementa acest tip de dispozitive și în vagoanele de călători (pentru a măsura parametri relevanți pentru confortul acestora – temperatură, umiditate, calitatea aerului ventilat, numărul de consumatori electrici din vagon etc.) sau în puncte critice ale infrastructurii (joncțiuni cu probleme, treceri periculoase la nivel cu calea ferată, poduri și podețe slăbite, stații cu flux mare de călători).

Dispozitivele IoT proiectate în această versiune sunt relativ ne-interactive cu personalul de locomotivă. În prezent, când începe o călătorie, un mecanic își completează datele specifice lui și trenului remorcat într-o foaie („Foaie de parcurs”) pe care o primește și o predă în stațiile în care preia și predă trenul. Acest proces se poate automatiza și digitaliza direct prin infrastructura IoT de pe locomotive, doar prin extinderea dispozitivului cu un ecran tactil. Dispozitivele rulează deja Windows 10 Core, deci implementarea va fi facilă.

Semnalizarea unor probleme observate în timpul mersului se face momentan prin completarea unui formular („Ordin de avizare”) și predarea acestuia la prima stație cu oprire. Având în vedere dimensiunea și calitatea infrastructurii feroviare române, este utilă o modalitate prin care să se poată semnala automat (similar Waze²⁹) defectele remarcate. La fel, soluția existentă se poate extinde să suporte și această funcționalitate.

Momentan, personalul angajat operatorilor de transport, prin soluția propusă și implementată, are acces la date despre materialul rulant în timp real. Un business care dorește să evolueze va cere, cel mai probabil, și posibilitatea analizei datelor din trecut: filtrate după zile, trenuri, personal responsabil sau vizibile sub forma unor servicii de business analytics. Datele de telemetrie fiind deja persistate, dezvoltarea unui modul de analiză va urma natural.

Din perspectiva tehnică a soluției, cele mai importante dezvoltări ar fi: refactorizarea administrării senzorilor într-o componentă mai complexă (care să gestioneze fiecare tip de

²⁹ <https://www.waze.com/ro/>

senzor de pe un anumit device) și sistem de realizare de update-uri remote pentru dispozitive.

7 BIBLIOGRAFIE

- [1] Microsoft Azure Documentation, „Azure Developer Guide,”, 2018. [Interactiv]. Available: <https://docs.microsoft.com/en-us/azure/guides/developer/azure-developer-guide>. [Accesat 2018].
- [2] Microsoft Azure Documentation, „IoT Hub,” 2018. [Interactiv]. Available: <https://docs.microsoft.com/en-us/azure/iot-hub/>.
- [3] Microsoft Azure Documentation, „Azure Application Architecture Guide,” 2018. [Interactiv]. Available: <https://docs.microsoft.com/en-us/azure/architecture/guide>. [Accesat 2018].
- [4] E. Evans, „Domain-Driven Design: Tackling Complexity in the Heart of Software,” Addison-Wesley, 2012.
- [5] J. Albahari și B. Albahari, C# 6.0 in a Nutshell, 6th Edition, O'Reilly, 2015.
- [6] A. Hunt și D. Thomas, The Pragmatic Programmer: From Journeyman to Master, Addison Wesley, 1999.
- [7] D. Crockford, JavaScript: The Good Parts, Yahoo Press, 2008.
- [8] R. R. Singh, Mastering Entity Framework, Packt Publishing, 2015.
- [9] A. Freeman, Pro ASP.NET Core MVC 2, Apress, 2017.
- [10] M. Fowler, Patterns of Enterprise Application Architecture, Pearson Education Inc., 2003.
- [11] Ministerul Transporturilor, Construcțiilor și Turismului, *Regulementul de Remorcare și Frânare - Numărul 006*, Editura Feroviară, 2006.
- [12] Ministerul Transporturilor, Construcțiilor și Turismului, *Regulamentul de semnalizare - Numărul 004*, Editura Feroviară, 2006.

- [13] Autoritatea Feroviară Română, *LE 060 EA 5100KW - Îndrumător de exploatare*, Editura ASAB, 2007.
- [14] Autoritatea Feroviară Română, *LE 060 EA 5100KW - Planşe*, Editura ASAB, 2007.
- [15] Google Maps Platform, 2018. [Interactiv]. Available: <https://cloud.google.com/maps-platform/>. [Accesat 2018].
- [16] J. Skeet, *C# in depth*, Manning Publications, 2008.
- [17] C. Calvert şi K. Dinesh, *Essential LINQ*, Addison-Wesley Professional, 2009.
- [18] B. Alan, *Learning SQL*, O'Reilly Media, 2009.
- [19] A. Molinaro, *SQL Cookbook*, O'Reilly Media, 2009.
- [20] M. Pilgrim, *HTML5: Up and Running*, O'Reilly Media, 2010.
- [21] E. Meyer şi E. Weyl, *CSS: The Definitive Guide, 4th Edition*, O'Reilly Media, 2017.

8 ANEXE

1. Controllere de bază ASP MVC Core

```
public class BaseController : Controller
{
    protected IMapper mapper;

    public BaseController(IMapper mapper)
    {
        this.mapper = mapper;
    }

    public override JsonResult Json(object data)
    {
        return base.Json(data, new JsonSerializerSettings());
    }

    protected UnprocessableEntityObjectResult Unprocessable()
    {
        return new UnprocessableEntityObjectResult(ModelState.GetErrors());
    }

    protected UnprocessableEntityObjectResult Unprocessable(string errorMessage)
    {
        return new UnprocessableEntityObjectResult(errorMessage);
    }

    protected ViewResult NotFoundView()
    {
        return View("NotFound");
    }

    protected ViewResult ForbidView()
    {
        return View("Forbid");
    }
}

public class AuthorizedController : BaseController
{
    protected readonly CurrentUser currentUser;
    protected readonly CurrentOperator currentOperator;
    protected readonly IAuthorizationService authorizationService;

    public AuthorizedController(
        CurrentUser currentUser,
        CurrentOperator currentOperator,
        IAuthorizationService authorizationService,
        IMapper mapper)
        : base(mapper)
    {
        this.currentUser = currentUser;
        this.currentOperator = currentOperator;
        this.authorizationService = authorizationService;
    }
}
```


2. Modele pentru o resursă ASP MVC Core

Model pentru introducere resursă nouă:

```
public class RouteCreateVm
{
    public int TrainRankId { get; set; }
    public List<SelectListItem> TrainRanks { get; set; }
    public string Number { get; set; }
    public int MaximumTrainLength { get; set; }
    public int MaximumTrainWeight { get; set; }
    public int BreakingPercentage { get; set; }
    public bool IsActive { get; set; }
}
```

Model pentru editare resursă existentă:

```
public class RouteUpdateVm
{
    public int Id { get; set; }
    public int TrainRankId { get; set; }
    public List<SelectListItem> TrainRanks { get; set; }
    public string Number { get; set; }
    public int MaximumTrainLength { get; set; }
    public int MaximumTrainWeight { get; set; }
    public int BreakingPercentage { get; set; }
    public bool IsActive { get; set; }
}
```

Model pentru afișat resurse sub formă de detalii (în cadrul unei liste)

```
public class RouteVm
{
    public int Id { get; set; }
    public string Number { get; set; }
    public bool IsActive { get; set; }
}
```

3. Validator custom pentru model de date ASP MVC Core

```
public class RouteCreateValidator : AbstractValidator<RouteCreateVm>
{
    private readonly RouteService routeService;

    public RouteCreateValidator(RouteService routeService)
    {
        this.routeService = routeService;

        RuleFor(x => x.TrainRankId)
            .NotNull()
            .WithMessage("Campul este obligatoriu")
            .NotEmpty()
            .WithMessage("Campul este obligatoriu");
        RuleFor(x => x.Number)
            .NotNull()
            .WithMessage("Campul este obligatoriu")
            .NotEmpty()
            .WithMessage("Campul este obligatoriu")
            .Must(number => !routeService.Exists(number))
            .WithMessage(n => $"Campul {n.Number} exista deja!")
            .MaxLength(100)
            .WithMessage("Campul trebuie sa aiba maxim 100 de caractere!");
        RuleFor(x => x.MaximumTrainLength)
            .GreaterThan(0)
            .WithMessage("Lungimea trebuie sa fie mai mare ca zero");
        RuleFor(x => x.MaximumTrainWeight)
            .GreaterThan(0)
            .WithMessage("Tonajul trebuie sa fie mai mare ca zero");
        RuleFor(x => x.BreakingPercentage)
            .GreaterThan(0)
            .WithMessage("Procentajul de franare trebuie sa fie mai mare ca zero");
        RuleFor(x => x.IsActive)
            .NotNull()
            .WithMessage("Campul este obligatoriu")
            .NotEmpty()
            .WithMessage("Campul este obligatoriu");
    }
}
```

4. Mapare de la model ASP MVC Core la Entitate (agnostică de biblioteci)

```
public class RouteProfile : Profile
{
    public RouteProfile()
    {
        CreateMap<Route, RouteVm>();

        CreateMap<RouteCreateVm, Route>();

        CreateMap<Route, RouteUpdateVm>();
        CreateMap<RouteUpdateVm, Route>();
    }
}
```

5. Configurare servicii pentru aplicația ASP MVC Core

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();
    services.AddSingleton<IActionContextAccessor, ActionContextAccessor>();
    services.AddSingleton<AppSettings>();

    services.AddDbContext<ThomasContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
    services.AddMvc(config =>
    {
        config.ReturnHttpNotAcceptable = true;
        config.OutputFormatters.Add(new XmlDataContractSerializerOutputFormatter());
        config.InputFormatters.Add(new XmlDataContractSerializerInputFormatter());
    })
        .AddFluentValidation(cfg =>
            cfg.RegisterValidatorsFromAssemblyContaining<CommonValidator>());

    // Add Api Services
    services.AddAzureIntegration();
    services.AddDerivedFrom<IService>(
        services.AddScoped,
        Assembly.GetExecutingAssembly());
    services.AddScoped<IThomasUnitOfWork, ThomasUnitOfWork>();

    //Add Mappers
    services.AddAutoMappers();

    services.AddAuthentication(Code.Security.Cookies.Constants.AuthenticationSchemeName)
        .AddCookie(Code.Security.Cookies.Constants.AuthenticationSchemeName,
            options =>
            {
                options.AccessDeniedPath = new PathString("/Account/Login");
                options.LoginPath = new PathString("/Account/Login");
            });
    services.AddAuthorizationHandlers();
    services.AddCurrentUser();
    services.AddCurrentOperator();
    services.AddApplicationSettings(Configuration);

    services.AddKendo();

    services.AddRouteJs();
}
```

6. Servicii în logica de business

```
public class CrudService : BaseService, IDisposable
{
    protected IThomasUnitOfWork UnitOfWork;

    public CrudService(IThomasUnitOfWork unitOfWork, ILogger logger = null)
        : base(logger)
    {
        UnitOfWork = unitOfWork;
        Logger = logger;
    }

    public void Dispose(bool disposing)
    {
        if (disposing)
        {
            if (UnitOfWork != null)
            {
                UnitOfWork.Dispose();
                UnitOfWork = null;
            }
        }
    }

    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }

    protected int Transaction(Action<IThomasUnitOfWork> predicate)
    {
        int savedState = 0;

        try
        {
            savedState = UnitOfWork.ExecuteInTransaction(() => predicate(UnitOfWork));
        }
        catch (Exception ex)
        {
            Logger.LogError(ex.Message);
        }

        return savedState;
    }

    protected T Query<T>(Func<IThomasUnitOfWork, T> predicate)
    {
        return predicate(UnitOfWork);
    }
}
```

7. Interfețe pentru accesul la baza de date

Metode pentru fiecare Repository:

```
public interface IRepository<TEntity> : IRepository
    where TEntity : IEntity
{
    IQueryable<TEntity> Query { get; }
    void Add(TEntity entity);
    void AddRange(IEnumerable<TEntity> entities);
    void Update(TEntity entity);
    void Remove(TEntity entity);
    void RemoveRange(IEnumerable<TEntity> entities);
}

public interface IRepository
{
}
```

Metode pentru UnitOfWork:

```
public interface IUnitOfWork : IDisposable
{
    int ExecuteInTransaction(Action action);
}
```

Entități cu care aplicația trebuie să poată lucra:

```
public interface IThomasUnitOfWork : IUnitOfWork
{
    IRepository<Administrator> Administrator { get; }
    IRepository<Device> Device { get; }
    IRepository<EngineCategory> EngineCategory { get; }
    IRepository<Engine> Engine { get; }
    IRepository<EngineType> EngineType { get; }
    IRepository<Operator> Operator { get; }
    IRepository<Permission> Permission { get; }
    IRepository<Role> Role { get; }
    IRepository<RolePermission> RolePermission { get; }
    IRepository<Route> Route { get; }
    IRepository<Station> Station { get; }
    IRepository<Timetable> Timetable { get; }
    IRepository<Train> Train { get; }
    IRepository<TrainEngine> TrainEngine { get; }
    IRepository<TrainRank> TrainRank { get; }
    IRepository<User> User { get; }
    IRepository<UserRole> UserRole { get; }
}
```

8. Configurări mapare pentru o resursă Entity Framework

```
public class RouteMap : IEntityTypeConfiguration<Route>
{
    public void Configure(EntityTypeBuilder<Route> builder)
    {
        builder.ToTable("Routes");

        builder.Property(e => e.Number)
            .IsRequired()
            .HasColumnType("nchar(100)");

        builder.Property(e => e.MaximumTrainLength)
            .IsRequired();
        builder.Property(e => e.MaximumTrainWeight)
            .IsRequired();
        builder.Property(e => e.BreakingPercentage)
            .IsRequired();
        builder.Property(e => e.IsActive)
            .IsRequired();

        builder.HasOne(d => d.Operator)
            .WithMany(p => p.Routes)
            .HasForeignKey(d => d.OperatorId)
            .OnDelete(DeleteBehavior.ClientSetNull)
            .HasConstraintName("FK_Routes_Operators");
    }
}
```

9. Utilitar pentru interogarea Azure Table Storage:

```
public class AzureTableStorageService : IService
{
    private readonly string connectionString;
    private readonly CloudTableClient tableClient;

    public AzureTableStorageService(AppSettings appSettings)
    {
        this.connectionString = appSettings.TableStorageConnectionString;
        var storageAccount = CloudStorageAccount.Parse(connectionString);
        tableClient = storageAccount.CreateCloudTableClient();
    }

    public TelemetryData Get(string deviceId)
    {
        TableContinuationToken continuationToken = null;
        var table = tableClient.GetTableReference(Tables.TelemetryData);

        var query = new TableQuery<TelemetryDataEntity>()
            .Where(TableQuery.GenerateFilterCondition(
                PropertyNames.PartitionKey,
                QueryComparisons.Equal, deviceId))
            .Take(1);

        var result = table.ExecuteQuerySegmentedAsync(query, continuationToken)
            .Result
            .FirstOrDefault();

        return result?.Extract() ?? new TelemetryData();
    }
}
```


10. Utilitar pentru interogarea device-urilor IoT prin IoT Hub

```
public class AzureDeviceService : IService
{
    private readonly RegistryManager registryManager;
    private readonly ServiceClient client;

    private readonly string connectionString;

    public AzureDeviceService(AppSettings appSettings)
    {
        this.connectionString = appSettings.IotHubConnectionString;

        registryManager = RegistryManager
            .CreateFromConnectionString(connectionString);
        client = ServiceClient.CreateFromConnectionString(connectionString);
    }

    public async Task<Twin> Query(string deviceId)
    {
        return await registryManager.GetTwinAsync(deviceId);
    }

    public async Task<bool> IsAvailable(string deviceId)
    {
        var twin = await Query(deviceId);

        return twin.ConnectionState.HasValue &&
            twin.ConnectionState.Value == DeviceConnectionState.Connected;
    }

    public async Task<CloudToDeviceMethodResult> StartReboot(
        string deviceId,
        int secondsTimeout = 30)
    {
        var method = new CloudToDeviceMethod("reboot")
        {
            ResponseTimeout = TimeSpan.FromSeconds(secondsTimeout)
        };

        return await client.InvokeDeviceMethodAsync(deviceId, method);
    }
}
```

11. Serviciu de trimitere mesaje de pe device spre Hub

```
public sealed class CloudMessageService
{
    private readonly DeviceClient deviceClient;

    private readonly string deviceId;
    private readonly string deviceKey;
    private readonly string iotHubUri;

    public CloudMessageService(string deviceId, string deviceKey, string iotHubUri)
    {
        this.deviceId = deviceId;
        this.deviceKey = deviceKey;
        this.iotHubUri = iotHubUri;

        deviceClient = DeviceClient.Create(
            iotHubUri,
            new DeviceAuthenticationWithRegistrySymmetricKey(deviceId, deviceKey),
            TransportType.Mqtt);
    }

    public void SendStartMessage()
    {
        var telemetry = new TelemetryData { MessageType = Messages.Start };
        SendMessage(telemetry, Routes.Storage);
    }

    public void SendStopMessage()
    {
        var telemetry = new TelemetryData { MessageType = Messages.Stop };
        SendMessage(telemetry, Routes.Storage);
    }

    public void SendTelemetryData(object telemetryData)
    {
        SendMessage(telemetryData, Routes.Storage);
    }

    public void SendCriticalEvent(object sender, SensorEventArgs args)
    {
    }

    private void SendMessage(object data, string route)
    {
        var dataString = JsonConvert.SerializeObject(data);
        var message = new Message(Encoding.ASCII.GetBytes(dataString));

        message.Properties["Route"] = route;

        deviceClient.SendEventAsync(message);
    }
}
```

12. Background Worker pentru device-ul IoT

```
public sealed class MainWorker
{
    private readonly CloudMessageService cloudMessageService;
    private readonly SensorsService sensorsService;

    private readonly Timer loop;

    public MainWorker()
    {
        cloudMessageService = new CloudMessageService(
            AppSettings.DeviceId,
            AppSettings.DeviceKey,
            AppSettings.IoTHubUri);

        sensorsService = new SensorsService();
        sensorsService.CriticalAction += cloudMessageService.SendCriticalEvent;

        loop = new Timer
        {
            Interval = 1000 * AppSettings.SecondsBetweenRequests,
        };
        loop.Elapsed += new ElapsedEventHandler(SendTelemetry);
    }

    public void Start()
    {
        cloudMessageService.SendStartMessage();
        loop.Start();
    }

    public void Stop()
    {
        cloudMessageService.SendStopMessage();
        loop.Stop();

        sensorsService.Dispose();
    }

    private void SendTelemetry(object source, ElapsedEventArgs args)
    {
        try
        {
            var telemetryData = sensorsService.ReadTelemetry();
            cloudMessageService.SendTelemetryData(telemetryData);
        }
        catch (Exception e)
        {
            Debug.WriteLine("Exception while sending device meta data :\n" + e.ToString());
        }
    }
}
```