

## **Abstract**

Mai mult de jumătate din populația Globului trăiește, în prezent la oraș. Viața în cadrul acestuia, din an în an, a devenit tot mai încărcată și, prin urmare, epuizantă. Practic trăim la serviciu, facultate ori în cadrul altor activități, care ne cer atenția. Într-un astfel de tempou grăbit nu mai avem deloc timp pentru noi, pur și simplu suntem lipsiți de acesta. În cadrul orașului (orașul fiind focusul lucrării date) sunt o mulțime de locuri, unde oamenii pot duce o viață socială cât și profesională complexă. Problema este, că nu avem timp să ținem cont de toate evenimentele la care am putea merge în timpul liber, pentru a petrece frumos alături de cei dragi. Cate evenimente interesante pierdem pentru că nu știm de existența lor la timpul potrivit? Lipsa odihnei spirituale și fizice duce la un stres psihologic destul de mare, progresând într-o problema reală, de aici a pornit și ideea lucrării. O cauză a problemei este lipsa de timp pentru planificarea odihnei. Pasul cu care să avansăm este unul accelerat și chiar alarmant astfel încât transformarea spațiului urban într-un mediul sănătos să devină o prioritate majoră pentru mine ca individ social. Venind din societate, vreau să întorc acesteia, drept semn de mulțumire, un efort util și productiv în scopul comunității. Drept soluție la problema dată vine aplicația "*TOWARDS A SMART CITY*" care face acest lucru pentru tine, și anume monitorizează evenimentele din cadrul orașului în care te află, urmărind nivelul de zgomot pentru fiecare zonă a acestuia cât și starea curentă și viitoare a vremii, pentru a putea oferi o experiență utilă și placută utilizatorului.

# Cuprins

<b>1</b>	<b>Introducere</b>	<b>5</b>
<b>2</b>	<b>Imagine de Ansamblu</b>	<b>7</b>
<b>3</b>	<b>Aplicații similare</b>	<b>9</b>
<b>4</b>	<b>Tehnologii utilizate</b>	<b>11</b>
4.1	Ionic . . . . .	11
4.2	Redis . . . . .	12
4.3	Azure . . . . .	12
4.3.1	MongoDB Azure Service . . . . .	12
4.4	AWS EC2 . . . . .	13
4.5	Military Grid Reference System . . . . .	14
<b>5</b>	<b>Arhitectura aplicației</b>	<b>17</b>
5.1	Cerințele aplicației . . . . .	17
5.2	Arhitectura generală . . . . .	18
5.3	Client . . . . .	19
5.4	Server . . . . .	20
5.5	Event API . . . . .	22
<b>6</b>	<b>Implementare</b>	<b>24</b>
6.1	Client . . . . .	24
6.2	Server . . . . .	29
6.3	Event API . . . . .	33
<b>7</b>	<b>Use Case</b>	<b>36</b>
<b>8</b>	<b>Concluzie</b>	<b>39</b>
<b>9</b>	<b>Bibliografie</b>	<b>40</b>
	<b>Appendices</b>	<b>42</b>
	<b>Anexa A Setarea mediului de lucru</b>	<b>42</b>
A.1	Instalare <i>Node Package Manager</i> . . . . .	42

A.1.1 Petru utilizatorii de Unix . . . . .	42
A.1.2 Pentru utilizatorii de Windows . . . . .	42
A.2 Android SDK . . . . .	42
A.2.1 Configurarea pe Ubuntu . . . . .	42
A.2.2 Configurarea pe Windows . . . . .	43
<b>Anexa B Configurare Ionic</b>	<b>44</b>
B.1 Configurări preleminare . . . . .	44
B.2 Instalam ultima versiune de cordova și ionic cu ajutorul <i>npm</i> . . . . .	44
<b>Anexa C Plugin-uri pentru Ionic</b>	<b>44</b>
C.1 DB Meter . . . . .	44
C.2 Geolocation . . . . .	45
<b>Anexa D Rularea Aplicatiei</b>	<b>45</b>

# 1 Introducere

În ultimul deceniu populația în mediul urban a crescut de aproximativ trei ori. Estimările procentuale pentru viitorul apropiat arată circa 66% pentru anul 2050. Creșterea aceasta vine cu noi provocări pentru infrastructura orașului. Cetățenii acestuia trebuie asigurați cu resurse necesare de calitate precum apă curată, mâncare sănătoasă și nu, în ultimul rand, menținerea unui mecanism economic și social active care devine o dificultate, soluția evidentă este orașul intelligent. [1]

Conceptul de oraș intelligent a devenit în ultimii ani un subiect de interes sporit atât pentru zona academică cât și pentru cea politică. Dar ce, mai exact transformă un oraș simplu în unul intelligent?

În cele ce urmează vom discuta conceptul de "smart" în cadrul unui oraș și cum putem folosi tehnologia pentru a crea un mediu social confortabil. Un oraș Smart (intelligent) înseamnă un oraș mai inclusiv, care creează oportunități egale pentru toți. Tehnologia este cuvântul cheie care stă la baza întregului concept aceasta nu este în mod necesar un lux, ci dimpotrivă s-a dovedit să ne simplifice existența. Ea implică tot ceea ce înseamnă procesul de comunicare dintre cetățean și mediul de afaceri, oportunități ce se traduc prin sporirea calității vieții, cu alte cuvinte, o interacțiune mai prietenoasă dintre oraș și cetățean prin intermediul unei aplicații. Scopul de bază merge dincolo de aceasta, nu se rezumă totul la interacțiunea dintre servicii oferite și consumarea acestora. Scopul fiind crearea unui loc mai atractiv pentru a locui, munci și recrea.

Orașul nu este un loc pe care doar îl vizitezi (acesta fiind scopul unor aplicații turistice cum ar fi *TripAdvisor*, *CityMapper*) ci este un loc unde trăiești.

Proiectul *Towards a smart city* vine cu o soluție care are același focus **Oraș, viață echilibrată**, dar spre deosebire de aplicațiile menționate anterior aceasta aduce o soluție de nivel abstract în ceea ce privește: **[locație, local, eveniment]**.

Uneori vrem să știm mai mult decât cel mai scurt drum până la un punct pe hartă ori care stele sunt un local bazându-ne pe note primite în trecut. Ar fi comod de exemplu dacă am cunoaște care sunt caracteristicile mediului în care dorim să mergem: nivelul de zgomot, temperatura, umiditatea, și alte atrăgătoare. Ne-ar plăcea dacă ne-am auzi unul pe altul în timp ce vorbim în cafenea ori dacă am ști că putem merge liniștită cu copilul la o plimbare în parc.

Despre crearea unui ecosistem intelligent unitar vom vorbi în Capitolul 2, în acesta de asemenea vom analiza mai profundat problema creării unui astfel de sistem de monitorizare. vom vedea soluții existente și cele posibile. **Ex:** cum aflăm nivelul de zgomot într-o anumita zonă,

care este starea vremii din zona respectiva. Aplicații similare ce ne oferă acestea și cu ce venim în plus.

În capitolul 3 voi prezenta tehnologiile utilizate în cadrul dezvoltării acestui proiect. *Angular 2*, *Ionic 2* pentru realizarea aplicației client, *AndroidSDK* pentru generarea apk-ului(Android Package Kit) în scopul distribuirii aplicației mobile. Redis și MongoDB drept soluție de *storage*. AWS și Azure furnizorii de cloud la care apelez pentru a găzdui componente ale aplicației.

Capitolul 4 descrie soluția arhitecturală a lucrării de față. Arhitectura generală a aplicației cat și conexiunea între componente. Si bineînțeles voi analiza fiecare componentă în detaliu.

Cu implementări propriu zise vom face cunoștință în Capitolul 5. Iar capitolul ce urmează conține o imagine de ansamblu asupra Softului dezvoltat cat și exemple de utilizare ale acestuia.

## 2 Imagine de Ansamblu

Un oraș intelligent este definit de capabilitatea de utilizare a tehnologiei în scopul creării unei infrastructuri complexe care este ușor de manevrat. De obicei este vorba despre utilizarea datelor obținute de la senzori pentru a crea mijloace automatizate de control asupra anumitor activități. De exemplu managementul semafoarelor prin intermediul wireless care aduce un cost redus și prin acest fapt are loc o micșorare de cheltuieli –x suma într-un anumit interval de timp. [2]

Un exemplu mai elocvent în cadrul lucrării respective ar fi monitorizarea nivelului de poluare a aerului (care ar ajuta persoanelor ce suferă de maladii respiratorii : ex. Astm bronsic), urmărirea nivelului de zgomot precum și starea vremii pentru a putea oferi cetățenilor (sau altor sisteme) informație din care să poată să ia decizii personalizate.

O aplicabilitate foarte bună este utilizarea senzorilor video pentru întreținerea ordinii publice. Ori existența anumitor senzori care să îți ofere informație privind numărul de locuri libere din parcare, care să permită declansarea unui sistem de sugestie către aplicația șoferilor ce ar indica calea către locul de parcare.

În cele din urma în construcția unui Oraș Intelligent identificăm trei cuvinte cheie:

1. Senzori
2. Conexiune
3. Comunicare

În timp ce procesul de urbanizare a sporit dramatic, inevitabilă a fost realizarea în practică a conceptului de *Smart City*. **Andhra Pradesh** din India este una din primele state care a pus în practică utilizarea tehnologiilor IoT pentru sporirea nivelului de trai nu doar în orașe ci și în cadrul mediului rural, drept încercare de a încetini procesul de urbanizare. Cu alte cuvinte nu este relevant dacă este oraș sau sat, se face abstractizare de așezarea spațială. Ideea în sine este de a monitoriza și spori capacitatele unui mediu într-un mod cat se poate de intelligent utilizând în realizarea acestui scop tehnologia IoT. Orașul intelligent este manifestarea acestei tehnologii ce implica utilizarea acumulatorilor (**senzori**) de date fie dedicati, ori integrati în cadrul altor dispozitive (senzorii integrati în dispozitivele mobile) pentru generarea de date care vor fi comunicate pe baza unei **conexiuni** la un centru de procesare. Care la rândul sau are grijă să transforme acele *raw records* în date procesate care pot fi privite ca niște metrii pe baza cărora se vor lua decizii pentru îmbunătățirea activităților în cadrul orașului. Putem merge mai departe pana la stocarea unui istoric de metrii astfel vom avea la indemâna un instrument în plus pe care îl vom putea folosi în comiterea deciziilor. [3]

Ca să realizăm acest flux: [colectarea-> comunicarea datelor-> procesarea acestora-> luarea deciziilor] avem nevoie de un sistem omogen (ceea ce este cam imposibil) de aici apare și necesitatea de **Comunicare** a componentelor (senzori, api-uri, servere de procesare)

Comunicarea este cheia principală în dezvoltarea tehnologiei IoT, potrivit raportului oferit de instituția McKinsey avem următoarele concluzii

Comunicarea dintre sisteme IoT este componenta crucială care deseori lipsește ori este prost realizată din cauza concurenței economice. Pentru a realiza întreg potențialul economic, asamblarea sistemelor IoT se necesita asigurarea caracteristicii comunicante circa 40% ba chiar 60% pentru anumite setări. De aceea se promovează crearea interfețelor deschise ori via API, pentru a permite creșterea progresiva a acestui sistem. [4]

Urbanizarea prin intermediul tehnologiei IoT. IoT, fundamental, pe baza căruia se construiește întregul sistem intelligent în cadrul orașului. Acest sistem trebuie să respecte câteva cerințe minime pentru a putea purta numele de intelligent: transport intelligent, management de energie și apă intelligent. o infrastructură intelligentă, spitale inteligente.Pivotul acestui ecosistem după cum observăm din figura 1 bineînțeles ca trebuie să fie unul bine pus la punct. [5]

În cadrul lucrării respective am reușit să creez un sistem de monitorizare la nivelul de zgomot care reușește să colecteze informație privind zgomotul pentru fiecare zonă de tip mgrs cu precizia de nivel 3 ( pătrate de lungime 100 metri mai detaliat despre aceasta în Capitolul 4 . 6) drept metrică fiind decibelul, folosind senzorii din dispozitivul mobil în care este instalată aplicația. Astfel oferind prin intermediul unui API acces la aceste date(comunicare).

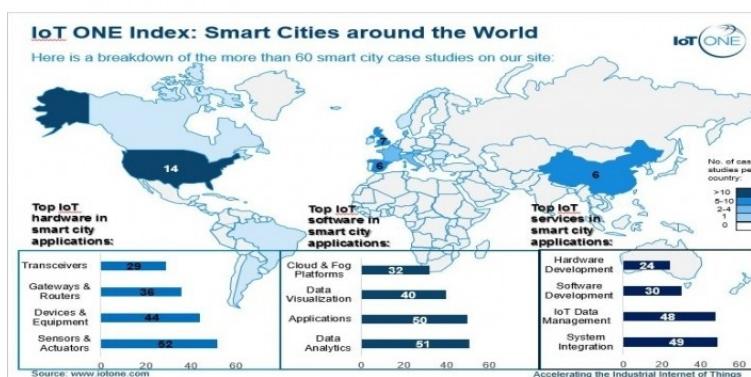


Figura 1: Numarul de cazuri de studiu în privința orașului intelligent grupate regiuni cat și nivelul de accelerare a tehnologiilor IoT în cadrul acestora.[6]

### 3 Aplicații similare

După un timp de cercetare al domeniului Smart City și tehnologiilor IoT am ajuns la cunoștință cu platforma *Smart Citizen*. Obiectivul principal al acesteia este da conectează datele, cunoștințele și oamenii. Pentru asigurarea acestui lucru a fost construit un sistem de monitorizare a indicatorilor mediului și livrarea acestora către utilizator prin intermediul unei aplicații web și mobile în forma unei hărți pe care putem vizualiza dispozitivele și desigur că putem vedea și ultimele date înregistrate de acestea[7]. Modulele acestei platforme pot fi urmărite în figura 2

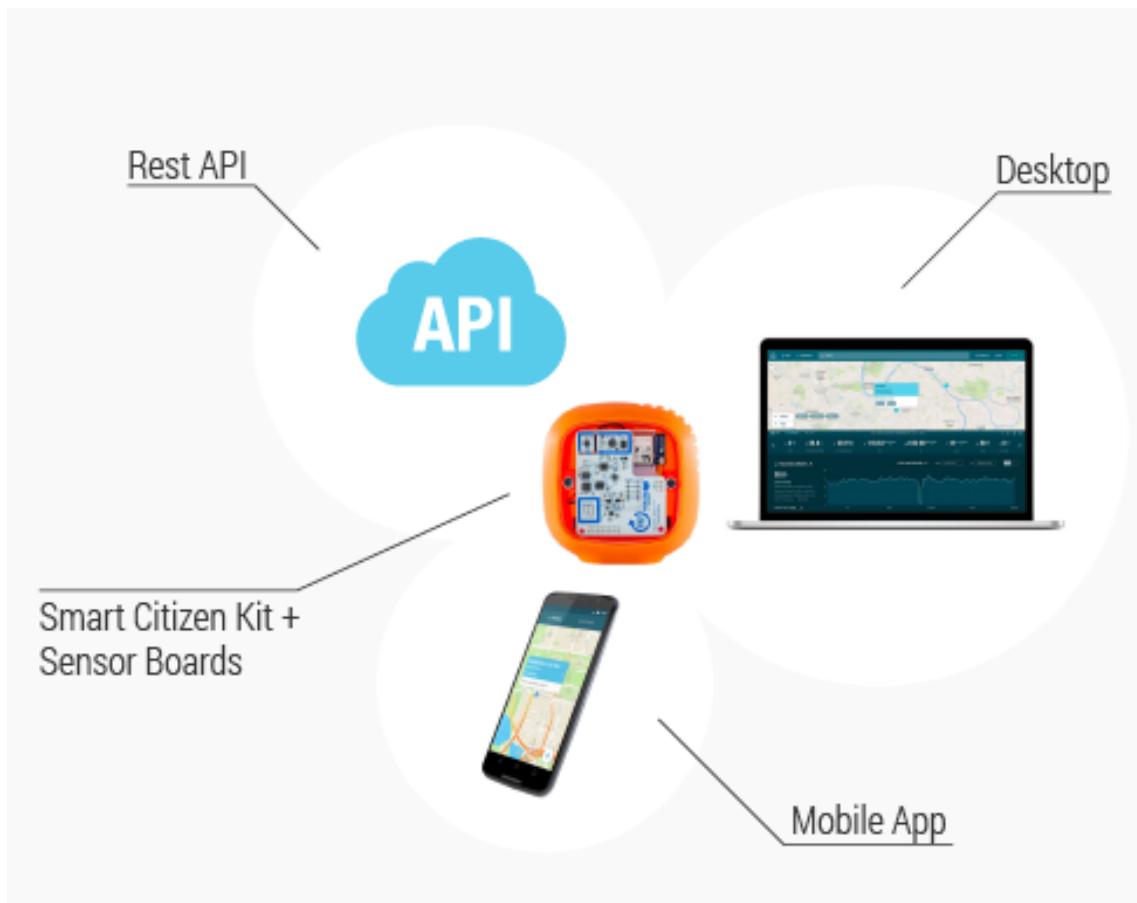


Figura 2: Diagrama generică a platformei SmartCitizen [7]

Necesitarea de apariție a acesteia a fost creșterea nemulțumirii orășenilor referitor la calitatea aerului din oraș. Drept nucleu al acestei platformă este "*Smart Citizen Kit*" acest kit poate fi văzut în figura 3 care este un dispozitiv ce realizează metrii asupra calității aerului. Metricile măsurate sunt:

1. Umiditatea aerului;
2. Temperatura;
3. Concentrația de  $CO$  și  $NO_2$ ;

4. Intensitatea luminii.

5. Nivelul de zgomot.

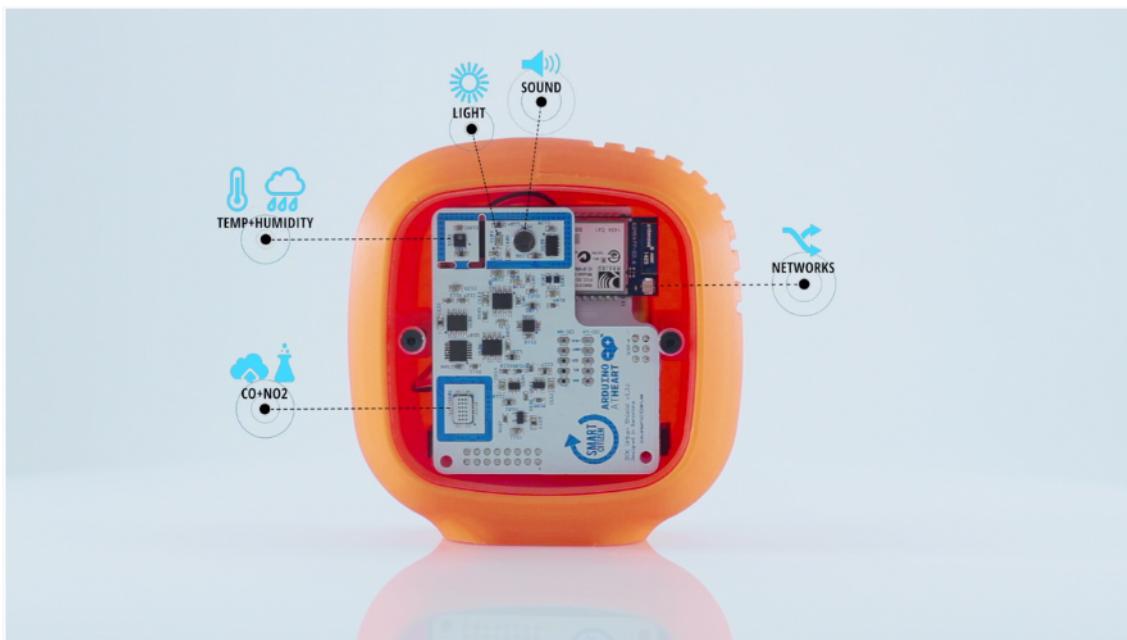


Figura 3: Dispozitivul Smart Citizen Kit [7]

Acest dispozitiv reprezintă o colecție de senzori și o placă de procesare a datelor acumulate de senzorii enumerați mai sus. Odată setat acesta va începe colectarea și procesarea datelor după ce va raporta în mod regulat prin intermediul conexiunii la rețea via modulul wireless din cadrul dispozitivului. [7]

Platforma de față mi-a servit drept exemplu util din care am putut învăța câteva lucruri, și anume: Cum pot construi un sistem de monitorizare la nivel global? Cum livrez utilizatorului datele colectate. În cele din urmă am observat oarecum și neajunsuri, unul din ele fiind faptul că dispozitivul de colectare a indicatorilor este static. Si nu în ultimul rand ideea de bază pe care am învățat-o de aici vine în crearea unui sistem evolutiv acestuia și anume combinarea datelor referitoare la mediu cu localurile și evenimentele care au loc în zona respectivului dispozitiv.

## 4 Tehnologii utilizate

### 4.1 Ionic

"WRITE ONCE. DEPLOY ANYWHERE" [8] este motoul acestui framework. Ionic2 este un sdk construit pe baza Angular 2 pentru a putea construi aplicații mobil de tip cross-platform. Sa vedem principalele motive de ce am ales sa lucrez anume cu acest framework în cadrul lucrării de licență.

#### 1. Cross Platform

Sper deosebire de alte framework-uri în care trebuie să scrii cod separat pentru aceleași funcționalități pentru platforme diferite, care îți consumă timp. Ionic oferă posibilitatea să scrii același cod care va fi suportat de toate platformele mobile. Astfel salvând timp și eliminând o sursă de posibile conflicte.

#### 2. Gratuit

Ionic intră în categoria open source.

#### 3. Bazat pe Apache Cordova

Apache Cordova este un framework open-source de construire a aplicațiilor mobile. Care permite dezvoltatorilor să utilizeze standardele tehnologiilor web (HTML5, CSS3 și JavaScript) pentru crearea de aplicații cross-platform. În cele din urmă Apache Cordova a devenit un standard pentru dezvoltarea aplicațiilor hibrid. Aceasta are o comunitate foarte dezvoltată de la care putem folosi plugin-uri care ne oferă acces nativ la platforma pe care rulează aplicația. Ceea ce permite crearea de funcționalități avansate specific platformei în cadrul aplicației. Despre plugin-uri mai detaliat în Capitolul 6. Aceasta oferă libertatea de utilizare a tehnologiei web în dezvoltarea aplicației și compilarea ale acesteia în executabile native ce oferă un plus de performanță.

#### 4. Angular 2 și TypeScript

După apariția suportului pentru Angular 2 și Typescript pentru Ionic, dezvoltarea în cadrul acestuia a devenit cu mai ușoară. Angular 2 fiind suport de Google și Angular este una din cele mai populare SPA(Single Page Application) framework. Care permite construirea unei aplicații complexe bazate pe **componente**. În angular 2 aplicația este organizată în forma unui arbore de componente ce au cuplaj minim.[9] În plus Angular2 față de AngularJS și-a sporit semnificativ caracteristicile de performanță .

## 5. Comunitatea

Uneori pierzi zile întregi pentru ca nu găsești sursa unei erori, din cauza unei documentații prost făcute ori deloc intuitive. Aici Ionic 2 vine cu o documentație bine pusă la punct în care ai tutoriale și detalii de utilizare a serviciilor oferite. Adițional celor spuse mai sus ne putem bucura un forum în care poți primi ajutor de la alți dezvoltatori.

### 4.2 Redis

În cadrul unei aplicații mobile bazate pe principiul client-serever un aspect important este viteza( ne dorim ca aceasta să fie cat mai rapidă, în ceea ce privește timpul de răspuns al serverului). De aceasta am ales să folosesc drept bază de date Redis.

Redis (REmote DIctionary Server) este o formă de stocare în memorie a structurilor de tip cheie-valoare ce poate fi folosită în calitate de baza de date. Aceasta este o alegere foarte bună în cadrul dezvoltării de aplicații Web, Mobile, Gaming și IoT, aplicații ce necesită performanță sporită.[10]. Suportă multiple structuri de date : string, hash-uri, liste, multimi etc. asupra cărora putem realiza operații atomice rapide spre exemplu adăugarea/eliminarea unui element în lista.

Persistența datelor este o alta opțiune oferită de redis. Aceasta utilizează strategii de stocare a stării curente pe disk la intervale date de timp.

### 4.3 Azure

Microsoft Azure mai numit Windows Azure este o platformă de tip cloud-computing dezvoltată de Microsoft ce oferă servicii de hosting și management la nivel de PaaS. Acesta ofere dezvoltatorilor la cerere instanțe de calcul cat și mediu de stocare în cadrul datacenterelor proprii cu un nivel de disponibilitate ridicat. Folosind aceste servicii putem crea și găzdui aplicații complexe care vor putea face față solicitărilor mărite datorită capacitaților scalabile oferite. În cadrul aplicației în acest cloud vor fi găzduite 2 componente precum voi utiliza și unul din serviciile de storage oferite. Am ales Azure deoarece este ușor de utilizat cu o interfață foarte intuitivă și un suport de calitate.

#### 4.3.1 MongoDB Azure Service

MongoDb este o baza de date de tip NoSQL. Aceasta este construită peste o arhitectură de tip colecții și documente. Un document conține o mulțime de perechi cheie-valoare, docu-

mentul este unitatea atomica de date din MongoDB. Colectia conține un set de documente cu funcționalități clasice de tip query din bazele de date relationale.[11]

Microsoft are soluția proprie în ceea ce privește stocarea datelor: Azure CosmosDB aceasta este următoarea generație al Azure DocumentDB. Un mediu de stocare distribuit la nivel global ce oferă scalabilitate pe orizontală (Scale-Out). Identificăm următorul serviciu "Database as a Service for MongoDB). Acesta fiind accesibil via API. [12]

O imagine de ansamblu asupra arhitecturii distribuite ale serviciului de stocare CosmosDB o putem vedea în figura 4

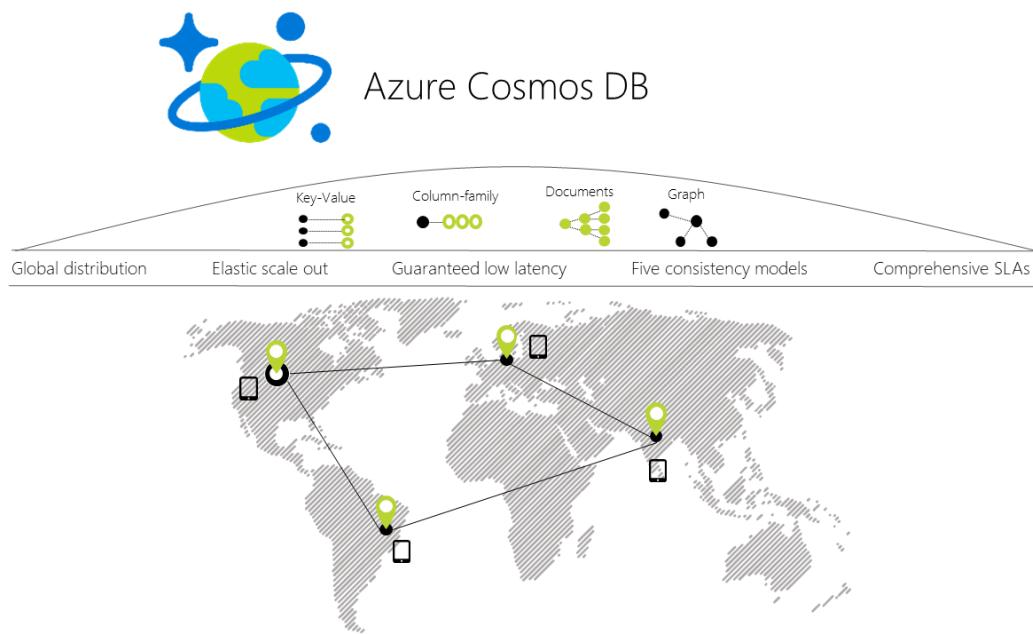


Figura 4: Cosmos DB imagine ed ansamblu.[13]

#### 4.4 AWS EC2

Amazon Elastic Compute Cloud este un serviciu cloud care oferă securitate și scalabilitate în ceea ce privește capacitatea de calcul în cloud, care are o interfață intuitivă și este foarte ușor de utilizat.

Interfața Web Amazon EC2 îți oferă posibilitatea să configurezi mașina care urmează să îți-o ridici cu efort minim. Primești control complet asupra resurselor computaționale pe care le obții în cadrul clusterului Amazon. Timpul de ridicare a unei noi instanțe de server este redus la minute ceea ce este un plus de performanță atunci când e nevoie să scalezi aplicația la creșterea sau micșorare cerințelor. Instanța rulează într-un mediu izolat cu o capacitate rezilientă sporită. Revenirea de la erori poate fi programată în cazul unor scenarii specifice prin anumite fișiere de configurare. Ce ține de sistemul de pay-per-use AWS-ul are pus la punct un sistem

de monitorizare a resurselor utilizate astfel asigurând faptul că plătești doar resursele pe care le utilizezi.[14]

În Cloud-ul AWS urmează să hostez instanțe ale serverului de baza. Acesta va monitoriza locațiile de tip mgrs de nivelul 3(zone de 100m) va tine la curent device-urile care se află într-o anumită zonă, precum și procesarea datelor care vin de la senzorii din dispozitivele mobile. Va răspunde la cererile utilizatorilor conform zonei în care se află la moment. Având acces direct la instantă de ec2 folosesc drept utilitar de stocare Redis pentru a micșora timpul de răspuns. Acesta fiind nucleul aplicației el trebuie să aibă o reziliență mare precum și disponibilitate sporită precum și scalabilitate la creșterea numărului de cereri, caracteristici asigurate de instanțele de EC2. Putem vedea detalii de despre instanță curentă de ec2 în figura5

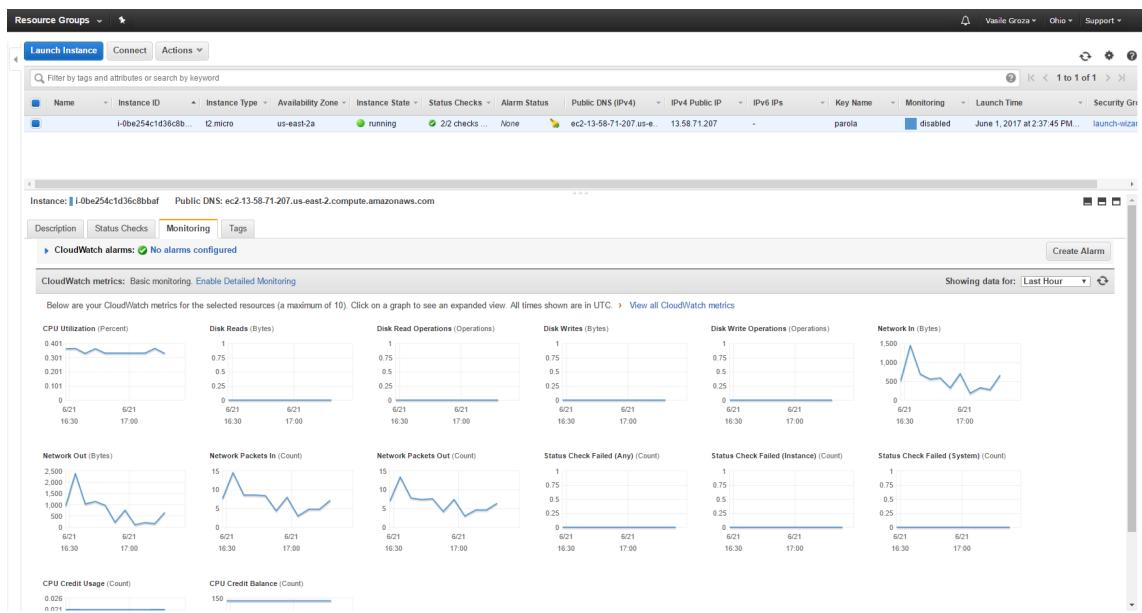


Figura 5: EC2 consola.

## 4.5 Military Grid Reference System

Sistemul "Military Grid Reference System" (MGRS) oferă o alternativă de reprezentarea unei locații pe suprafață globală, utilizând un string alfanumeric. Acesta are o referință de ordin ierarhic care se bazează pe principiile sistemului "Universal Transverse Mercator". Locațiile sunt împărțite în pătrate de la 6-8 grade până la pătrate foarte mici de ordinul 1m la cea mai mare precizie( detalii figura 6).

### Truncated position formats for less precise positions

The MGGRS format is designed to support measurement precisions of 1m, 10m, 100, 1,000m, and 10,000m, by truncating the grid coordinate values.

10S GJ 06832 44683	- Locates a point within a 1 meter square
10S GJ 0683 4468	- Locates a point within a 10 meter square
10S GJ 068 446	- Locates a point within a 100 meter square
10S GJ 06 44	- Locates a point within a 1,000 meter or 1 kilometer square
10S GJ 0 4	- Locates a point within a 10,000 meter or 10 kilometer square
10S GJ	- Locates a point within a 100,000 meter or 100 kilometer square

Figura 6: precizia sistemului MGGRS[15]

Avantajul major ale acestui sistem de reprezentarea este în managementul și vizualizarea datelor de într-o anumită locație, cat și flexibilitatea alegerii zonelor geografice și atașarea la acestea a unor informații (precum nivelul de zgomot, device-urile dintr-un anumit patrat, etc.) ca în cele din urmă oferind și posibilitatea de interogare dinamica pe baza de string-uri asociate zonei (perechi cheie valoare) pentru a monitorizarea și interpretare a datelor dintr-o anumită arie.[16] Aceasta metodă de reprezentarea a zonelor îmi servește drept baza pentru sistemul de monitorizare ale atributelor mediului la fiecare zona înregistrată în cadrul orașului.

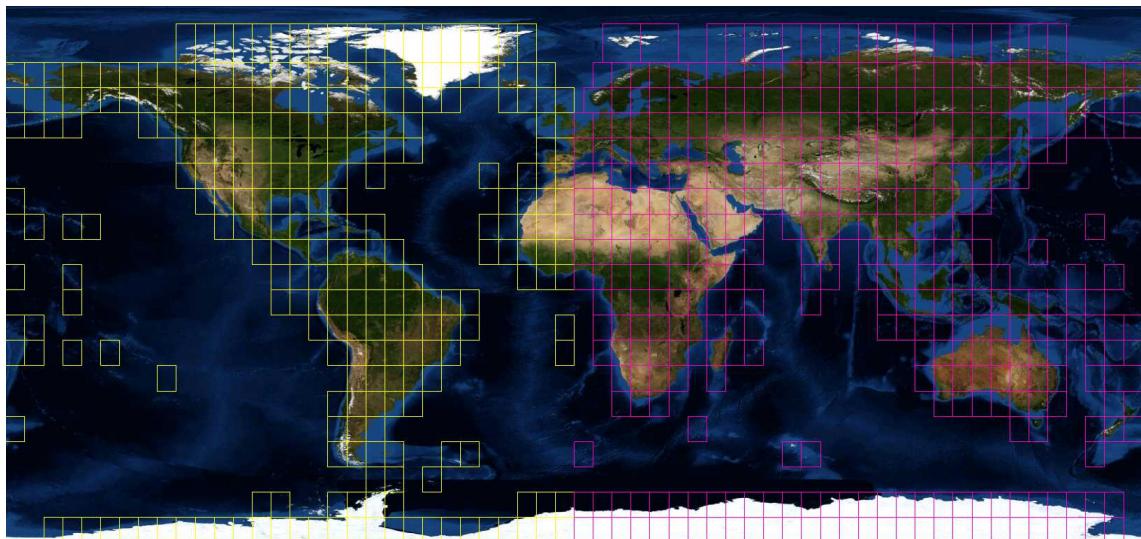


Figura 7: Sunt selectate zonele care intersectează uscatul.(verde : emisfera vestică , violet: emisferaestică)[16]

Ex: Facultatea de Informatică Iași se află la coordonata mgrs cu valoarea: **35TNN43702460**  
acest lucru poate fi văzut în figura 8

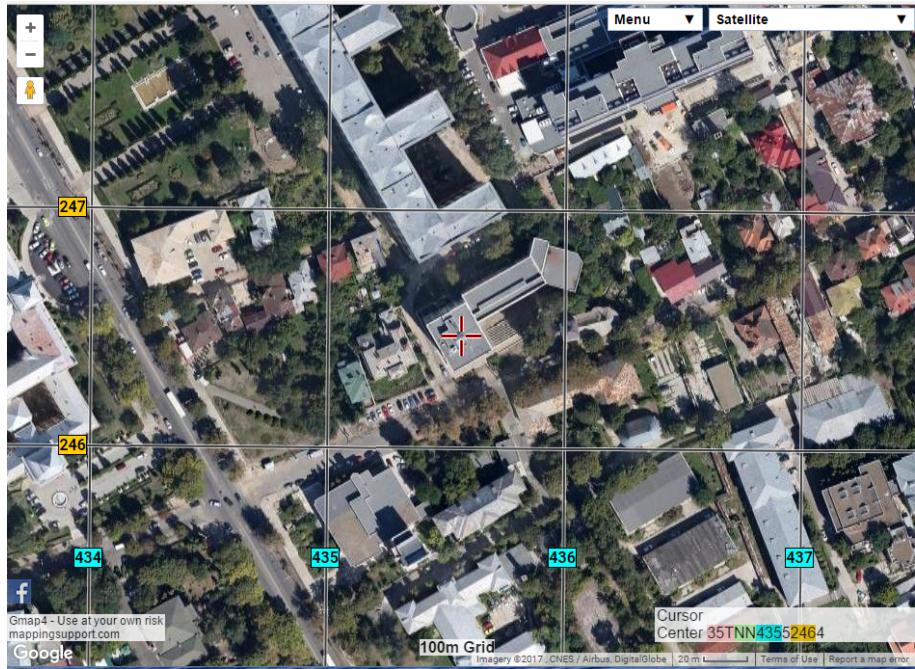


Figura 8: Facultatea de Informatică pe harta sistemului mgrs[17]

## **5 Arhitectura aplicației**

"Construcția unei aplicații Orientate pe Obiect este similară cu cea a unei case: dacă nu are o structură solida se dărâma ușor"

### **5.1 Cerințele aplicației**

Prima etapa în dezvoltarea unui sistem software este determinarea scopurilor și obiectivelor care se cer a fi îndeplinite de sistemul care urmează a fi construit. Aplicația "Towards a Smart City" vine ca o aplicație ce încearcă să ofere o soluție de integrare a orașului intelligent prin intermediul unei aplicații mobile.

Obiectivul principal al aplicației este de a ține utilizatorul la curent cu ceea ce se întâmplă în orașul în care se află. Astfel îi vor fi oferite informații despre:

1. Oraș
2. Evenimentele ce au loc în oraș
3. Localurile din apropierea lui după categorii.
4. Nivelul de zgomot de la o anumită locație.
5. Starea vremii de la locația la care dorește să meargă

Odată ce utilizatorul dispune de aceste informații el ar vrea să meargă spre exemplu la unul din evenimente. Prin urmare apar următoarele cerințe

1. Posibilitatea de a salva/șterge evenimentele din lista de activități a utilizatorului
2. Crearea unui istoric cu evenimentele la care a participat utilizatorul
3. Vizualizarea pe hartă a locației la care are loc evenimentul( cu posibilitatea de a vizualiza și drumul pana la acesta).
4. Pe lângă evenimente organizate în cadrul orașului utilizatorul ar dori să își poată programa o ieșire în una din locațiile oferite de aplicație din apropierea al acestuia. ex: o plimbare în parc. O ieșire la cafenea.

## 5.2 Arhitectura generală

Acestea fiind cerințele principale pe care trebuie să le îndeplinească aplicația, să urmărim cum realizăm aceste funcționalități. Dar pentru început să urmărim arhitectura generală a aplicației din figura 9

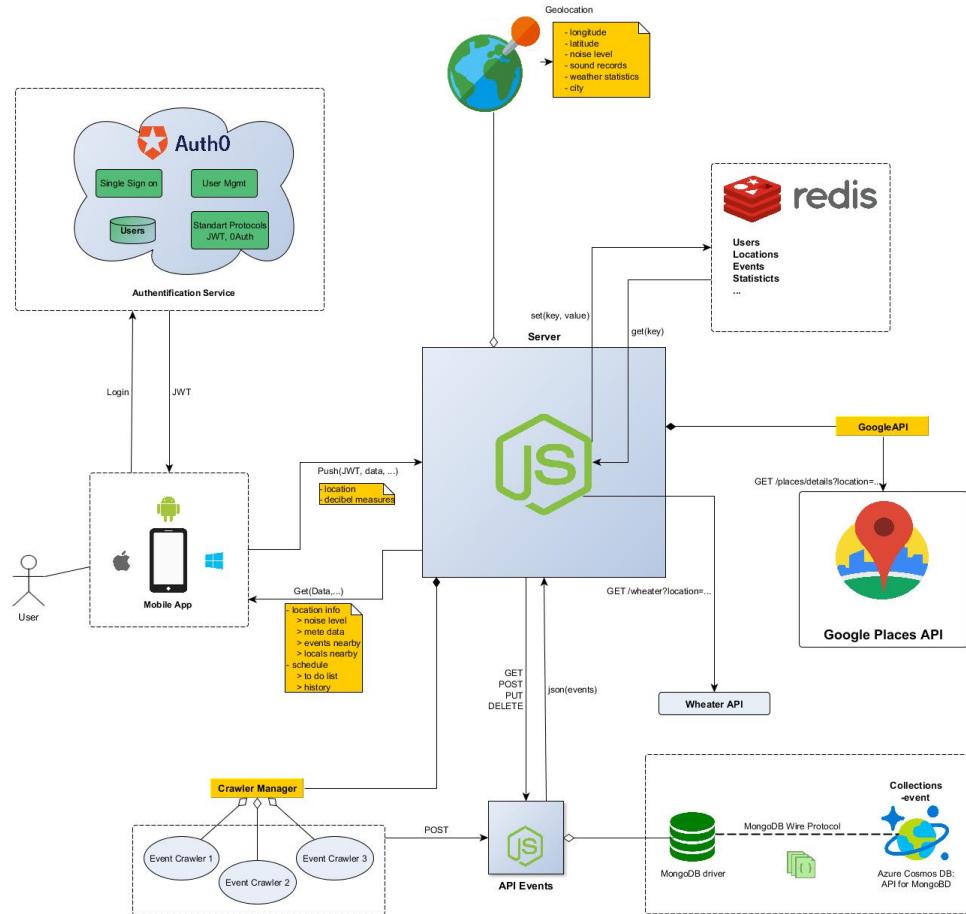


Figura 9: Arhitectura generală a aplicației

Prin intermediul aplicației mobil utilizatorul intră în contul său, comunicând date prin intermediul internetului către server (locația la care se află, nivelul de zgomot). Acesta de la server drept răspuns primește informații despre oraș, evenimente și localuri cat și alte informații aferente. Din figura 9 observăm ca avem aplicația separată în 3 mari componente

1. Client
2. Server
3. Event API

Despre acestea vom discuta detaliat în subcapitolele următoare.

### 5.3 Client

Componenta Client este o aplicație mobilă disponibilă și în mediul web. Aceasta este scrisă în framework-ul Ionic 2, aspectele generale de ce am ales să scriu aplicația folosind această tehnologie se află în capitolul 4 . 1. Iar motivul de bază a fost faptul că este foarte ușor să generez pachetul de instalare a aplicației mobilă odată ce am realizat-o prin intermediul tehnologiilor web (figura 10).

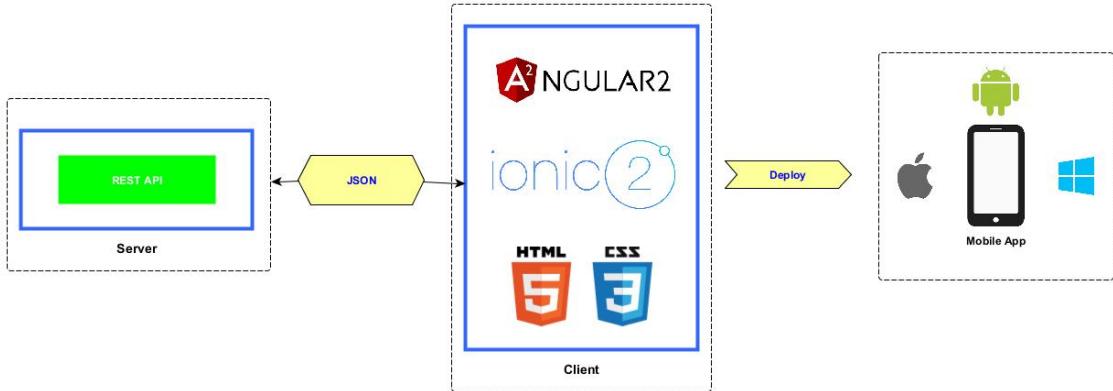


Figura 10: Aplicația mobil

Combinarea Angular 2 și Typescript, îmi permit crearea unei aplicații robuste. Angular 2 având la baza principiul de construcție bazat pe componente oferă posibilitatea de a privi fiecare entitate din pagina web ca fiind o componentă independentă. O componentă este un bloc independentă care are o logică, un view și model care să o reprezinte. Template-ul ce reprezintă felul în care va fi vizualizată componenta, metodele care controlează interacțiunea cu acea componentă și atributele propriu zise (modelul) permit implementarea principiului Model-View-Controller la nivel de componentă, asigurând astfel reutilizarea acelei componente în cadrul aplicației.[18] Ceea ce permite modularizarea la nivel de client.

În Ionic2 având suport Angular2 se face abstractie la nivel mai înalt și anume la nivel de pagini (similar stivei de activități din Android) astfel organizarea devine mai puțin dificilă. Prin intermediul pluginurilor obținem acces la mediu de stocare (localStorage), senzori precum GPS și AudioInput pentru a determina Locație și Nivelul de zgomot. Despre aceasta mai detaliat în capitolul 6. O imagine de ansamblu putem vedea în figura 11, Ionic 2 fiind un nivel superior pentru Cordova el are aceeași schemă.

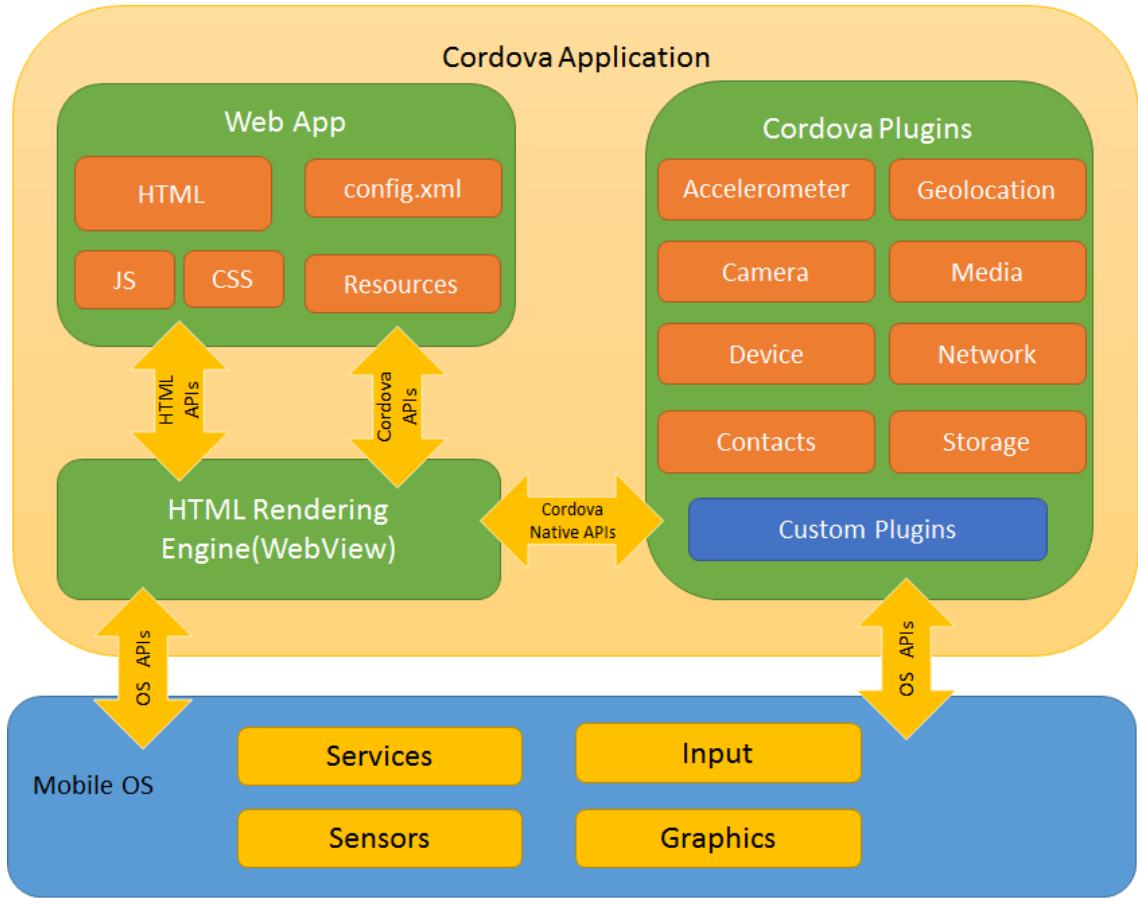


Figura 11: Arhitectură Cordova [19]

## 5.4 Server

Componenta server este scrisă în limbajul Node.js folosind framework-ul express.js. Această combinație face posibilă crearea unui API REST securizat într-o manieră ușoară. Din figura 12 observăm aspectul general al arhitecturii serverului și anume, se ieșe în evidență următoarele lucruri:

1. **Authentication Middleware** Aceasta verifică validitatea utilizatorului care face request-ul prin verificarea JWT-ului(Json Web Tokens)
2. **Crawler Manager** O clasă creată pentru management-ul web crawlerilor care trebuie să îndeplinească o unică funcție și anume să actualizeze lista de evenimente de la locațiile la care sunt cerute.
3. **Modul de Stocare** Drept modalitate de stocare pe partea de server utilizez Redis. Acesta fiind baza de date in-memory asigură o viteză de accesare foarte rapidă astfel micșorând timpul de răspuns al serverului. În Cadrul acesteia am 3 tipuri de perechi cheie valoare

și anume: 1.user\_id: User, 2. device\_id: Device , 3. mgrs: Geolocation. Detaliat despre acestea în capitolul următor *"Implementare"*.

4. **Google API** este o clasă care este intermediarul dintre API-ul oferit de google și aplicația utilizator, Aceasta are ca scop procesarea cererilor după ce au fost receptate de Layer-ul Rest și au trecut de autentificare și oferirea unui răspuns(ex informații despre un anumit local din apropierea utilizatorului).

5. **Wheater API** Acesta este modulul care se ocupă de obținerea informațiilor meteorologice de la o anumită locație și oferă aceasta ca un serviciu.

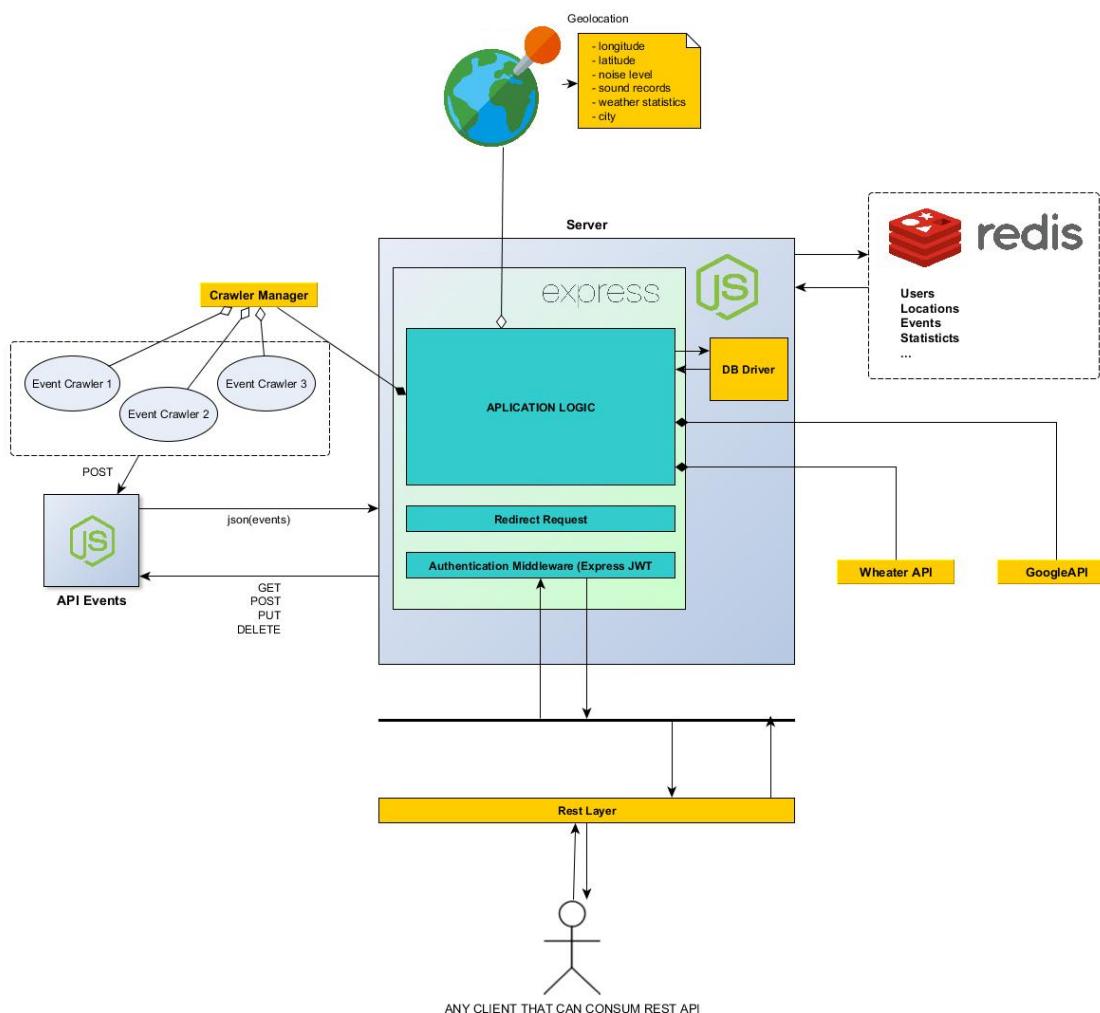


Figura 12: Arhitectură Server

## 5.5 Event API

urmează să descriu modul în care în care este construit API-ul cum stochez în baza de date. API-ul de evenimente este construit folosind framework-ul Express.js în Node.js, unicul model fiind **Eveniment**-ul. Prin intermediul API-ului RESTful se realizează principiile CRUD (CREATE, READ, UPDATE, DELETE) cărora le corespunde următoarele verbe HTTP: (POST, GET, PUT, DELETE) care sunt expuse către mediul extern. Astfel API-ul expus devine un utilitar pentru management asupra evenimentelor. Diagrama acestui api poate fi urmărită în figura 13

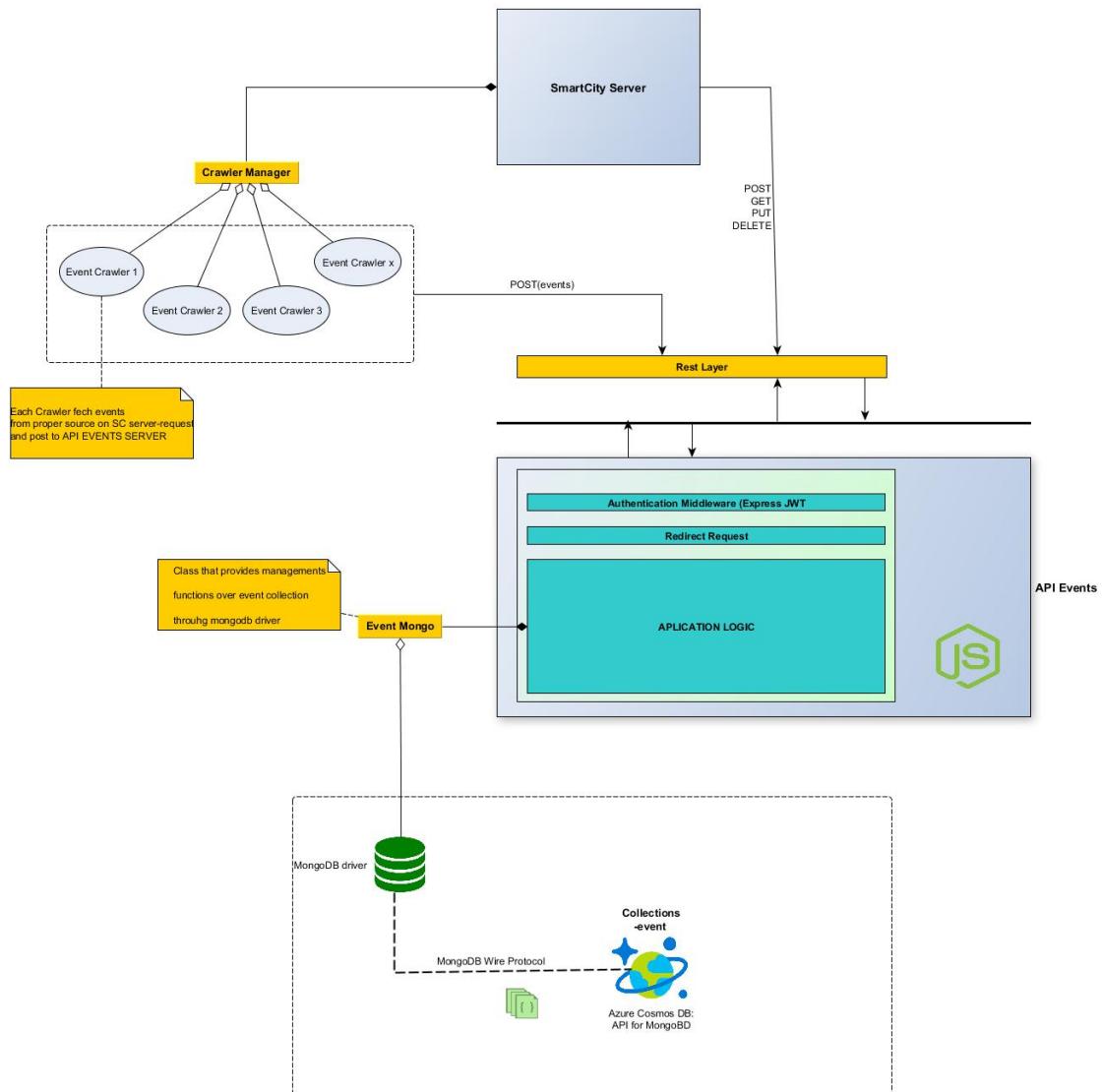


Figura 13: Diagrama API-ului de evenimente

Această mod de organizare a modulului Event API permite extensibilitatea (scale-out) având în vedere că serviciul de stocare distribuie la nivel global CosmosDB. Aceasta mai oferă și posibilitatea de distribuție după regiuni asigurând disponibilitatea și timp de acces resurse mai rapid pentru utilizatorii ai api-ului din regiunea din care este făcută cererea.

De asemenea "Event Crawlers" aceştia pot fi cat funcții atât cat și mici servere care știu să facă următoarele lucruri:

1. Să colecteze evenimente dintr-o anumită sursă.
2. Să realizeze un request de tip POST în care să trimită evenimentele colectate pentru a crea înregistrări cu acestea în baza de date mongoDB din spatele layer-ului REST al API-ului de evenimente.

Prin urmare și numărul acesta poate crește oferind astfel posibilitate de extindere a aplicației.

## 6 Implementare

### 6.1 Client

Aplicația *Towards a Smart city* este o aplicație mobil construită folosind framework-ul Ionic 2 bazat pe Angular 2 și Typescript. Aceasta fiind organizată pe pagini, componente, directive și servicii oferă posibilitatea de a crea o aplicație modulară ceea ce facilitează o organizare elegantă a codului cat și posibilitatea de reutilizare a acestuia<sup>14</sup>. În cele ce urmează vă voi prezenta funcționalitățile fiecărei pagini din aplicație.

Această este comușă din următoarele pagini

#### 1. Login

Aceasta este pagina de autentificare a utilizatorului în contul propriu în cadrul aplicației.

Autentificarea este pe baza de token-uri JWT acestea asigurând proprietatea de independentă a stării serverului față de client. Pentru autentificare am creat un serviciu și anume serviciu **auth** care are conținut logica pentru protocolul de autentificare bazat pe token-uri. În cadrul unei pagini utilizarea serviciului de autentificare arată în felul următor

```
1 export class StartUpPage {  
2   constructor(private auth: AuthService) {...}  
3   login() {  
4     this.auth.login();  
5   }  
6   logout() {  
7     this.auth.logout();  
8   }  
9 }
```

Acesta prin intermediul modulului http autentifică utilizatorul prin apeland la serviciului cloud de încredere Auth0. De asemenea este implementată logica pentru autentificare prin intermediul profilului de facebook, twiter etc. După o logare cu succes pe partea de client se salvează în local storage profilul utilizatorului cat și token-ul de autentificare pentru a rămâne logat pentru o perioadă de 36 de ore după ultima logare. După care cu ajutorul acestui token aplicația client are acces la API-ul REST expus de serverul aplicației.

#### 2. Home

Pe această pagină suntem redirectați după ce are loc logarea cu succes(Sau se deschide aceasta dacă am rămas logați din sesiunea precedentă) Aici vom găsi informație generală

despre Orașul în care ne aflăm precum și lista de localuri din apropierea noastră. Monitorizarea locației și a nivelului de zgomot de la poziția utilizatorului folosind serviciul creat **SensorCollector** care expune în formă de funcții ce returnează obiect de tip Observable cu ajutorul căruia lucrăm în mod asincron. și trimitera acestor date către serverul principal prin intermediul serviciului **ServerEmmitter** aceeași fel se realizează într-o manieră concurrentă datorită aceluiași pattern.

```

1 @IonicPage()
2 @Component({
3   selector: 'page-home',
4   templateUrl: 'home.html',
5 })
6 export class Home {
7   user: Object
8   city: any;
9   placesDictionary: Array<{ title: string, placesObj: Object, icon:
10   string, showDetails: boolean }> = [];
11   constructor(public sensorCollector: SensorCollector,
12     public auth: AuthService,
13     public serverEmmitter: ServerEmmitter) {
14     this.watchLocation();
15   }
16   watchLocation() {
17     this.positionSubscription = this.sensorCollector.getLocation()
18       .subscribe(position => {
19         this.serverEmmitter.emmitLocation(location)
20           .subscribe(
21             data => {
22               // location changed update city data and load places
23               this.loadCityData();
24               this.loadPlaces();
25             });
26     }
27     measureNoise(location) {
28       this.sensorCollector.recordDecibel(10000).then((recordResult)
29         => {
30           let decibelMeasure = {"mgrs": location,
31                             "recordResult": recordResult}
32           this.serverEmmitter.emmitNoiseLevel(decibelMeasure);
33         });
34     }
35   }
36 }
```

```

31   loadPlaces() {
32     this.serverEmmiter.getCityPlaces(this.auth.user.
33       mgrs_currentLocation)
34   }
35   loadCityData() {
36     this.serverEmmiter.getCityGeneralData(this.auth.user.
37       mgrs_currentLocation)
38   }

```

### 3. Events Nearby

Această pagina are drept scop principal afișarea evenimentelor din apropierea locației la care se află utilizatorul în momentul curent. Pentru început se accesează localstorage-ul pentru a vedea setările utilizatorului (raza R și numărul de zile X) pentru a putea face un request la server și de a cere toate evenimentele din jurul clientului la R kilometri pentru următoarele X zile.

```

1  export class EventListPage {
2    events: Array<Object>
3    storage: Storage = new Storage('localStorage')
4    constructor(public serviceEmmiter: ServerEmmiter,
5      public authService: AuthService,) {
6
7      let radius = this.storage.get('radius');
8      let nrDays = this.storage.get('nrDays');
9      this.loadEvents(radius, nrDays);
10    }
11    loadEvents(radius, nrDays) {
12      this.serviceEmmiter.loadAllEvents(params)
13        .subscribe(
14          data => { //save events });
15        })
16    }

```

Randarea evenimentului realizându-se de componenta Event din cadrul aplicației pe care apăsând vom deschide pagina cu detalii despre eveniment în care vor fi informații generale

despre eveniment cat și metadate despre locația la care se desfășoară acel eveniment: starea vremii și nivelul de zgomot la momentul curent.

Fiind oferită și posibilitatea de a vizualiza pe hartă locația la care va avea loc acel eveniment dând click pe o zonă din cadrul card-ului ce reprezintă evenimentul.

```
1 eventTapped(e, event) {
2     this.navCtrl.push(EventDetails, {
3         "event": event,
4         "prevPage": this.navCtrl.getActive().name
5     });
6 }
```

4. To Do List În urma analizei problemei și stabilirea cerințelor ce trebuie îndeplinite (în capitolul 5.1) apare necesitatea de a crea o pagină separată (pentru comoditate) care va avea rol de tablă cu notițe în care utilizatorul își va salva evenimentele la care ar vrea să meargă. Desigur și posibilitatea de a șterge "notițe" din aceasta.

Cum am menționat mai devreme Angular 2 este bazat pe componente care pot fi reutilizate, comune pentru aceste ultimele 2 pagini menționate sunt: lista de evenimente și evenimentul în sine, precum și utilizarea acestora și servicii create anterior: Încărcarea evenimentelor personale are loc prin crearea unei cereri de tip GET pe ruta */schedule* a serverului apelând la serviciul **ServerEmmiter**.

5. Settings Aici utilizatorul își poate seta anumite preferințe în cadrul aplicației mobil pentru o satisfacerea necesităților acestuia.

Se pot seta următoarele proprietăți

- (a) Unitatea de măsură a temperaturii (Celsius, Fahrenheit, Kelvin)
- (b) Distanța dă căutare a evenimentelor în kilometri (1-21)
- (c) Numărul de zile are următorul înțeles: pentru cate zile în viitor să se afișeze evenimente din apropiere.

Și nu în ultimul rand cele mai importante componente ale aplicației sunt plugin-urile Apache-Cordova care oferă acces nativ la funcționalități ale device-ului acestea în cadrul aplicației fiind:

1. cordova-plugin-geolocation: care îmi oferă acces direct la senzorul de GPS din cadrul dispozitivului mobil, astfel obțin locația cu precizia de 10-50 m.

2. cordova-plugin-dbmeter: ce îmi permite să accesez microfonul dispozitivului și să realizez măsurători ale nivelului de zgomot din jurul acestuia.(Realizând o medie ale măsurătorii anume pe aceasta o trimit către server în scopuri de "privacy protection".
3. cordova-plugin-googlemaps : instalarea acestui plugin (plus cheile de securitate din google API pentru Android și IOS, dacă este cazul) permite utilizarea google maps la nivel nativ în aplicația persoană ce sporește viteza de încărcare a acesteia cât și câștigul de trafic care ar fi fost pierdut dacă utilizam încărcarea dinamică a hărții din serviciul oferit public de Google.

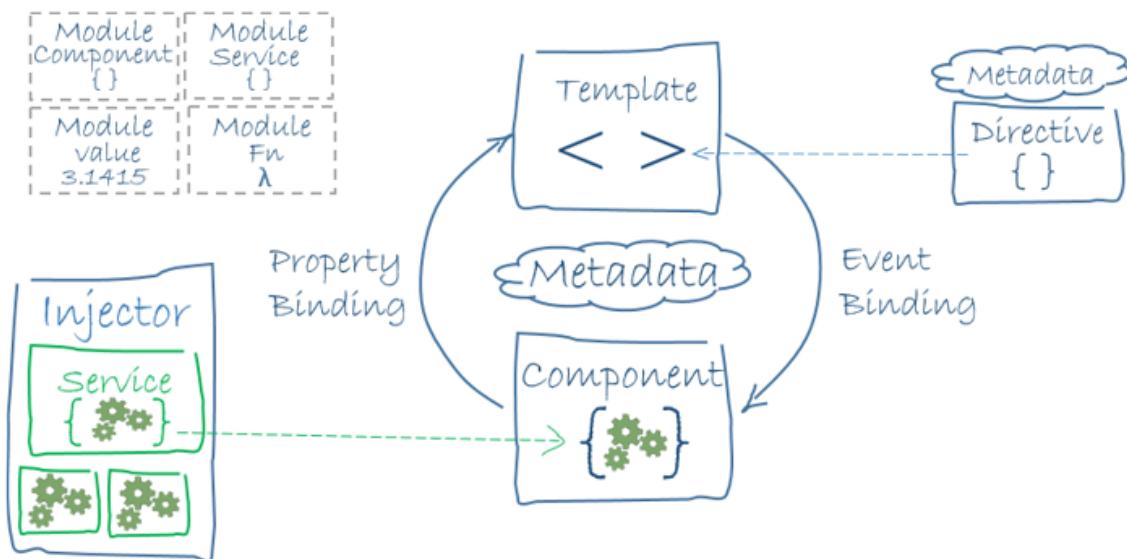


Figura 14: Angular 2 general architecture.[20]

## 6.2 Server

Serverul "Towards a Smart City" este scris în limbajul nodejs folosind framework-ul express.js cu ajutorul căruia este foarte ușor să construieni un API REST folosind modulul ROUTER. Utilizarea acestuia se demonstrează în secvență imediat următoare.

```
1 var express = require('express');
2 var router = express.Router();
3 router.route('/location')
4   .post((req, res) => {
5     //update user location
6   })
7 router.route('/noise')
8   .post((req, res) => {
9     // add noise record to user's location
10  })
11 ...
```

Având Layer-ul REST creat următorul pas a fost securizarea acestuia. Pentru a realiza securizarea API-ului am utilizat autorizarea via JWT(JSON WEB TOKENS) care asigură serverul să fie stateless. Diagrama de secvență ce descrie fluxul este reprezentată în figura 15

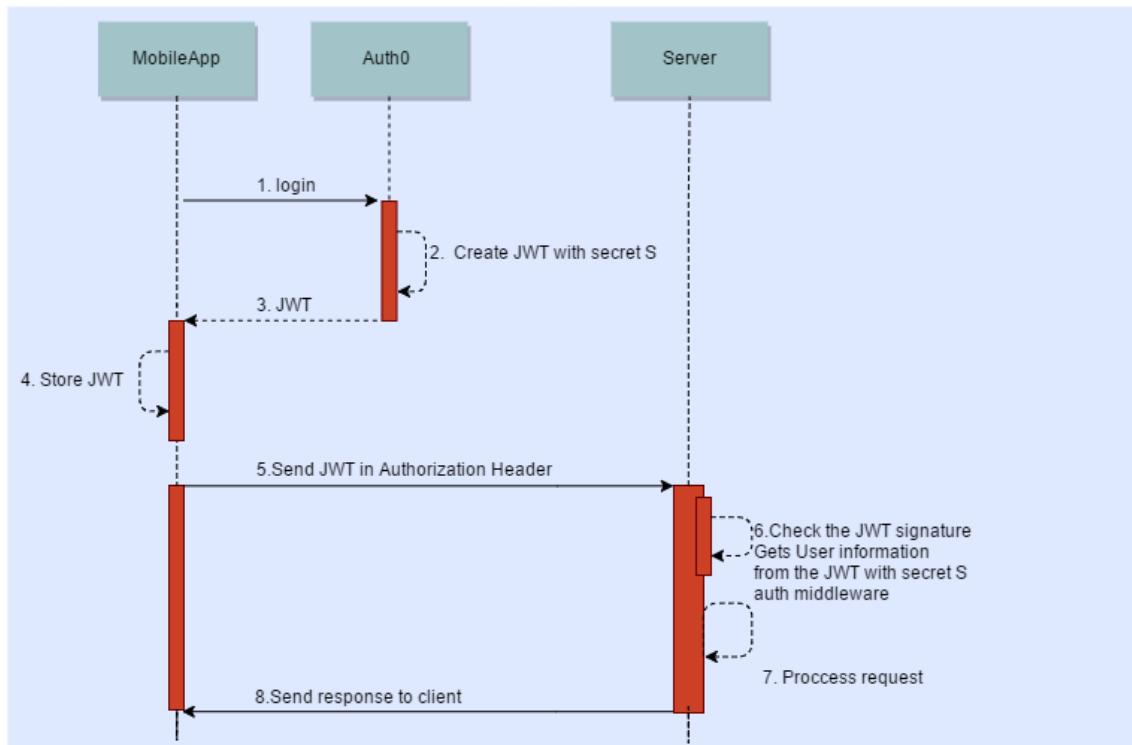


Figura 15: Diagrama de secvență de autentificare.

Serviciul Cloud Auth0 cât și serverul partajează același secret pentru verificarea semnăturii JWT-ului primit de la client în momentul în care acesta realizează o cerere la server. În partea de cod acesta se realizează prin intermediul unui middleware pentru simplitate am utilizat librăria "express-jwt", astfel încât fiecare request mai întai trebuie să treacă de validarea jwt-ului în ordine de a ajunge la resursele oferite de API-ul REST.

```
1 var jwt = require('express-jwt');
2 var jwtCheck = jwt({
3   secret: auth0Settings.secret,
4   audience: auth0Settings.audience
5 })
6 router.use(jwtCheck)
```

Baza de date, după cum am menționat în capitolul anterior, este Redis. În cadrul acesteia memorez perechi cheie valoare. Pe server am 3 clase separate și anume: pe baza cărora este construit modelul logic al aplicației

### 1. User

```
1 class User{
2   constructor(user_id){...}
3   addEvent(event, ownEvent, next){...}
4   removeEvent(event_id, rejected, next){...}
5 }
```

### 2. Device

```
1 class Device{
2   constructor(device_id, currentLocation){...}
3   updateLocation(newLocation){...}
4 }
```

### 3. Geolocation

Această clasă poate fi considerată nucleul propriu zis, deoarece în cadrul acesteia se realizează managementul device-urilor de la o locație respectivă implicit pătrate de lungime de 100 de metri cu alte cuvinte o precizie de ordin 3 în coordonate mtrs. Cât și monitorizarea nivelului de zgromot asociat zonei.

```

1 const mgrs_converter = require('mgrs');
2 class Geolocation{
3     constructor(longitude, latitude, mgrs_value) {...}
4     addDevice(device){...}
5     removeDevice(device){...}
6     addRecord(record){...}
7     clearNoiseMeasures(){...}
8 }

```

Scrierea/Citirea acestora din mediul de stocare realizându-se prin intermediul clasei RedisConnector. Aceasta realizând operațiile de *get* și *set* asupra bazei de date Redis oferind funcționalități mai complexe conservând principiul de asincronism cum putem observa în secvența următoare de cod:

```

1 var redis = require("redis");
2 var Coordinate = require("./coordinate");
3 class RedisConnector {
4     constructor(ip, port) {
5         this.redis_cli = redis.createClient();
6         this.redis_cli.on('connect', () => {...});
7     }
8     getCoordinate(mgrs_value, next) {
9         let obj;
10        this.redis_cli.get(mgrs_value, (err, result) => {
11            if (err)
12                return next(err);
13            if (!result) {
14                obj = new Coordinate(null, null, mgrs_value);
15            } else {
16                let coordinate_data = JSON.parse(result);
17                obj = new Coordinate(coordinate_data);
18            }
19            return next(null, obj);
20        });
21    }
22 ...

```

Pentru a oferi utilizatorului localurile apropriate în partea de server am creat o clasă care se ocupă de apelurile către API-ul *Google Places* oferit de Google cu ajutorul căruia pot obține localurile din o rază x km și o anumită categorie. Astfel în acest mod păstrează privată cheia de acces al api-ului. Un exemplu de utilizare al clasei *GoogleAPI* ar fi următorul. Această secvență obține informație despre orașul căruia aparține portiunea asociată valorii *mgrs* din sistemul "Militaty Grid Reference System".

```

1 router.route('/city/:mgrs_value').get((req, res) => {
2     let geoPoint = mgrs.toPoint(req.params.mgrs_value);
3     googleAPI.getCityByGeocoordinate(geoPoint[1], geoPoint[0], (err,
4         result) => {
5             res.status(200).json({
6                 status: 'ok',
7                 result: result
8             });
9         });
10 });

```

Și desigur ca nu în ultimul rand componenta de *Wheather API* joacă un rol important în caserul serverului, cu ajutorul acestuia obținem informațiile referitoare la starea vremii la o anumită locație, cat și prognoza pentru următoarele 7 zile. Un exemplu de răspuns cu prognoza meteo pentru următoarele 3 zile de la serverul aplicației arată în felul următor figura 16

```

1 <
2   "city": { },
3   "cod": "200",
4   "message": 0.08448,
5   "cnt": 3,
6   "list": [
7       {
8           "dt": 1498384800,
9           "temp": {
10               "day": 30,
11               "min": 23.39,
12               "max": 30,
13               "night": 23.39,
14               "eve": 30,
15               "morn": 30
16           },
17           "pressure": 1012.2,
18           "humidity": 68,
19           "weather": [
20               {
21                   "id": 800,
22                   "main": "Clear",
23                   "description": "sky is clear",
24                   "icon": "01d"
25               }
26           ],
27           "speed": 2.67,
28           "deg": 196,
29           "clouds": 0
30       },
31       { },
32       { }
33   ]
34 }
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90 >

```

Figura 16: Wheather Response

Clasa care este se ocupă de managementul evenimentelor în mod indirect este CrawlerManager, ce conține o listă de instanțe ale claselor Crawler ce implementează Interfață generică Crawler. Acestea trebuie să implementeze metoda *notify* și trebuie să accepte configurarea serverului Event API pentru a putea fi integrată în fluxul general al serverului. În cadrul serverului execută un cron-job care o dată la zi notifică toți prin intermediul CrawlerManager-ului toți crawlerii ca să actualizeze lista de evenimente din sursa pentru care sunt creați și să realizeze că un request de tip *POST* pe adresa serverului EventAPI pentru a salva acele înregistrări de evenimente continuare în capitolul 6.3.

### 6.3 Event API

Serverul Event API este responsabil de managementul evenimentelor în cadrul arhitecturei generale a aplicației. Aceasta expune rutele

1. \api\events
2. \api\events\event\_id

Asupra acestor rute sunt suportate verbele POST, GET, PUT, DELETE pentru asigurarea principiilor CRUD (Create, Read, Update, Delete), acestea se realizează prin intermediul clasei EventMongo care este un nivel abstract ce permite accesarea datelor din CosmosDB prin intermediul MongoAPI și oferă instrumente de inserare, ștergere, actualizare și interogare asupra datelor din mediul de stocare distribuit oferit de Microsoft Azure. Interrogările sunt parametrizate, această funcționalitate este oferită de QueryStrings din url-ul resursei ai API-ului REST.

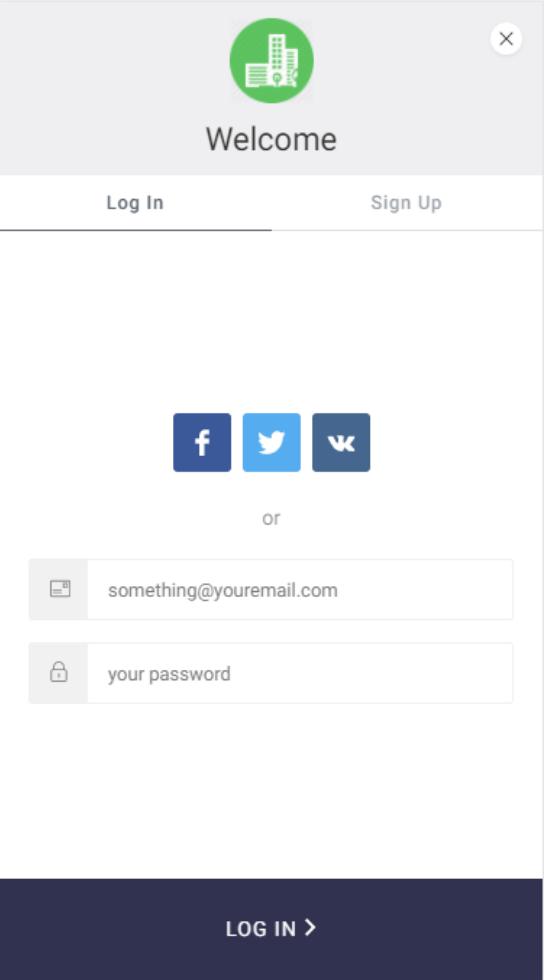
```

1 router.route('/events').post(function (req, res) {
2   eventModel.addEvent(req.body.source, req.body.events, (err, result)
3     => {
4       if (err) {
5         return res.status(500).json({ "status": "error while saving into
6           database" });
7       }
8       return res.status(200).json({
9         "status": "events created",
10        "result": result});
11     }
12   })

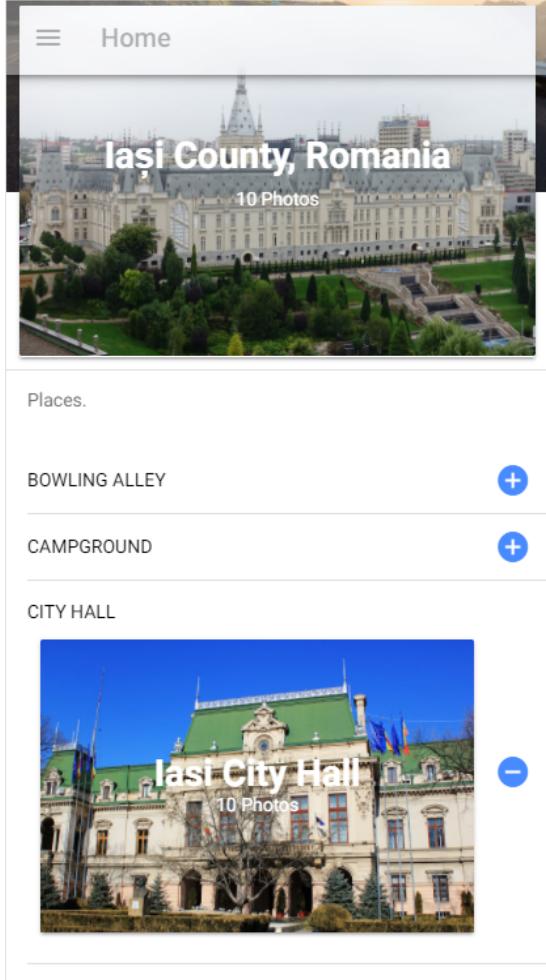
```

În capitolul anterioare am discutat despre arhitectura generală cât și pentru fiecare componentă aparte, am urmărit și implementarea acestora. Este timpul să vedem și un flux general al aplicației. Cu exemple de utilizare practice în cadrul aplicației create. Primul pas este să te înregistrezi în cadrul aplicației utilizând o adresă de email și o parolă ori logarea via profilul de *facebook*, *twiter* sau *vk* deja existent. După o logare cu succes se determină locația utilizatorului prin plugin-ul *plugin-cordova-geolocation* cat și orașul în care se află acesta. Având la dispozitie locația user-ului și orașul în care se află informațiile despre acesta se salvează pe partea de server.

Pasul următor fiind încărcarea în pagina Home a utilizatorului lista localurilor din apropierea sa din cadrul orașului cat și informații despre oraș. Acestea pot fi vizualizată în o pagina separată PlaceDetails unde avem posibilitatea de a vedea informație generală despre local cat și metadata despre zona în care se află (nivelul de zgomot și vremea acestea fiind obținute de la server prin intermediul unui request autorizat cu JWT token).



(a) Formularul de login

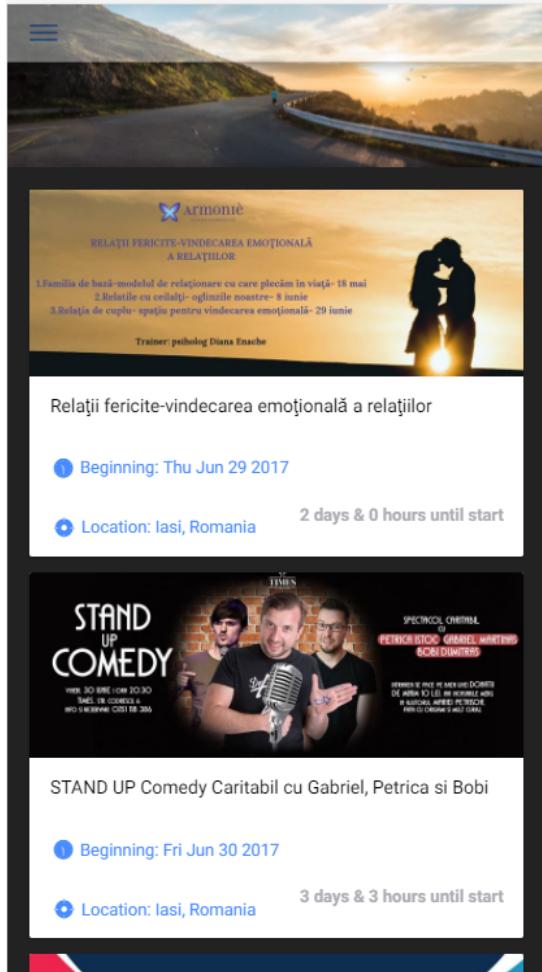


(b) Pagina Home

Figura 17: Autentificarea în aplicație

Desigur să nu uităm și despre evenimentele care le obținem de la Event API prin intermediul

serverului principal. Acestea la nivel de client sunt interpretate cu ajutorul componentei event-list-component. Preferințele în privința obținerii evenimentelor pot fi setate în pagina Settings a aplicației. Detalii despre eveniment cat și acțiuni suplimentare asupra acestuia pot fi găsite în pagina EventDetails care se deschide în momentul în care realizăm un click pe card-ul ce reprezintă evenimentul. Dispunem de următoarele acțiuni, :vizualizarea detaliilor despre eveniment, nivelul de zgomot de la locația la care are loc evenimentul cat și starea vremii pentru următoarele 7 zile pentru zona respectivă cat și posibilitate de a vedea pe hartă unde are loc acel eveniment.



(a) Pagina cu Evenimente

(b) Pagina cu detalii a evenimentului

Figura 18: Vizualizarea evenimentelor din apropiere de utilizator

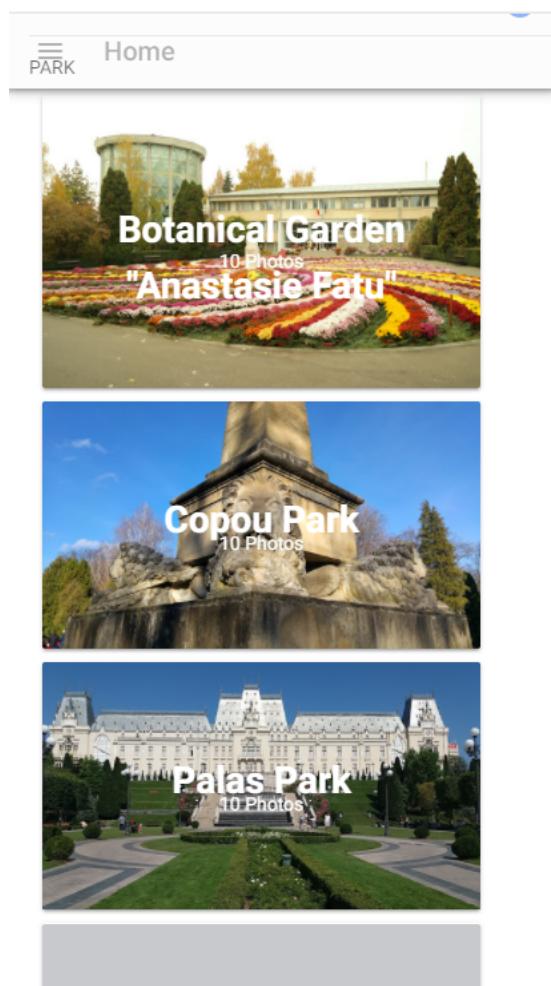
## 7 Use Case

Să urmărim niște cazuri în care am putea folosi aplicația creată în viața reală

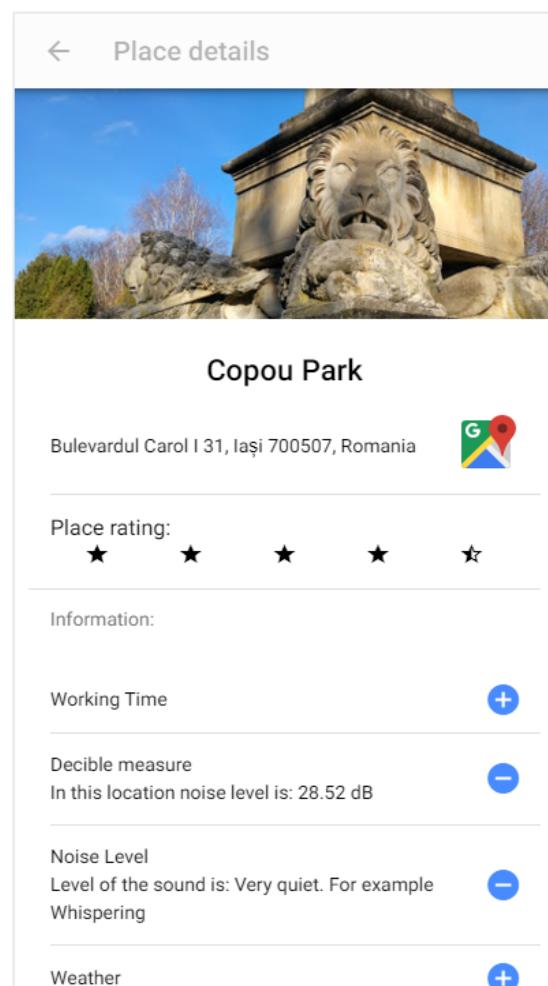
### 1. O plimbare?

Să ne imaginăm că suntem o mămică și vrem să ieșim o plimbare împreună cu copilul. Prima dorință a noastră ar fi ca copilul să nu fie deranjat de mediul înconjurător și ca acesta să poată avea o plimbare liniștită alături de mama.

Am putea merge la plimbare în cel mai apropiat parc, dar nu cunoaștem nivelul de zgomot în zona acestuia, dar cu ajutorul aplicației "Toward a Smart City" putem afla nivelul de zgomot la locația acestuia. Având la dispozitie această informație putem proteja copilul de la un stres în cazul în care în acel moment în apropiere de parc se realizează lucrări cu un nivel ridicat de zgomot.



(a) Parcul Copou



(b) Detaliile de la locația parcului

Figura 19: O plimbare în parcul Copou

## 2. Sunteți nou în oraș?

Sunteți nou în oraș și nu cunoașteți nimic despre acesta, care sunt locurile cele mai importante din apropierea dumnevoastră? Librăriile, muzeele, parcurile, universitățile, cafenelele, restaurantele și alte localuri. Unde puteți merge?

Cum puteți ajunge?

Și ce știm despre locul în care ai ales să mergeți?

La întrebările formulate mai sus vă poate răspunde aplicația creată. În pagina Home avem locurile din oraș după categorii figura 17b. Realizând un "Tap" asupra unui local se va deschide pagina cu detalii al acestuia figura 20a unde putem vedea orele în care localul este deschis precum și informațiile colectate în zona în care se află localul. Tot aici putem vedea locul pe hartă și să trecem în aplicația "Google Maps" ca să alegem drumul pe care putem ajunge acolo figura 20c.

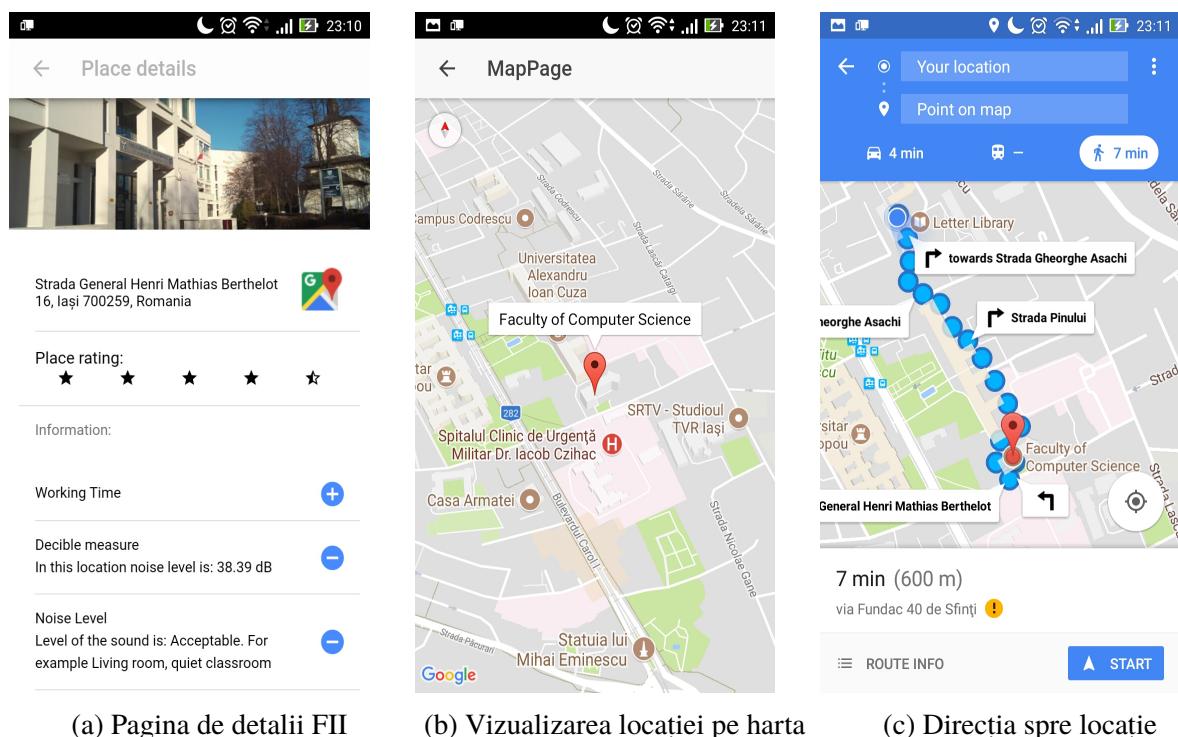
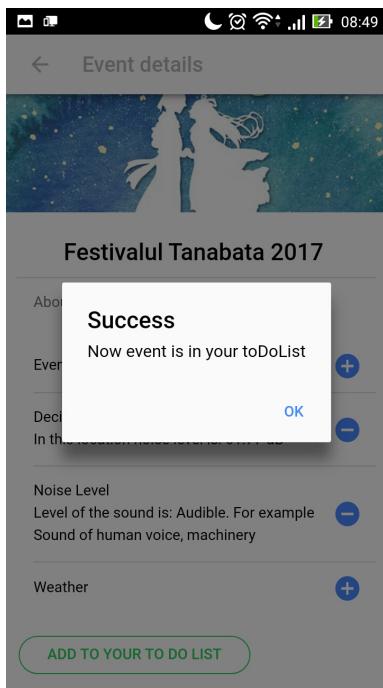


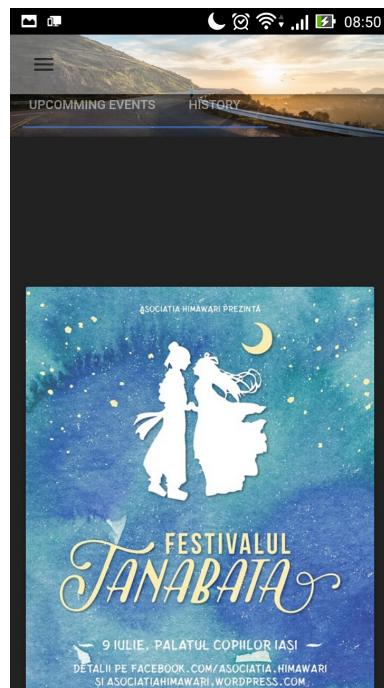
Figura 20

## 3. Plan?

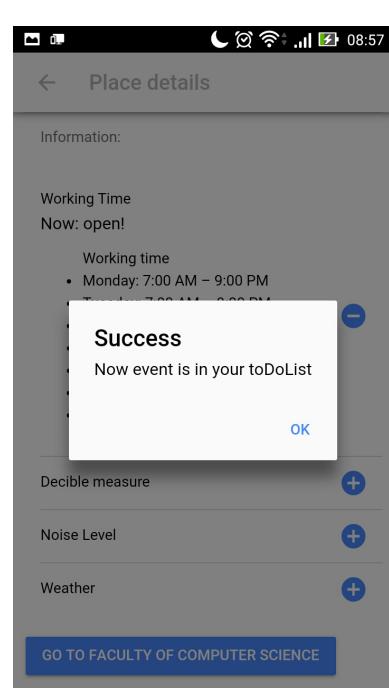
Am creat special pentru comoditate un To-Do-List în care utilizatorul își poate adăuga evenimentele la care dorește să meargă precum și localuri la care planifică o vizită, putem observa asta în figurile 21,. Utilizarea acesteia simplifică managementul timpului de care și asa utilizatorul duce lipsă.



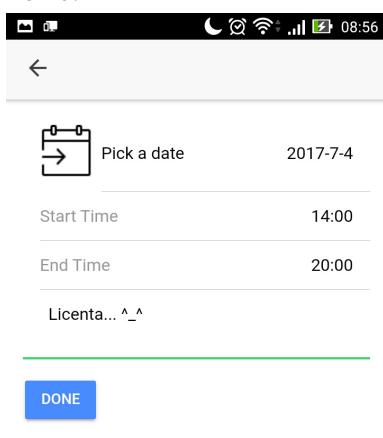
(a) Adăugarea unui eveniment la ToDo.



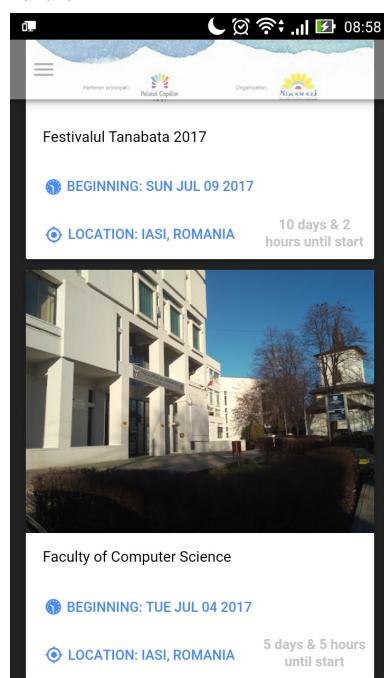
(b) Evenimentul în lista utilizatorului.



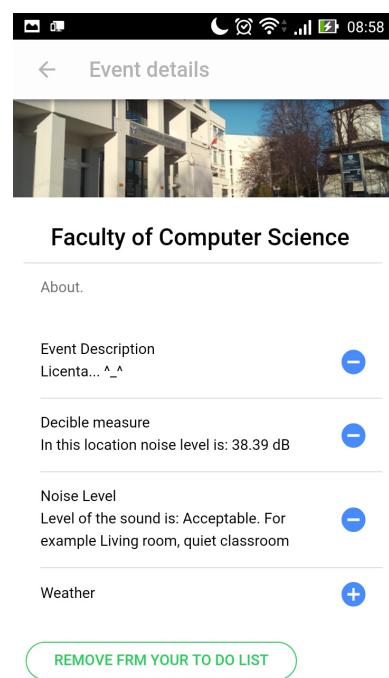
(c) Adăugare de vizită la locație.



(d) Detalii despre evenimentul utilitorului.



(e) Evenimentul utilizatorului la FII.



(f) Detalii despre evenimentul utilizatorului la FII

Figura 21: Adăugarea la To Do List

## 8 Concluzie

În cadrul realizării acestei lucrări a fost creată o platformă ce aduce orașul mai aproape de cetățean. Aplicația pe care o folosește acesta îi dă posibilitatea de a afla evenimentele ce au loc în apropierea lui precum și localurile din împrejurime, dar cel mai important lucru este că pentru fiecare eveniment și local este oferită informație referitoare la starea mediului (zgomot, temperatură, umiditate, presiune) de la locația acestuia. Platforma este construită modular astfel încât să fie posibilă adăugarea de noi module și prin urmare funcționalitățile acesteia să fie extinse. Pas cu pas vom aduce mai aproape de un *Oraș Inteligent*.

## 9 Bibliografie

- [1] *Why cities need to become smart now.* URL: <http://www.iec.ch/smartcities/>.
- [2] Matt Hamblen. “Just what IS a smart city?” In: (Oct 1, 2015). doi: <http://www.computerworld.com/article/2986403/internet-of-things/just-what-is-a-smart-city.html>.
- [3] Peter Williams. “What, Exactly, is a Smart City?” In: (Jul 13, 2016). doi: <http://meetingoftheminds.org/exactly-smart-city-16098>.
- [4] James Manyika. “Unlocking the potential of the Internet of Things”. In: (June 2015). doi: <http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/the-internet-of-things-the-value-of-digitizing-the-physical-world>.
- [5] Kavitha Gopalan. “Smart Cities – Urbanization through IoT”. In: (July 21, 2015). doi: <http://blogs.mobodexter.com/smart-cities-urbanization-through-iot/>.
- [6] *IoT ONE Index: Smart Cities Around The World.* URL: <https://www.iotone.com/guide/iot-one-index-smart-cities-around-the-world/g647>.
- [7] *Smart Citize Platform.* URL: <https://smartcitizen.me>.
- [8] *Ionic Preface.* URL: <https://ionicframework.com/>.
- [9] *Angular 2 and TypeScript - A High Level Overview.* URL: <https://www.infoq.com/articles/Angular2-TypeScript-High-Level-Overview>.
- [10] *What is Redis?* URL: <https://aws.amazon.com/elasticsearch/what-is-redis/>.
- [11] *MongoDB.* URL: <http://searchdatamanagement.techtarget.com/definition/MongoDB>.
- [12] *Database as a service for MongoDB.* URL: [https://azuremarketplace.microsoft.com/en-us/marketplace/apps/Microsoft.DocumentDB\\_MongoDB\\_Support?tab=Overview](https://azuremarketplace.microsoft.com/en-us/marketplace/apps/Microsoft.DocumentDB_MongoDB_Support?tab=Overview).
- [13] *Welcome to Azure Cosmos DB.* URL: <https://docs.microsoft.com/en-us/azure/cosmos-db/introduction>.
- [14] *Amazon Elastic Compute Cloud (Amazon EC2).* URL: <https://aws.amazon.com/ec2/>.
- [15] *Guide to Using MGRS Coordinates.* URL: [https://www.maptools.com/tutorials/mgrs/quick\\_guide](https://www.maptools.com/tutorials/mgrs/quick_guide).

- [16] *MGRS DATA*. URL: <http://mgrs-data.org/>.
- [17] *Military Grid Reference System*. URL: <https://mappingsupport.com/p/gmap4.php?tilt=off&mgrs=14SPG34308382&z=5&t=t1>.
- [18] Rakesh Shrestha. “Angular 2: A Component-Based MVC Framework”. In: (November 11, 2016). doi: <https://dzone.com/articles/angular-2-a-component-based-mvc-framework>.
- [19] Irvin Rangel. “Ionic Framework the easy way!” In: (Dec 15, 2016). doi: <http://inherit.mx/en/blog/technology/ionic-framework-the-easy-way/#.WUy8RGiGOHs>.
- [20] *Architecture Overview*. URL: <https://angular.io/guide/architecture>.

# Appendices

## A Setarea mediului de lucru

Mediul pe care a fost dezvoltată și testată funcționalitatea acestei lucrări a fost *Windows 10* și *Ubuntu 16.04*. drept încapsulare a dependentilor am folosit Node Package Manager

### A.1 Instalare Node Package Manager

#### A.1.1 Petru utilizatorii de Unix

```
1 user@name:~$ sudo apt-get update  
2 user@name:~$ sudo apt-get install nodejs  
3 user@name:~$ sudo apt-get install npm
```

bash

#### A.1.2 Pentru utilizatorii de Windows

”<https://nodejs.org/dist/v6.10.3/node-v6.10.3-x64.msi>”

## A.2 Android SDK

### A.2.1 Configurarea pe Ubuntu

Instalare Java

```
1 user@name:~$ sudo apt-get update  
2 user@name:~$ sudo dpkg --add-architecture i386  
3 user@name:~$ sudo apt-get install libbz2-1.0:i386  
4 user@name:~$ sudo apt-get install libc6:i386  
5 libncurses5:i386 libstdc++6:i386 lib32z1  
6 user@name:~$ sudo apt-get install openjdk-8-jdk
```

bash

Adaugă JAVA\_HOME prin intermediul fișierului de configurare /.bashrc

```
1 user@name:~$ export JAVA_HOME=/usr/lib/jvm/java-8-openjdk
```

bash

Instalare Android Studio Descărcați arhiva ZIP pentru Linux de pe link-ul:

”<https://developer.android.com/studio/install.html>”

- move the .zip to /opt
- extract it
- chown the folder to your name
- chmod 777 studio.sh and run it for the installer

După ce sdk-ul de android este instalat în ~/Android/Sdk. Este de preferat sa adaugăm următoarele cai în variabila de sistem PATH

```
1 user@name:~$ export PATH=${PATH}:~/Android/Sdk/tools  
2 user@name:~$ export PATH=${PATH}:~/Android/Sdk/platform-tools
```

bash

Rulați comanda android, deschideți Tools -> Manage AVDs și asigurați-vă ca instalati o versiune de sdk mai mare de android-23 și verificați sa existe directorul în /Android/Sdk/platforms/

### A.2.2 Configurarea pe Windows

Instalați ultima versiune de **Java JDK** (NU doar JRE).

Setam variabilele de sistem pentru JAVA\_HOME care referă la folder-ul în care a fost instalat Java JDK. dacă ai instalat Java JDK în folder-ul default C:\Program Files\Java\jdk8, atunci JAVA\_HOME ar trebui sa refere la aceasta cale. După ce ați setat variabila JAVA\_HOME trebuie sa adăugați la variabila PATH calea către folder-ul bin. Având în vedere configurarea de mai sus aceasta ar trebui sa arate în felul următor %JAVA\_HOME%\bin

Android SDK

Instalare Android Studio. Detalii despre aceasta găsiți pe pagina oficială Android.

Nu uitam să setam variabila de sistem

ANDROID\_HOME (C:\Users<username>\AppData\Local\Android\Sdk) care referă către calea unde a fost instalat SDK-ul de Android. La fel este recomandat ca directoarele:

- tools\bin
- platform-tools

din ANDROID\_HOME sa fie adăugate și ele în variabila PATH %ANDROID\_HOME%\tools ...

## B Configurare Ionic

### B.1 Configurări preleminare

- Node.js v6
- npm v3

### B.2 Instalam ultima versiune de cordova și ionic cu ajutorul *npm*

```
1 user@name:~$ npm install -g cordova ionic
```

bash

## C Plugin-uri pentru Ionic

### C.1 DB Meter

Acest plugin permite accesarea microfonului ca să pot colecta pentru a determina nivelul de zgomot

```
1 user@name:~$ ionic cordova plugin add cordova-plugin-dbmeter  
2 user@name:~$ npm install --save @ionic-native/db-meter
```

bash

## C.2 Geolocation

Acest plugin permite accesarea modulului GPS din cadrul dispozitivului mobil.

```
1 user@name:~$ ionic cordova plugin add cordova-plugin-geolocation  
2 user@name:~$ npm install --save @ionic-native/geolocation
```

bash

## D Rularea Aplicatiei

Pentru început trebuie îndepliniti pașii de mai sus.

```
1 user@name:~$ npm install  
2 user@name:~$ npm start
```

bash