ADAM Marian Master TAID

DUMITRAȘCU Andrei Aplicații Software pentru Terminalul Mobil

ȘERBĂNESCU Andrei

Aplicație de asistență pentru nevăzători

**1.Breviar teoretic**

Mobilitatea este una din principalele probleme întâmpinate de oamenii cu deficiențe de vedere în viața de zi cu zi. De-a lungul timpului au fost folosite diferite metode de a ușura modul de deplasare al acestora și de a oferi un anumit grad de siguranță în deplasare, folosind fie bețe de ghidare, fie câini special instruiți. Însă, anumite situații limită pot duce la rătăcirea unei persoane căreia orientarea în spațiu folosind diferite repere nu îi este la îndemână. Odată cu evoluția tehnologiei, diferite aplicații se pot dezvolta în scopul asistării acestor persoane.

Scopul acestui proiect este dezvoltarea unei soluții ce vine atât în sprijinul persoanelor cu deficiențe de vedere, cât și în sprijinul persoanelor ce se ocupă de îngrijirea acestor persoane. Soluția propusă este formată din două aplicații, fiecare având funcționalitate specifică. Pentru persoana ce se ocupă de întreținere este util ca, în orice moment, să poată localiza persoana respectivă. De asemenea, este util ca, în cazul în care localizarea nu este foarte exactă, să poată avea acces la camera dispozitivului pentru a putea observa diferite repere în fotografii și identifica poziția exactă.

Totuși, nu se poate asigura o supraveghere permanentă a persoanei. În acest sens, este esențial ca persoana cu deficiențe de vedere să poată trimite o alarmă în momentul în care consideră ca s-a pierdut.

**2. Funcționalitate**

Cum a fost menționat anterior, soluția propune două aplicații. Aplicația destinată persoanelor cu deficiențe de vedere oferă următoarele funcționalități:

* Preia locația în timp real, înregistrând orice modificare în coordonatele GPS, pe care o încarcă în baza de date oferită de Google Firebase, locație ce va fi preluată mai departe de aplicația complementară
* Oferă două metode de alertare în caz de pierdere: un buton pe ecran de SOS sau folosirea butoanelor de volum în acest sens
* În cazul trimiterii unei alerte sunt efectuate mai multe acțiuni: în funcție de opțiunea utilizatorului, se realizează un număr de fotografii la intervale mici de timp se sunt stocate în baza de date Firebase Storage, de unde sunt preluate de a doua aplicație. De asemenea, se trimite o notificare pe al doilea dispozitiv pentru a informa întreținătorul că persoana s-a pierdut

Cea de a doua aplicație vine sa interpreteze informațiile primite de la prima, oferind o interfață prin care întreținătorul poate verifica persoana cu deficiențe de vedere, prin:

* Prima funcționalitate, și cea mai importantă, este afișarea locației dispozitivului pe hartă în timp real. De asemenea, se poate stabili un traseu ce, odată încălcat, să fie trimisă o alertă care să anunțe acest lucru.
* Într-o altă activitate pot fi vizualizate ultimele fotografii realizare cu dispozitivul persoanei cu dizabilități, fotografii preluate din baza de date. Numărul de fotografii poate fi editat de utilizator.
* Preia mesajele de alertă venite de la primul dispozitiv.

**3. Explicarea codului**

1. Aplicația nevăzătorului

**MainActivity.java**

* **sign**(): funcție ce realizează autentificarea user-ului
* folosind **LocalBroadcastManager** se preiau informațiile de latitudine și longitudine și sunt încărcate în baza de date
* **captureButton**.**setOnClickListener** este folosit pentru butonul de SOS. În momentul apăsării butonului este trimisă notificarea către cel de al doilea dispozitiv folosind câmpul din baza de date „SOS”. După asta, folosind un ’’for’’, se realizează fotografiile. **mCamera**.**startPreview**()inițializează camera, iar metoda **mCamera**.**takePicture**() realizează fotografiile. Pentru a permite salvarea pozelor și încărcarea acestora în baza de date, este folosită o întârziere între realizarea acestora.
* **onPictureTaken(byte[] data, Camera camera**)este o metodă folosită după realizarea fotografiei și primește ca parametri informația despre fotografie și obiectul cameră. Se declară **timeStamp** pentru a denumi fotografiile în funcție de ora la care au fost realizate. Urmează apelarea funcției **getOutputMediaFile(String timeStamp)**, de tip File, ce primește ca parametru **timeStamp** și returnează un fișier gol. Mai departe se va scrie în acest fișier informația despre fotografie prin **fos.write(data)** și se apelează funcția **uploadFile(newUri, timeStamp**, ce încarcă fotografia în baza de date.
* **getOutputMediaFile(String timeStamp):**se încearcă crearea unui director unde se vor salva pozele, în cazul în care acesta nu există deja. Se setează în variabila **mediaStorageDir** calea către acest director și se creează un fișier gol cu numele **IMG+timeStamp.jpg** în acea cale.
* **uploadFile(Uri imagUri, String timeStamp):** se verifică dacă imaginea există și se face o referință pentru Firebase Storage către locul unde va fi salvată. Imaginea este încărcată folosind **sref.putFile(imagUri**), fiind implementat și un **addOnSuccessListener** ce este folosit în momentul în care încărcarea este finalizată. Astfel, se generează URL-ul de download al imaginii, acesta fiind încărcat în Firebase Database. Mai departe, URL-urile imaginilor încărcate vor fi preluate de a doua aplicație pentru a le afișa.
* **public boolean onKeyLongPress(int keyCode,KeyEvent event)** este alternativa pentru butonul de SOS. Metoda primește ca parametru tasta ce este apăsată, în acest caz tasta de micșorare a volumului. Evenimentul ce declanșează alerta este apăsarea lungă a butonului de volum, pentru a fi evitată declanșarea prin apăsarea involuntară. Logica din interiorul butonului de volum este aceeași ca și în cazul butonului SOS
* în onResume se apelează aceleași metode pentru a asigura continuitatea aplicației în cazul în care este minimizată
* **startStep1()** verifică prezența GooglePlayServices ce sunt folosite pentru locație, fiind, în caz afirmativ, apelată startStep2()
* **startStep2()** verifică dacă există conexiune la internet. În caz contrar, cere utilizatorului să pornească conexiunea. Urmează verificarea permisiunilor de locație
* **startStep3()** pornește serviciul de localizare ce rulează cât timp aplicația nu este închisă complet (eliberată din memoria RAM)

În clasa **CameraPreview.java** este inițializat cadrul în care va apărea imaginea venită de la cameră. De asemenea, sunt stabiliți parametri camerei, cum ar fi dimensiunea cadrului, tipul de focus, sau FrameRate-ul. De asemenea, sunt implementați constructorii și destructorii, pentru a inițializa suprafața de afișare a camerei la pornirea aplicației și distrugerea ei în momentul închiderii.

1. Aplicatia întreținătorul

**DisplayActivity.java**

Activitate este responsabila cu preluarea coordonatelor GPS din baza de date,si afisarea lor pe harta.

* In **onCreate()** este creat un **SupportMapFragment**,o componenta care reprezinta cea mai simpla metoda de a introduce o harta in cadrul aplicatiei.Aceata componenta este un fragment,si poate fi adaugata in layout-ul activitatii in XML.Este apelata apoi metoda onMapReady astfel: **mapFragment.getMapAsync(this)**.
* in **onMapReady(GoogleMap googleMap)** este creat obiectul mMap din clasa **GoogleMap**.Un obiect **GoogleMap** poate fi instantiat doar folosind metoda getMapAsync pe un **MapFragment**.
* **subscribeToUpdates():** este creata o referinta la baza de date,in locatia denumita „Locatie” si se obtine o instanta **DataSnapshot**,care contine date din baza de la acea locatie.Este creat un **ValueEventListener**,care monitorizeaza schimbarile in baza de date de la locatia respectiva si este declansat atunci cand datele se modifica.
* **setMarker(final DataSnapshot dataSnapshot)** este o metoda in care sunt sunt citite valorile pentru latitudine si longitudine in doua variabile,folosind **dataSnapshot.child().getValue().toString()**. Acestea sunt folosite pentru a crea o instanta **LatLng**,care reprezinta o pereche de coordonate de latitudine si longitudine. Este implementat un **OnMapClickListener()** pe harta.Atunci cand utilizatorul atinge harta intr-un anumit punct,se va executa codul din acesata metoda.Sunt preluate coordonatele punctului respectiv si se creeaza un nou marker,precum si un cerc cu centrul in acest punct si cu o raza a carei valoare este introdusa de catre utilizator intr-un EditText.Cercul are rolul de a stabili un perimetru de siguranta.Se adauga un marker pe harta la locatia respectiva si este animata camera astfel incat marke-ul sa fie in centrul hartii. Fiecare marker creat este adaugat in **HashMap<Double,Marker>**. Cand este creat un now marker,se extrage vechiul marker din HashMap si se sterge apleand **marker.remove()**,astfel incat pe harta sa ramana doar cel mai recent marker.Acelasi lucru se intampla si in cazul cercului,acesta fiind stocat intr-o lista List<Circle> . Tot in **OnMapClickListener()** este creat un intent catre clasa **NotificationService**,fiind transmise in cadrul acestuia coordonatele centrului cercului precum si raza acestuia.

**NotificationService.java**

Este un serviciu care ruleaza in fundal si care are rolul de a trimite o notificare in doua situatii:

1. Nevazatorul a iesit din interiorul cercului definit de catre întreținător
2. Nevazatorul a apasat butonul de SOS

Mai intai se citesc valorile coordonatelor pentru centrul cercului si raza acestuia,trimise prin intent din DisplayActivity.java,folosind **intent.getStringExtra()** . Se extrag apoi coordonatele GPS la care se afla nevazatorul,din baza de date in acelasi mod ca in DisplayActivity.java. Se extrage si o valoare denumita SOS,care este True doar atunci cand nevazatorul apasa butonul de SOS. Se calculeaza distanta dintre cele doua seturi de coordonate,si daca este mai mare decat raza cercului,se trimite o notificare. Se verifica daca valoarea campului SOS din baza de date este True,caz in care se trimite notificare.Este creat un **PendingIntent** care are rolul de a deschide activitatea DisplayActivity() atunci cand utilizatorul apasa pe notificarea primita.

**PicturesActivity.java**

Este o activitate care are rolul de a descarca pozele din baza de date si de a le afisa.

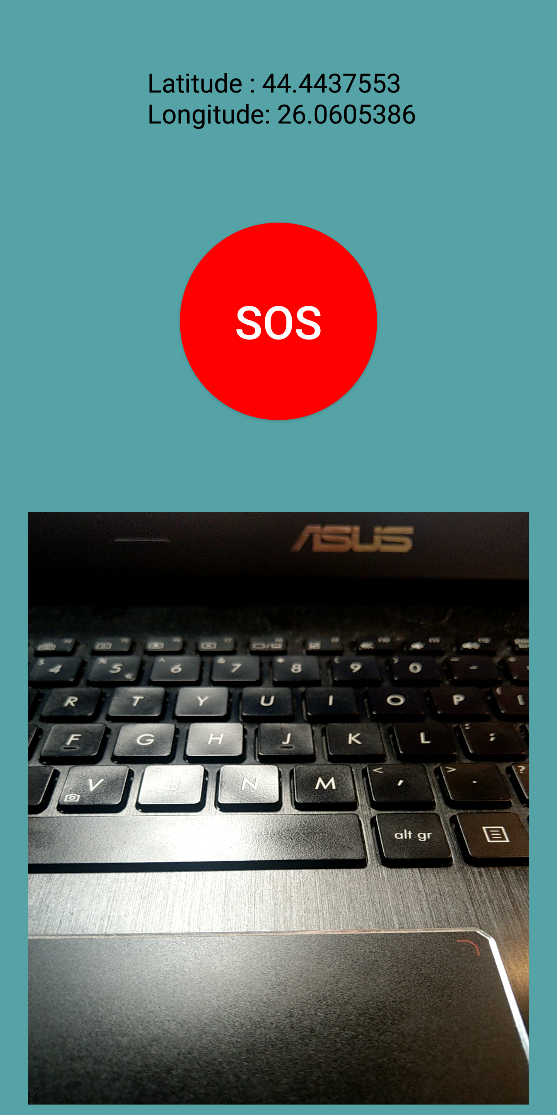
* **buttonAction(View view)** este o metoda care se executa atunci cand utilizatorul apasa pe butonul Display,dupa ce a introdus numarul de poze in EditText.Este creata o referinta la baza de date la locatia „downloadURL” ,unde se afla adresele URL de la care vor fi descarcate pozele.Aceste adrese sunt stocate in **ArrayList<String>** . Valoarea introdusa de utilizator in EditText este utilizata in **limitToLast(value)**,pentru a seta numarul de valori care vor fi extrase din baza de date,incepand de la cele mai recente.

Pentru a afisa pozele se foloseste **RecyclerView** impreuna cu un adapter care primeste ca date de intrare ArrayList-ul.**Adapter**-ul este responsabil cu asigurarea accesului la date si cu crearea unui View pentru fiecare element din setul de date.

**Adapter.java**

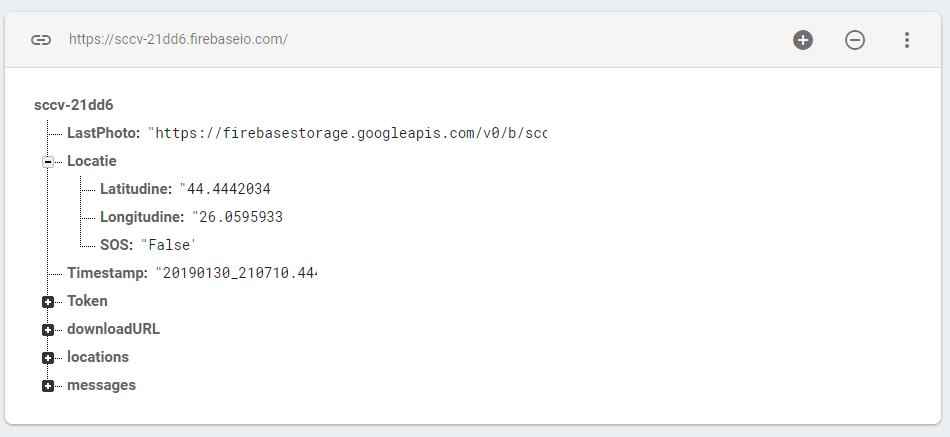
* **public Adapter.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType)** este o metoda care creaaza un ViewHolder pentru fiecare element din ArrayList,si anume un TextView in care este specificate data la care a fost facuta poza,si un PhotoView pentru a afisa poza.Spre deosebire de ImageView,**PhotoView** permite si operatii precum zoom in poza.
* **public void onBindViewHolder(final ViewHolder holder, int position)** este o metoda in care sunt descacate pozele de la adresa URL folosind libraria Glide,si se extrage data din numele poze.

**4. Capturi de ecran**



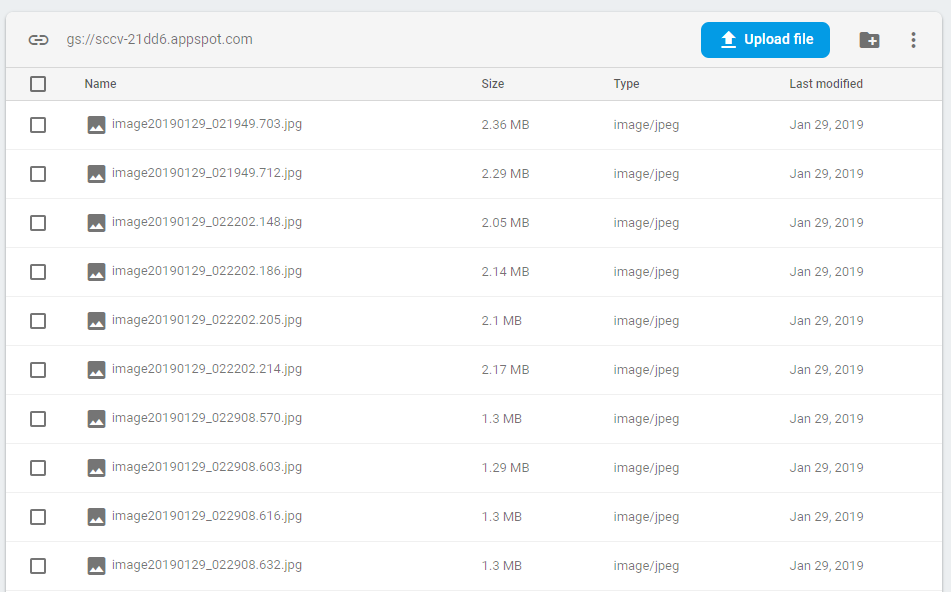
Aplicația destinată nevăzătorilor prezintă o interfață destul de simplă, în care sunt afișate, în special pentru etapa de dezvoltare a aplicației, coordonatele locației și imaginea preluată de la cameră. Butonul de alertare este suficient de mare, însă, prin eliminarea cadrului de imagine, se poate mări și plasa central, fiind mai ușor de văzut.

Structura bazei de date este următoarea:



Câmpul LastPhoto conține URL-ul ultimei poze încărcate în baza de date. Câmpul Locație conține coordonatele GPS și câmpul folosit pentru alertarea trimisă pe device-ul întreținătorului. Câmpul downloadURL conține toate URL-urile pozelor încărcate în Firebase Storage, în timp ce câmpul locations conține parametri folosiți în poziționarea pe hartă a locației primite de la device-ul nevăzătorului.

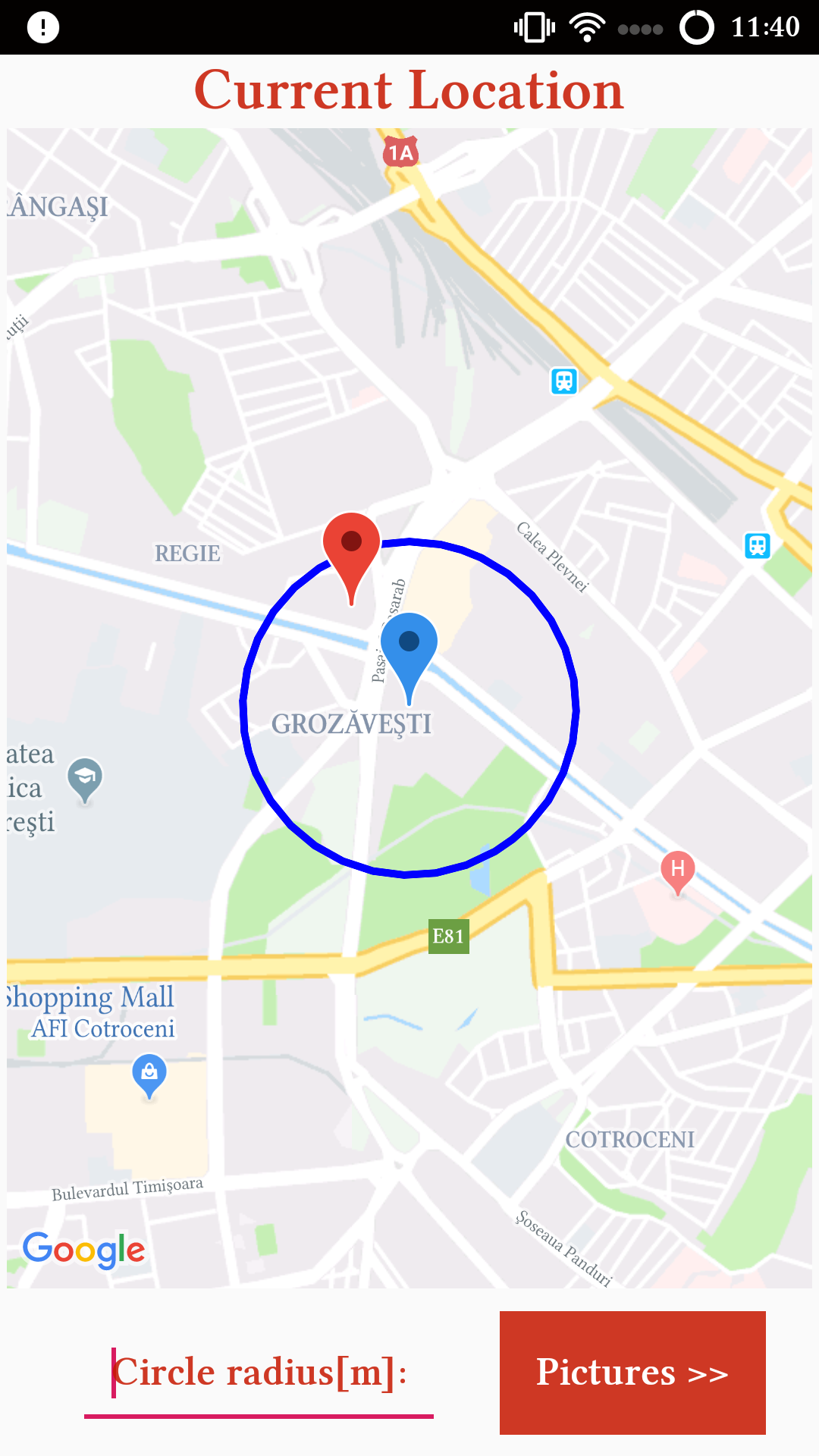
În Firebase Storage, pozele sunt încărcate astfel:



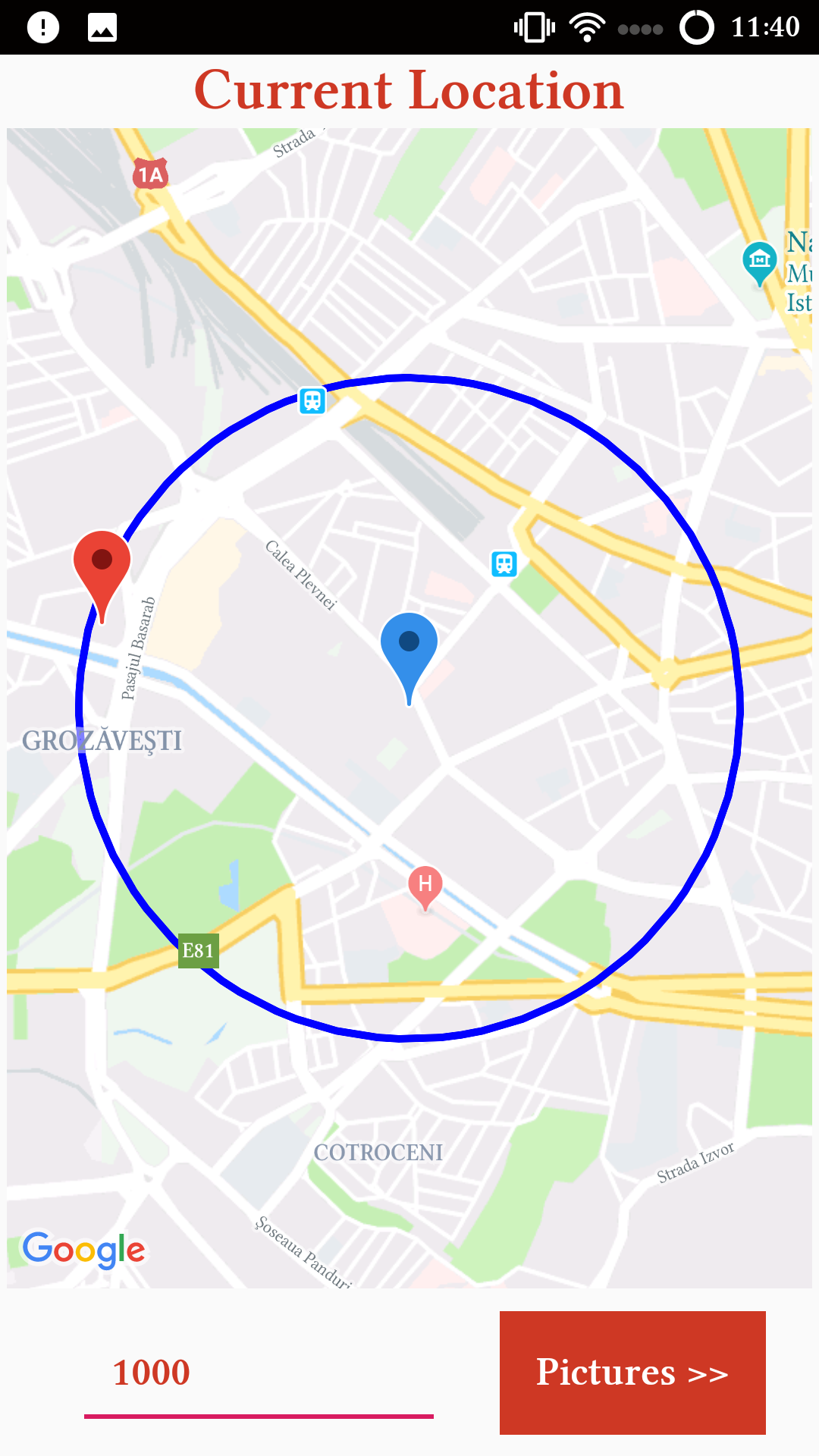
Pozitia curenta a nevazatorului:



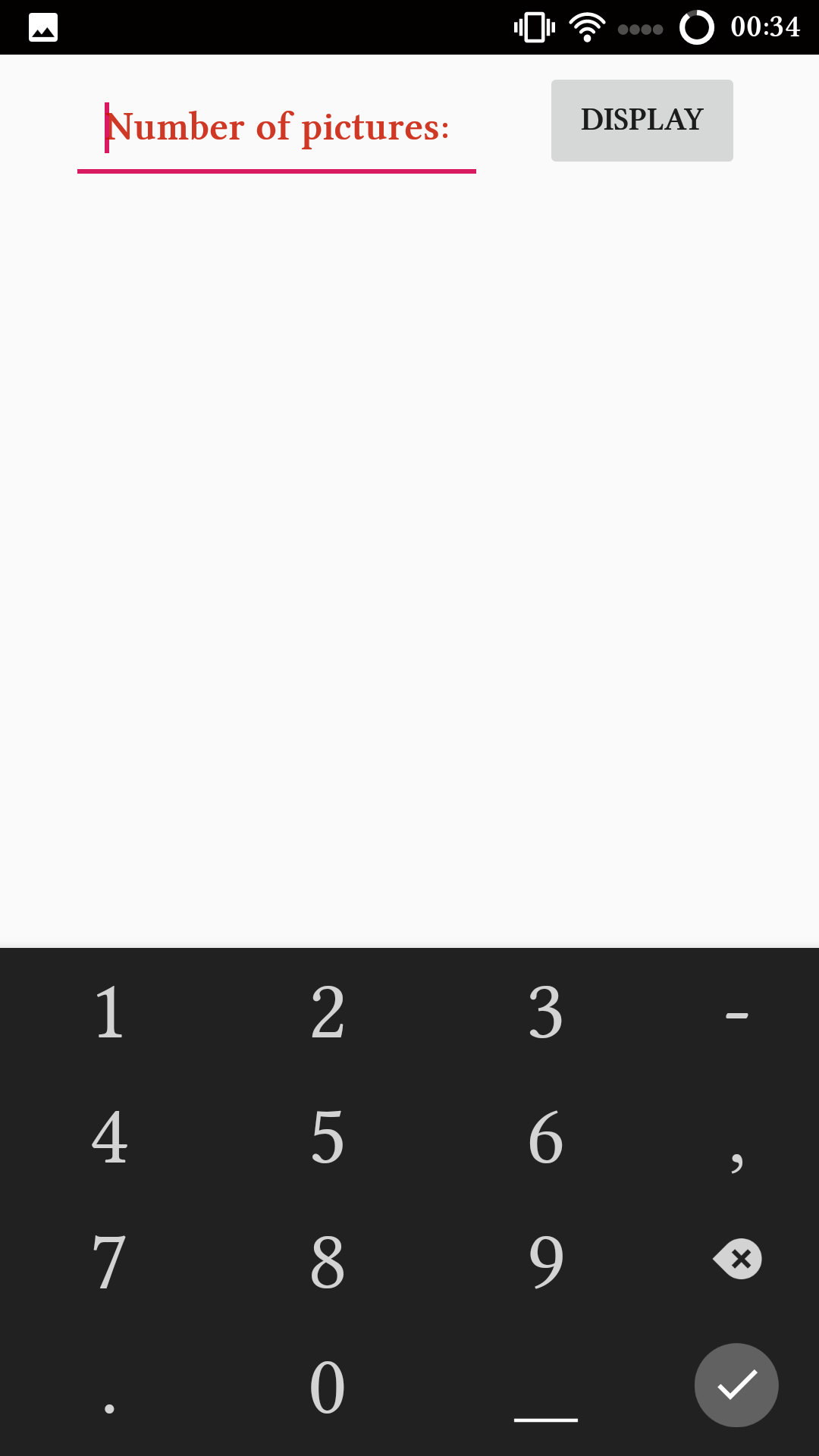
Cer cu raza implicita de 500m



Cerc cu raza de 800m:



Selectare numar de poze:



Afisare poze:



Zoom in poza:

