**Waldo Low-level Design**

**1. Descriere sumară a funcţionalităţilor**

Aplicaţia detectează magazine de bicicletă din aproprierea utilizatorului. Pentru aceasta, trebuie să aibă datele pornite şi GPS-ul. Magazinele sunt afişate sub forma unei liste. Dacă se dă click pe un magazin, se deschide harta cu poziţia lui şi a utilizatorului. Dacă doreşte, poate primi comenzi vocale de navigaţie până la magazin prin intermediul Google Maps.

Din aplicaţie se pot şi adăuga magazine de bicicletă. Pe moment, datele introduse de utilizator vor fi trimise către un server, iar adăugarea efectivă se va face printr-un script care interacţionează cu interfaţa Google.

Cazurile de utilizare, precum şi scheme ale ecranelor se găsesc pe Dropbox.

**2. Clase şi metode**

**2.1 SplashScreen.java**

Primul ecran care se afişează utilizatorului este logoul Waldo. Clasa aferentă este SplashScreen.java, aflată în pachetul waldo.bike.waldo. Logica este următoarea: la deschiderea aplicaţiei se verifică să fie pornite datele şi activat GPS-ul. Verificarea efectivă se face prin apelul metodelor *checkGpsEnabled()*, respectiv *checkInternetConnection()* din Utilities.DeviceConnection.java. Acest lucru se întâmplă în metoda nativă sistemului onResume(), apelată la pornirea activităţii. Dacă oricare din cele două condiţii nu este îndeplinită, se afişează un mesaj de avertizare prin intermediul metodelor *showInternetDisabledAlertToUser()* şi *showGPSDisabledAlertToUser()*. Mesajele afişează şi un buton la a cărui apăsarea se deschide ecranul corespunzător pentru activare datelor sau GPS-ului.

Dacă ambele sunt activate, se intră direct în activitatea principală (MainActivity.java). Acest lucru se face prin apelul metodei startActivity(Context c) din onResume() sau onReceive() din NetworkChangeReceiver. Metoda onReceive() din această clasă are rolul de "a asculta" reţeaua - dacă se schimbă starea (se activează Internetul), mesajul de avertizare aferent este închis şi se porneşte ecranul principal, verificându-se în prealabil dacă şi GPS-ul este pornit (booleana isGPSEnabled). Această metodă este necesară, deoarece datele mobile nu se conectează instant. La întoarcerea în aplicaţie se apelează din nou onResume() care verifică starea datelor, dar ele sunt în curs de conectare. În momentul în care se conectează, metoda onReceive() din NetworkChangeReceiver este apelată.

Pentru aceasta, am definit în AndroidManifest.xml un receptor:

<receiver android:name="waldo.bike.waldo.SplashScreen$NetworkChangeReceiver">

<intent-filter>

<action android:name="android.net.conn.CONNECTIVITY\_CHANGE"/>

</intent-filter>

</receiver>

**Atenţie!** Dacă va fi vreodată nevoie să se schimbe activitatea în care se verifică starea datelor mobile, trebuie schimbat şi numele receptorului pentru a indica acea clasă în care "se ascultă" (android:name="waldo.bike.waldo.SplashScreen$NetworkChangeReceiver"). De asemenea, a se avea în vedere că orientarea acestei activităţi este mereu portrait, iar acest lucru este specificat în manifest: android:screenOrientation="portrait".

**2.2 Utilities.DeviceConnection**

Conţine metode ce verifică accesul la reţea ori locaţie. Numele metodelor sunt intuitive şi oferă o idee legat de ce fac.

**2.3. Utilities.Constants**

Toate constantele folosite în aplicaţie. Nu include mesajele de eroare/informare, care vor fi/sunt trecute în values/strings.xml

**2.4. Utilities.Utility**

Clasă ce conţine diverse funcţii ajutătoare - de formatare, de extragere a preferinţelor utilizatorului etc.

**2.5. waldo.bike.waldo.MainActivity**

Activitatea principală din aplicaţie. La pornire, se face conectarea la GPS (fiind deja pornit din activitatea precedentă). În onCreate() se defineşte, iar în onStart() se conectează. Clasa implementează interfeţele GoogleApiClient.ConnectionCallbacks, GoogleApiClient.OnConnectionFailedListener și LocationListener pentru partea de GPS. În onConnected(), apelată de sistem după ce se iniţializează conexiunea, se stabilesc detaliile de conectare (precizia - înaltă, frecvenţa actualizării - o secundă) şi se cer actualizări ale locaţiei. Acestea sunt transmise în onLocationChanged(), unde se poate converti la String.

Am luat în calcul varianta în care utilizatorul iese din aplicaţie şi închide GPS-ul. Tratăm acest caz în onResume(), apelată când MainActivity revine pe ecran, prin apelarea lui checkGpsEnabled() din DeviceConnection.

Comportamentul GPS la schimbare ecranului

Am definit variabila previousOrientation pentru a controla comportamentul în caz că orientarea ecranului se schimbă. Este inițializată cu 0, iar în onCreate() ia valoarea orientării actuale a ecranului (landscape = 2; portrait = 1) prin apelarea metodei getScreenOrientation(). În onPause(), care se apelează când altă activitate apare pe ecran,previousOrientation ia valoarea 4, iar deconectarea se face în onStop() doar dacă previousOrientation == 4 (adică e altă activitate pe ecran). Vezi stările unei activități pentru a înțelege complet logica: http://developer.android.com/reference/android/app/Activity.html .

Starea rețelei

Clasa MainNetworkReceiver din MainActivity este cea care "ascultă" la schimbările de rețea. Momentan doar afișăm un Toast când se schimbă starea. Mai încolo, vom avea o bandă care va apărea în partea de sus a ecranului. De asemenea, dacă se trece de la WiFi la date sau viceversa nu se afișează acel mesaj de reconectare. Facem acest lucru prin definirea variabilei previousNetworkState, care e inițializată ca fiind CONNECTED la pornirea activității (pentru că verificăm ca telefonul să fie într-adevăr conectat în SplashScreen.java). Întreaga logică se află în onReceive(). De asemenea, nu afișăm nici un mesaj la prima încărcare a activității. Controlăm comportamentul prin booleanul firstLoad,inițializat cu *true* la pornirea activității și făcut *false* la prima schimbare a stării rețelei.

Slider Menu

Din Main Activity se încarcă şi meniul care glisează în ecran. Codul aferent este pus în onCreate() şi semnalizat. Celelalte metode folosite de meniu şi semnalizate ca atare se află spre finalul clasei. Comportamentul la click este gestionat în SlideMenuClickListener.

**2.6. waldo.bike.waldo.SettingsActivity**

Clasa aferentă ecranului de setări. De aici se pot face trei lucruri de către utilizator: stabilirea razei pe care se detectează magazine de bicicletă, a unităţii de măsură şi dacă să primească notificări de la noi. În onCreate() este apelată bindPreferenceSummaryToValue, care modifică interfaţa în funcţie de alegerea făcută de utilizator.

**2.7 Utilities.GlobalState**

Clasă folosită pentru a stoca variabilele globale.

**2.8 Places. FetchGooglePlaces**

Clasa este folosită pentru a accesa API-ul Google. Momentan, acest lucru se face printr-un AsyncTask, însă vom migra la un serviciu pe parcurs. Apelul efectiv se face în doInBackground; getPlaceDataFromJson() este folosită pentru a prelucra datele în format JSON întoarse de Google, iar rezultatul este publicat în onPostExecute. Aici sunt scrise în adaptor (mShopsAdapter), care este folosit în ShopsFragment pentru a popula lista.

**2.9 waldo.bike.waldo. ShopsFragment**

Fragmentul ataşat activităţii principale. În onCreateView() se populează lista. Adaptorul este populat din printr-un SimpleCursorAdapter, care interoghează baza de date printr-un URI (vezi în directorul "Tutoriale" arhitectura acestuia). updateShopList() actualizează lista de magazine printr-o sincronizare instantă (SyncAdapter.syncImmediately(getActivity());).

Momentan, avem un buton de refresh care apelează updateShopList(), declanşând sincronizarea. În viitor, vom elimina acest buton.

**2.10 Pachetul sync SyncAdapter**

Pachetul de aducere a datelor de pe server în aplicaţie. Am renunţat la vechea implementare de AsyncTask în favoarea acesteia. onPerformSync este apelat din syncImmediately, care la rândul lui este apelat de fiecare dată când nevoie de reîmprospătarea listei de magazine. În onPerformSync se face apelul de Google API şi prelucrarea datelor în format JSON. Pentru a funcţiona, adaptorul are nevoie de un serviciu de autentificare şi de un "autentificator" - de faţadă în cadrul nostru, nu sunt folosite -, şi de un serviciu de sincronizare.

**2.11 Pachetul data**

Conține codul pentru baza de date. Aceasta este de tip SQLite. Numele coloanelor și URI-urile sunt stocate în ShopsContract, iar în ShopsDbHelper comportamentul la crearea bazei de date (comanda CREATE TABLE) și la actualizare (DROP TABLE IF EXISTS + table\_name).

**2.12 MapsActivity**

The map used when displaying all of the shops or only one. The origin of the call to the map is identified in setUpMap(), and the map is populated accordingly. If there is only one shop, the data regarding it is transferred through a bundle. Otherwise, we extract the data from the database via the ContentProvider.