Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corso di Laurea Magistrale in Ingegneria delle Comunicazioni

Tecniche Audiovisive

"Scouty: applicazione Android per lo scautismo"

Prof. Gianni Orlandi

Prof. Andrea Proietti

Christian Chiummariello

Introduzione

Lo scautismo (o scoutismo) è un movimento a carattere non partitico, aperto a tutti senza distinzione di origine, razza e fede religiosa, nato da un'idea di Sir Robert Baden - Powell, barone di Gilwell, più noto come Baden-Powell o semplicemente B.P.

Oggi il movimento scout è diffuso a livello mondiale e, contando più di quaranta milioni di iscritti, è una delle più grandi organizzazioni di educazione non formale. Scopo dello scautismo, fondato sul volontariato, è l'educazione dei giovani a un civismo responsabile mediante lo sviluppo delle proprie attitudini fisiche, morali, sociali e spirituali. Il metodo educativo si basa sull'*imparare facendo* attraverso attività all'aria aperta e in piccoli gruppi.

Dinanzi alla possibilità di cimentarmi nello sviluppo di una applicazione nella piattaforma Android, l'ormai celeberrimo sistema operativo per dispositivi mobili sviluppato da Google Inc., i ricordi della mia infanzia legati alle esperienze trascorse nel mondo degli scout mi hanno suggerito l'idea di scrivere un applicativo per smartphones e tablets che rendesse di immediata disponibilità gli strumenti maggiormente utili nell'ambito dello scautismo.

E' nata così "Scouty" , app che mette a disposizione dell'utente, registratosi con un proprio account personale, una torcia, una bussola, una camera, un riproduttore mp3 per il richiamo degli uccelli, una rapida chiamata a numeri di emergenza ed una piattaforma di registrazione e gestione degli itinerari percorsi visualizzabili su mappa.

Logo di Scouty

Creazione del progetto

L' applicazione è stata sviluppata utilizzando l'ambiente di sviluppo integrato (IDE), realizzato da Google, noto come Android Studio.

Logo di Android Studio

Come minima versione di Android sulla quale l'applicazione è installabile è stata scelta la versione 2.3.3 Gingerbread (API level 10) mentre come versione di Android per la compilazione è stata scelta la versione 5.1 Lollipop (API level 22).

Logo di Android Gingerbread

Logo di Android Lollipop

Login/Logout & Registra

Si è realizzato un sistema completo di registrazione e login in Android utilizzando PHP, MySQL e SQLite.

Per interagire con un database MySQL bisogna prima costruire una API (Application Programming Interface) REST. Il compito della API REST è quello di ottenere la richiesta dal client, interagire con il database e dare infine la risposta al client. Pertanto si è innanzitutto creata una semplice PHP, MySQL API la quale svolge le seguenti azioni:

1.Accetta richieste nei metodi GET / POST;

2.Interagisce con il database attraverso l'inserimento / recupero dei dati.

3.Restituisce infine una risposta in formato JSON

Si è quindi creato un account sul sito "it.altervista.org" per ottenere la disponibilità di un database denominato "my_tesinatav". Si è aperto phpmyadmin e si sono eseguite le seguenti queries per creare la tabella "users" necessaria per immagazzinare le informazioni di login degli utenti.

```
create table users(
id int(11) primary key auto_increment,
unique_id varchar(23) not null unique,
name varchar(50) not null,
email varchar(100) not null unique,
encrypted_password varchar(80) not null,
salt varchar(10) not null,
created_at datetime,
updated_at datetime null
);
```

Successivamente si sono scritti i seguenti files di estensione php:

- Config.php

```
<?php
/**
* Variabili di configurazione del database
*/
define("DB_HOST", "localhost");
define("DB_USER", "tesinatav");
define("DB_PASSWORD", "");
define("DB_DATABASE", "my_tesinatav");
?>
```

- DB_Connect.php in cui si gestiscono apertura e chiusura della connessione al database.

```
<?php
class DB_Connect {
private $conn;
// Connessione al database
public function connect() {
require_once 'Config.php';
// Connessione al database mysql
$this->conn = new mysqli(DB_HOST, DB_USER, DB_PASSWORD,
DB_DATABASE);
// restituzione del gestore del database
return $this->conn;
}
}
?>
```

- DB_Functions.php il quale contiene le funzioni per immagazzinare un utente nel database ed ottenere un utente dal database; si utilizzano:

  1.id univoco – si genera l'id univoco dell'utente in php utilizzando la funzione uniqid(", true);

  2.Password Crittografata – le passwords sono immagazzinate utilizzando il metodo base64_encode. Ciascuna password necessita di due colonne per essere immagazzinata nel database: l'una per immagazzinare la password crittografata, l'altra per immagazzinare il "sale" (salt) utilizzato per crittografare la password.

```
<?php
class DB_Functions {
private $conn;
// costruttore
function __construct() {
require_once 'DB_Connect.php';
// connessione al database
$db = new Db_Connect();
$this->conn = $db->connect();
}
// distruttore
function __destruct() {
}
/**
* Immagazzinamento nuovo utente
* restituisce i dettagli d'utente
*/
```

```
public function storeUser($name, $email, $password) {
$uuid = uniqid('', true);
$hash = $this->hashSSHA($password);
$encrypted_password = $hash["encrypted"]; //password crittografata
$salt = $hash["salt"]; // sale
$stmt = $this->conn->prepare("INSERT INTO users(unique_id, name,
email,
encrypted_password, salt, created_at) VALUES(?, ?, ?, ?, ?,
NOW())");
$stmt->bind_param("sssss", $uuid, $name, $email,
$encrypted_password, $salt);
$result = $stmt->execute();
$stmt->close();
// controllo del successo dell'immagazzinamento
if ($result) {
$stmt = $this->conn->prepare("SELECT * FROM users WHERE email
= ?");
$stmt->bind_param("s", $email);
$stmt->execute();
$user = $stmt->get_result()->fetch_assoc();
$stmt->close();
return $user;
} else {
return false;
}
}
/**
* Ottenere l'utente mediante email e password
*/
public function getUserByEmailAndPassword($email, $password) {
$result = mysqli_query($this->conn,"SELECT * FROM users WHERE
email = '$email'") or
die(mysqli_connect_errno());
// controllo del risultato
$no_of_rows = mysqli_num_rows($result);
if ($no_of_rows > 0) {
$result = mysqli_fetch_array($result);
$salt = $result['salt'];
$encrypted_password = $result['encrypted_password'];
$hash = $this->checkhashSSHA($salt, $password);
// controllo della password
if ($encrypted_password == $hash) {
return $result;
}
} else {
return false;
}
}
/**
* Controllo che l'utente esista o no
*/
public function isUserExisted($email) {
$stmt = $this->conn->prepare("SELECT email from users WHERE email
= ?");
$stmt->bind_param("s", $email);
$stmt->execute();
$stmt->store_result();
if ($stmt->num_rows > 0) {
// utente esistente
```

```
$stmt->close();
return true;
} else {
// utente non esistente
$stmt->close();
return false;
}
}
/**
* Password crittografata
* @param password
* restituisce sale e password crittografata
*/
public function hashSSHA($password) {
$salt = sha1(rand());
$salt = substr($salt, 0, 10);
$encrypted = base64_encode(sha1($password . $salt, true) . $salt);
$hash = array("salt" => $salt, "encrypted" => $encrypted);
return $hash;
}
/**
* Decrittografare la password
* @param salt, password
* restituisce stringa hash
*/
public function checkhashSSHA($salt, $password) {
$hash = base64_encode(sha1($password . $salt, true) . $salt);
return $hash;
}
}
?>
```

- register.php: Si crea l' endpoint per la registrazione utente il quale accetta **name**, **email** e **password** come parametri *POST* ed immagazzina l'utente nel database MySQL.

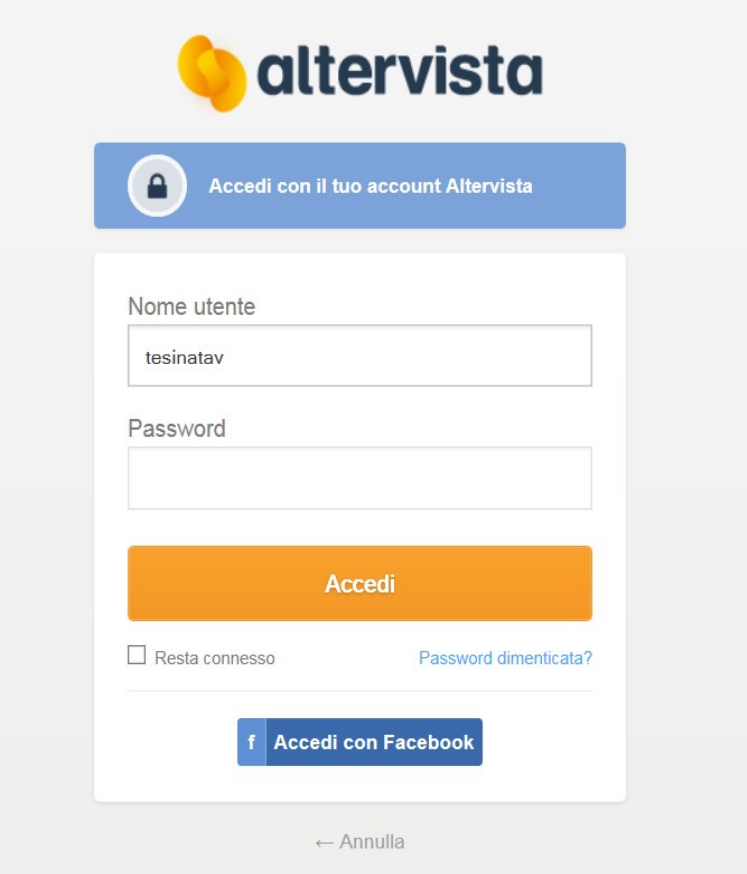
```
<?php
require_once 'DB_Functions.php';
$db = new DB_Functions();
// array di risposta json
$response = array("error" => FALSE);
if (isset($_POST['name']) && isset($_POST['email']) &&
isset($_POST['password'])) {
// ricevere i parametri post
$name = $_POST['name'];
$email = $_POST['email'];
$password = $_POST['password'];
// controllo se l'utente già esiste con la stessa email
if ($db->isUserExisted($email)) {
// utente già esistente
$response["error"] = TRUE;
$response["error_msg"] = "User already existed with " . $email;
echo json_encode($response);
} else {
```

```
// crea un nuovo utente
$user = $db->storeUser($name, $email, $password);
if ($user) {
// utente immagazzinato con successo
$response["error"] = FALSE;
$response["uid"] = $user["unique_id"];
$response["user"]["name"] = $user["name"];
$response["user"]["email"] = $user["email"];
$response["user"]["created_at"] = $user["created_at"];
$response["user"]["updated_at"] = $user["updated_at"];
echo json_encode($response);
} else {
// immagazzinamento utente fallito
$response["error"] = TRUE;
$response["error_msg"] = "Unknown error occurred in
registration!";
echo json_encode($response);
}
}
} else {
$response["error"] = TRUE;
$response["error_msg"] = "Required parameters (name, email or
password) is missing!";
echo json_encode($response);
}
?>
```

- login.php: si crea un altro endpoint per il login il quale accetta email e password come parametri POST. Dopo aver ricevuto l' email e la password, esso controlla nel database per l'utente corrispondente. Se l'utente viene abbinato, fa eco il successo della risposta JSON.

```
<?php
require_once 'DB_Functions.php';
$db = new DB_Functions();
// array di risposta json
$response = array("error" => FALSE);
if (isset($_POST['email']) && isset($_POST['password'])) {
// ricevere i parametri post
$email = $_POST['email'];
$password = $_POST['password'];
// ottenere l'utente mediante email e password
$user = $db->getUserByEmailAndPassword($email, $password);
if ($user != false) {
// utente trovato
$response["error"] = FALSE;
$response["uid"] = $user["unique_id"];
$response["user"]["name"] = $user["name"];
$response["user"]["email"] = $user["email"];
$response["user"]["created_at"] = $user["created_at"];
$response["user"]["updated_at"] = $user["updated_at"];
echo json_encode($response);
} else {
```

```
// utente non trovato con le credenziali
$response["error"] = TRUE;
$response["error_msg"] = "Login credentials are wrong. Please try
again!";
echo json_encode($response);
}
} else {
// mancanza dei parametri post richiesti
$response["error"] = TRUE;
$response["error_msg"] = "Required parameters email or password is
missing!";
echo json_encode($response);
}
?>
```



Screenshot altervista

A questo punto nel progetto in Android Studio si è aperto "build.gradle" e si è aggiunta la libreria di supporto "volley" utilizzata per effettuare chiamate HTTP.
Android volley è una networking library la quale è stata introdotta per rendere le chiamate di networking molto più semplici e veloci. Di default tutte le chiamate di rete volley operano in modo asincrono, pertanto non occorre più il ricorso ad asynctask:

```
compile 'com.mcxiaoke.volley:library-aar:1.0.0'
```

Poi si sono create le seguenti classi:
- AppConfig.java in cui si dichiarano gli urls di login e registrazione.

```
// url del login utente server
public static String URL_LOGIN = "http://tesinatav.altervista.org/
login.php";
// url del registra utente server
public static String URL_REGISTER =
"http://tesinatav.altervista.org/register.php";
```

- AppController.java che estende Application e viene eseguita al lancio dell' app e nella quale si iniziano tutti i core objects di "volley".

```
public RequestQueue getRequestQueue() {
    if (mRequestQueue == null) {
        mRequestQueue =
Volley.newRequestQueue(getApplicationContext());
    }
    return mRequestQueue;
}
public <T> void addToRequestQueue(Request<T> req, String tag) {
    req.setTag(TextUtils.isEmpty(tag) ? TAG : tag);
    getRequestQueue().add(req);
}
public <T> void addToRequestQueue(Request<T> req) {
    req.setTag(TAG);
    getRequestQueue().add(req);
}
public void cancelPendingRequests(Object tag) {
    if (mRequestQueue != null) {
        mRequestQueue.cancelAll(tag);
```

```
    }
}
```

- SessionManager.java la quale gestisce i dati di sessione attraverso l'applicazione utilizzando le SharedPreferences. Si immagazzina una boolean flag "isLoggedIn" in shared preferences per verificare lo stato del login.

```
// Preferenze condivise
SharedPreferences pref;
private static final String KEY_IS_LOGGEDIN = "isLoggedIn";
public void setLogin(boolean isLoggedIn) {
    editor.putBoolean(KEY_IS_LOGGEDIN, isLoggedIn);
    // confermare le modifiche
    editor.commit();
    Log.d(TAG, "User login session modified!");
}
public boolean isLoggedIn(){
    return pref.getBoolean(KEY_IS_LOGGEDIN, false);
}
```

- SQLiteHandler.java la quale si occupa di memorizzare i dati utente in un database SQLite. Ogni volta che si ha bisogno di ottenere le informazioni dell'utente loggato, esse si recuperano dal database SQLite invece di fare richiesta al server.

```
// Creazione Tabelle
@Override
public void onCreate(SQLiteDatabase db) {
    String CREATE_LOGIN_TABLE = "CREATE TABLE " + TABLE_USER + "("
            + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NAME + "
TEXT,"
            + KEY_EMAIL + " TEXT UNIQUE," + KEY_UID + " TEXT,"
            + KEY_CREATED_AT + " TEXT" + ")";
    db.execSQL(CREATE_LOGIN_TABLE);
    Log.d(TAG, "Database tables created");
}

/**
 * Immagazzinamento dettagli d'utente nel database
 * */
```

```
public void addUser(String name, String email, String uid, String
created_at) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(KEY_NAME, name); // Nome
    values.put(KEY_EMAIL, email); // Email
    values.put(KEY_UID, uid); // UID
    values.put(KEY_CREATED_AT, created_at); // Creato a
    // Inserimento Riga
    long id = db.insert(TABLE_USER, null, values);
    db.close(); // Chiusura connessione con il database
    Log.d(TAG, "New user inserted into sqlite: " + id);
}
/**
 * Ottenere i dati d'utente dal database
 * */
public HashMap<String, String> getUserDetails() {
    HashMap<String, String> user = new HashMap<String, String>();
    String selectQuery = "SELECT  * FROM " + TABLE_USER;
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);
    // Passare alla prima riga
    cursor.moveToFirst();
    if (cursor.getCount() > 0) {
        user.put("name", cursor.getString(1));
        user.put("email", cursor.getString(2));
        user.put("uid", cursor.getString(3));
        user.put("created_at", cursor.getString(4));
    }
    cursor.close();
    db.close();
    // restituzione utente
    Log.d(TAG, "Fetching user from Sqlite: " + user.toString());
    return user;
}
```

- LoginActivity.java in cui il metodo checkLogin() verifica i dettagli di login sul server effettuando una richiesta http volley.

```
/**
 * funzione per verificare i dettagli di login nel db mysql
 * */
private void checkLogin(final String email, final String password)
{
    // Tag usato per annullare la richiesta
    String tag_string_req = "req_login";
    pDialog.setMessage("Logging in ...");
    showDialog();
    StringRequest strReq = new StringRequest(Method.POST,
            AppConfig.URL_LOGIN, new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            Log.d(TAG, "Login Response: " + response.toString());
            hideDialog();
```

```
                try {
                    JSONObject jObj = new JSONObject(response);
                    boolean error = jObj.getBoolean("error");
                    // Controllo del nodo errore in json
                    if (!error) {
                        // utente loggato con successo
                        // Crea sessione di login
                        session.setLogin(true);
                        // Ora immagazzina l'utente in SQLite
                        String uid = jObj.getString("uid");
                        JSONObject user = jObj.getJSONObject("user");
                        String name = user.getString("name");
                        String email = user.getString("email");
                        String created_at = user
                                .getString("created_at");
                        // Inserimento riga nella tabella users
                        db.addUser(name, email, uid, created_at);
                        // Lancio dell'activity principale
                        Intent intent = new Intent(LoginActivity.this,
                                MainActivity.class);
                        startActivity(intent);
                        finish();
                    } else {
                        // Errore in login. Messaggio errore
                        String errorMsg = jObj.getString("error_msg");
                        Toast.makeText(getApplicationContext(),
                                errorMsg, Toast.LENGTH_LONG).show();
                    }
                } catch (JSONException e) {
                    // Errore JSON
                    e.printStackTrace();
                    Toast.makeText(getApplicationContext(), "Json
error: " + e.getMessage(), Toast.LENGTH_LONG).show();
                }
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                Log.e(TAG, "Login Error: " + error.getMessage());
                Toast.makeText(getApplicationContext(),
                        error.getMessage(), Toast.LENGTH_LONG).show();
                hideDialog();
            }
        }) {
            @Override
            protected Map<String, String> getParams() {
                // Invio parametri all'url di login
                Map<String, String> params = new HashMap<String,
String>();
                params.put("email", email);
                params.put("password", password);
                return params;
            }
        };
        // Aggiunta richiesta alla coda delle richieste
        AppController.getInstance().addToRequestQueue(strReq,
tag_string_req);
    }
```

- RegisterActivity.java in cui

1.registerUser() immagazzina l'utente passando name, email e password al php,mysql server.

2.db.addUser() inserisce l'utente nell' SQLite database una volta che si è registrato con successo.

```
/**
* La funzione per immagazzinare l'utente nel database MySQL
*invierà i parametri(tag, name,email, password)all'url di
*registrazione
* */
private void registerUser(final String name, final String email,
                          final String password) {
    // Tag usato per annullare la richiesta
    String tag_string_req = "req_register";
    pDialog.setMessage("Registering ...");
    showDialog();
    StringRequest strReq = new StringRequest(Method.POST,
            AppConfig.URL_REGISTER, new
Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            Log.d(TAG, "Register Response: " +
response.toString());
            hideDialog();
            try {
                JSONObject jObj = new JSONObject(response);
                boolean error = jObj.getBoolean("error");
                if (!error) {
                    // Utente immagazzinato con successo in MySQL
                    // Ora immagazzina l'utente in sqlite
                    String uid = jObj.getString("uid");
                    JSONObject user = jObj.getJSONObject("user");
                    String name = user.getString("name");
                    String email = user.getString("email");
                    String created_at = user
                            .getString("created_at");
                    // Inserimento riga nella tabella users
                    db.addUser(name, email, uid, created_at);
                    Toast.makeText(getApplicationContext(), "User
successfully registered. Try login now!",
Toast.LENGTH_LONG).show();
                    // Lancio dell'activity di login
                    Intent intent = new Intent(
                            RegisterActivity.this,
                            LoginActivity.class);
                    startActivity(intent);
                    finish();
                } else {
                    // Errore occorso in registrazione. Messaggio
                    // errore
```

```
                    String errorMsg = jObj.getString("error_msg");
                    Toast.makeText(getApplicationContext(),
                            errorMsg, Toast.LENGTH_LONG).show();
                }
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Log.e(TAG, "Registration Error: " +
error.getMessage());
            Toast.makeText(getApplicationContext(),
                    error.getMessage(), Toast.LENGTH_LONG).show();
            hideDialog();
        }
    }) {
        @Override
        protected Map<String, String> getParams() {
            // Invio parametri all'url di registrazione
            Map<String, String> params = new HashMap<String,
String>();
            params.put("name", name);
            params.put("email", email);
            params.put("password", password);
            return params;
        }
    };
    // Aggiunta richiesta alla coda delle richieste
    AppController.getInstance().addToRequestQueue(strReq,
tag_string_req);
}
```

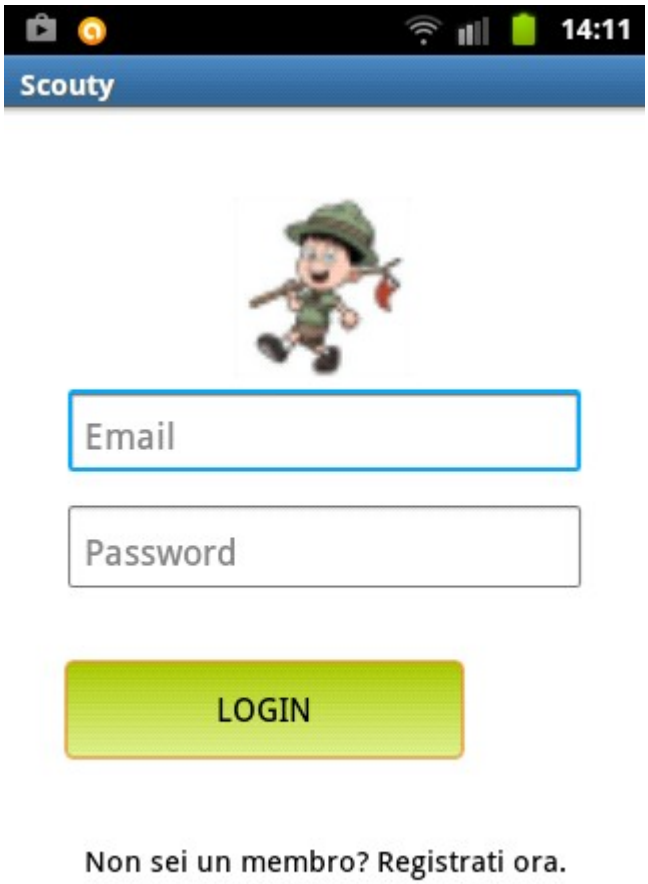
- MainActivity.java aggiunge la schermata per mostrare le informazioni dell'utente loggato. Queste informazioni sono recuperate dal database SQLite una volta che l'utente si è loggato. Inoltre un bottone di logout disconnette l'utente annullando la sessione e cancellando l'utente dalla tabella SQLite.

```
button = (Button) findViewById(R.id.button);
btnLogout = (Button) findViewById(R.id.btnLogout);
// Gestore del database SqLite
db = new SQLiteHandler(getApplicationContext());
// Recuperare i dettagli d'utente da sqlite
HashMap<String, String> user = db.getUserDetails();
String name = user.get("name");
```

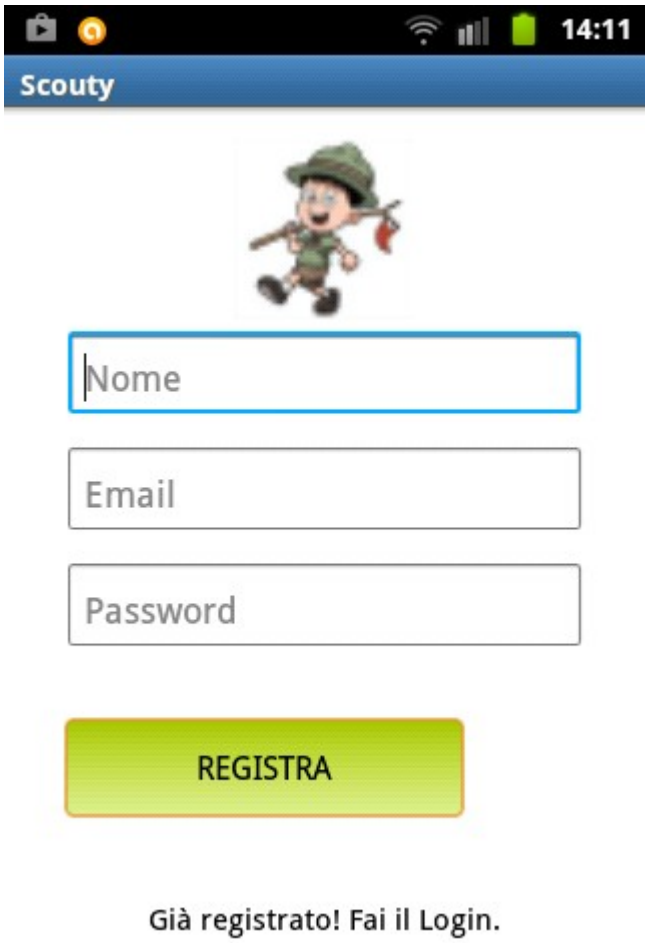
```
// Mostrare i dettagli d'utente sullo schermo
txtName.setText(name);
View.OnClickListener gstr = new View.OnClickListener() {
      public void onClick(View view) {
         switch(view.getId()) {
            case R.id.button:
               startActivity(new
Intent("android.intent.action.PrimaPagina"));
               break;
            case R.id.btnLogout:
               logoutUser();
               break;
         }
      }
   };
   button.setOnClickListener(gstr);
   btnLogout.setOnClickListener(gstr);
}
private void logoutUser() {
   session.setLogin(false);
   db.deleteUsers();
   // Lancio dell'activity di login
   Intent intent = new Intent(MainActivity.this,
LoginActivity.class);
   startActivity(intent);
   finish();
}
```

Infine, naturalmente, si è aggiunto nel AndroidManifest.xml il permesso INTERNET:

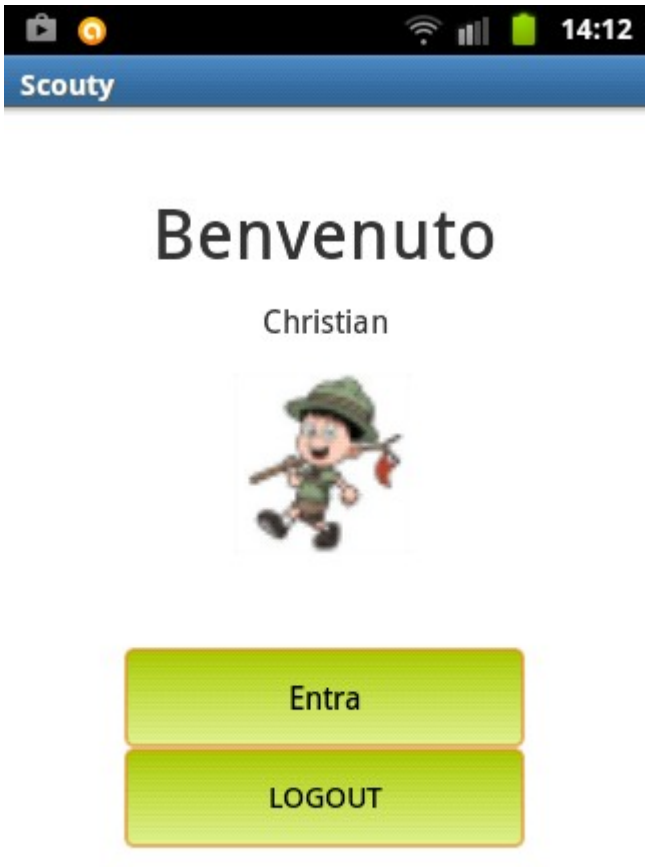
```
<uses-permission android:name="android.permission.INTERNET" />
```



Screenshot Login



Screenshot Registra



Screenshot Entra/Logout

Prima Pagina

La pagina dell'applicazione alla quale si accede premendo il tasto "Entra", dopo aver effettuato il login, mostra i bottoni che consentono all'utente di accedere alle varie funzionalità dell'applicativo.

```
 bottone1 = (Button) findViewById(R.id.bottone1);
 bottone2 = (Button) findViewById(R.id.bottone2);
 bottone3 = (Button) findViewById(R.id.bottone3);
 bottone4 = (Button) findViewById(R.id.bottone4);
 bottone5 = (Button) findViewById(R.id.bottone5);
 bottone6 = (Button) findViewById(R.id.bottone6);

View.OnClickListener gestore = new View.OnClickListener() {
         public void onClick(View view) {
           switch(view.getId()) {
             case R.id.bottone1:
             startActivity(new
Intent("android.intent.action.TorciA"));
             break;
             case R.id.bottone2:
          startActivity(new
Intent("android.intent.action.CamerA"));
          break;
             case R.id.bottone3:
             startActivity(new
Intent("android.intent.action.BussolA"));
             break;
             case R.id.bottone4:
             startActivity(new
Intent("android.intent.action.NaturA"));
             break;
             case R.id.bottone5:
             startActivity(new
Intent("android.intent.action.EmergenzA"));
             break;
         case R.id.bottone6:
            startActivity(new
Intent("android.intent.action.attivita.GestisciPercorso"));
            break;
           }
         }
       };

       bottone1.setOnClickListener(gestore);
       bottone2.setOnClickListener(gestore);
       bottone3.setOnClickListener(gestore);
       bottone4.setOnClickListener(gestore);
       bottone5.setOnClickListener(gestore);
       bottone6.setOnClickListener(gestore);
```

Screenshot Prima Pagina

Torcia

Per realizzare la funzionalità torcia occorre innanzitutto implementare l'accensione e lo spegnimento manuali della luce flash del dispositivo:

- Accensione:

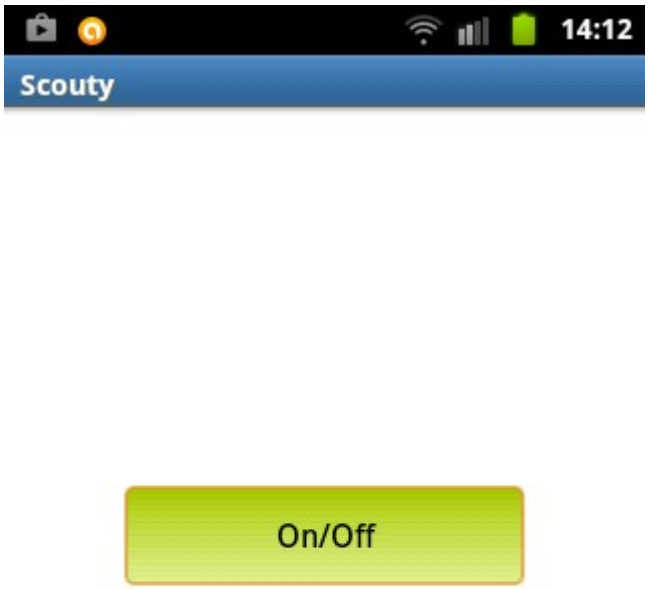
```
camera = Camera.open();
final Parameters p = camera.getParameters();

p.setFlashMode(Parameters.FLASH_MODE_TORCH);

camera.setParameters(p);

camera.startPreview();
```

- Spegnimento:

```
camera = Camera.open();

final Parameters p = camera.getParameters();

p.setFlashMode(Parameters.FLASH_MODE_OFF);

camera.setParameters(p);

camera.stopPreview();
```

Infine bisogna aggiungere i seguenti permessi nel file AndroidManifest.xml:

```
<uses-permission android:name="android.permission.CAMERA" />

<uses-feature android:name="android.hardware.camera" />
```



Screenshot Torcia

Bussola

Perchè la bussola possa funzionare il dispositivo deve essere dotato di un motion sensor. Il codice java che implementa la bussola inizializza il SensorManager nel metodo onCreate() e lo muove ed anima utilizzando il metodo onSensorChanged():

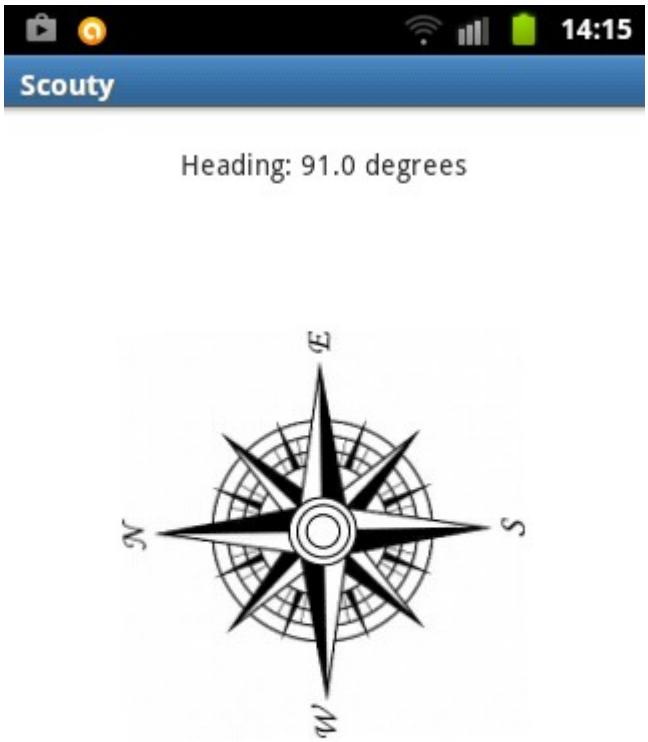
```
mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

@Override
public void onSensorChanged(SensorEvent event) {

    // ottiene l'angolo ruotato rispetto all'asse z
    float degree = Math.round(event.values[0]);
```

```
    tvHeading.setText("Heading: " + Float.toString(degree) + "
degrees");

    // crea un' animazione di rotazione
    RotateAnimation ra = new RotateAnimation(
            currentDegree,
            -degree,
            Animation.RELATIVE_TO_SELF, 0.5f,
            Animation.RELATIVE_TO_SELF,
            0.5f);

    // durata dell'animazione
    ra.setDuration(210);

    // setta l'animazione dopo la fine dello stato di riservazione
    ra.setFillAfter(true);

    // Avvia l'animazione
    image.startAnimation(ra);
    currentDegree = -degree;
}
```

Ulteriori osservazioni:

- Non ci sono speciali permessi da aggiungere nel AndroidManifest.xml;

- Android Sensor Manager utilizza il Polo Nord Magnetico che è diverso dal Polo Nord Geografico.



Screenshot Bussola

Camera

Si è realizzata una funzionalità Camera la quale consente all'utente, una volta scattata la foto, di scegliere se salvarla o eliminarla. Nel primo caso, dopo il salvataggio, viene chiesto all'utente se desidera o meno scattare un' altra fotografia.

Nel metodo onCreate():

```
try {

    // intent per avviare la camera del dispositivo
    Intent intent = new
Intent("android.media.action.IMAGE_CAPTURE");

    // creazione del filename
    String fileName = "immagine";

    // dove vengono salvate le immagini
```

```
    String path = Environment.getExternalStorageDirectory() + "/"
            + fileName + ".jpg";
    File file = new File(path);

Uri outputFileUri = Uri.fromFile(file);

intent.putExtra(MediaStore.EXTRA_OUTPUT, outputFileUri);

//Quando l' activity cessa viene restituito il codice 0 nel metodo
//onActivityResult()
startActivityForResult(intent, 0);
```

Nel metodo onActivityResult():

```
// cancella l'ultima immagine da dcim

if(!deleteLastSavedDcimImage())

// chiede all'utente se vuole scattare un'altra foto

takeAnother();
```

IL metodo takeAnother():

```
public void takeAnother() {

    try {
        new AlertDialog.Builder(this).setTitle("Scouty")
                .setMessage("Vuoi scattare un'altra foto?")
                .setPositiveButton("SI", new OnClickListener() {
                    public void onClick(DialogInterface arg0, int
arg1) {
                        arg0.dismiss();
                        Intent nextActivity = new Intent(
                                CameraActivity.this,
CameraActivity.class);
                        CameraActivity.this.finish();
                        startActivity(nextActivity);
                    }
                })
                .setNegativeButton("NO", new OnClickListener() {
                    public void onClick(DialogInterface arg0, int
arg1) {
                        arg0.dismiss();
                        Toast.makeText(CameraActivity.this,
                                "Fatto.", Toast.LENGTH_LONG)
                                .show();

                        CameraActivity.this.finish();
                    }
                }).show();
```

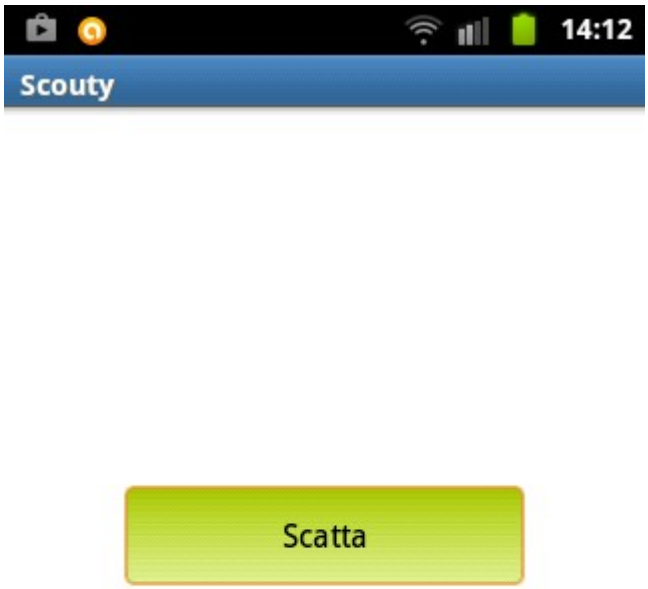
```
    } catch (NullPointerException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Nel metodo deleteLastSavedDcimImage():

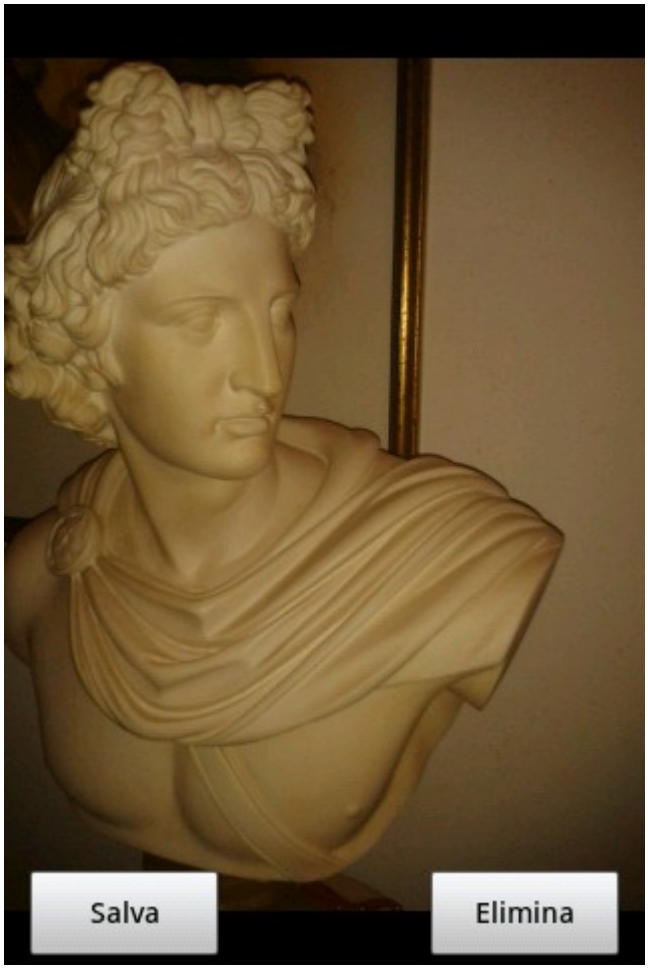
```
try {

      //lista le immagini nella directory /DCIM/Camera del
      //dispositivo
    File[] images = new
File(Environment.getExternalStorageDirectory()
            + "/DCIM/Camera").listFiles();
    File lastSavedImage = images[0];
    int imagesLen =  images.length;

      //loop e check per l'ultima immagine modificata per ottenere
      //l'ultima immagine salvata
    for (int i = 1; i < imagesLen; ++i) {
        if (images[i].lastModified() >
lastSavedImage.lastModified()) {
            lastSavedImage = images[i];
        }
    }

      //quindi cancella l'ultima immagine salvata
    success = new File(lastSavedImage.toString()).delete();
}
```

Infine nel AndroidManifest.xml:

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```



Screenshot Camera fig.1



Screenshot Camera fig.2

Screenshot Camera fig.3

Richiamo degli uccelli

Si sono realizzati quattro bottoni ciascuno dei quali, se cliccato, avvia la riproduzione del verso di un volatile salvato come file mp3. Android supporta musica e suoni attraverso la classe MediaPlayer. Questa classe è disponibile nel package android .media.
Il metodo setVolumeControlStream() viene usato per consentire all'utente di alzare ed abbassare il volume mentre l'applicazione è in esecuzione, indipendentemente dal volume della suoneria.

Quindi innanzitutto nel metodo onCreate() si pone:

```
setVolumeControlStream(AudioManager.STREAM_MUSIC);
```

Una volta fatto ciò, nel metodo OnClick() bisogna creare una istanza MediaPlayer e chiamare il metodo start():

```
public void onClick(View v) {

 int resId;
 switch (v.getId()) {
 case R.id.button_1:
   resId = R.raw.allodola;
   break;
 case R.id.button_2:
    resId = R.raw.fringuello;
   break;
 case R.id.button_3:
    resId = R.raw.merlo;
    break;
 case R.id.button_4:
    resId = R.raw.usignolo;
    break;
 default:
   resId = R.raw.allodola;
   break;
 }

 if (mp != null) {
    mp.release();
 }

//this è il context corrente, resId è l'id della risorsa
//multimediale
 mp = MediaPlayer.create(this, resId);
 mp.start();
}
```

Una volta terminata la riproduzione audio, bisogna rilasciare le risorse assegnate al media player.Ciò può essere fatto nel metodo onDestroy().

```
@Override
protected void onDestroy() {
  if(null!=mp){
 mp.release();
  }
  super.onDestroy();
}
```

Screenshot Richiamo degli uccelli

Emergenza

Si sono realizzati tre bottoni ciascuno dei quali, se cliccato, avvia la chiamata immediata ad un numero di emergenza. Il metodo fondamentale è il metodo Chiama() : esso ricava il numero dalla TextView ed effettua la chiamata tramite l'applicazione interna del sistema.

```
public void Chiama(View view) {
   TextView txtnumero =
(TextView)this.findViewById(R.id.txtnumero);
   Intent intent = new Intent(Intent.ACTION_CALL);
   intent.setData(Uri.parse("tel:" +
txtnumero.getText().toString()));
   startActivity(intent);
}
```

Occorre poi inserire nel AndroidManifest.xml il permesso per poter effettuare la chiamata:

```
<uses-permission android:name="android.permission.CALL_PHONE"/>
```

Screenshot Emergenza

Itinerari

L'applicazione consente di registrare e visualizzare su mappa i percorsi seguiti.

Si è scelto di utilizzare "OpenStreetMap" (OSM)la quale è una mappa del mondo creata da persone comuni e libera da utilizzare secondo una licenza aperta. In particolare ci si è avvalsi della libreria "osmdroid" la cui classe MapView fondamentalmente sostituisce la classe MapView di Google (Google Maps Android API v1). Si sono quindi aggiunte le seguenti dipendenze nel file "build.gradle":

```
compile 'org.osmdroid:osmdroid-android:4.2'
compile 'org.slf4j:slf4j-android:1.6.1-RC1'
```

Per mostrare la mappa con un'icona per lo zoom in ed un'icona per lo zoom out occorre creare un file di layout di estensione xml con:

```
<org.osmdroid.views.MapView
    android:id="@+id/displaytrackmap_osmView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />
<ImageView
    android:id="@+id/displaytrackmap_imgZoomIn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:contentDescription="@string/acc.zoom_in"
    android:src="@drawable/zopiu" >
</ImageView>
<ImageView
    android:id="@+id/displaytrackmap_imgZoomOut"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:contentDescription="@string/acc.zoom_in"
    android:src="@drawable/zomeno" >
</ImageView>
```

Inoltre nel file di estensione java, avente il layout descritto dal precedente file xml, bisogna inserire:

1.osmView.getController() per inizializzare la vista OSM e muovere la mappa su un punto di vista di default

2.osmView.setMultiTouchControls(true) per aggiungere l'abilità di zoomare con 2 dita (multi-touch)

3.osmViewController.zoomIn() e osmViewController.zoomOut() per attivare rispettivamente la funzionalità di zoom in e la funzionalità di zoom out attraverso la pressione sui corrispondenti bottoni

```
// Inizializza la vista OSM
osmView = (MapView) findViewById(R.id.displaytrackmap_osmView);
osmView.setMultiTouchControls(true);
osmViewController = osmView.getController();

// Registra i listeners per i bottoni di zoom
findViewById(R.id.displaytrackmap_imgZoomIn).setOnClickListener( n
ew OnClickListener() {
   @Override
   public void onClick(View v) {
      osmViewController.zoomIn();
   }
});
findViewById(R.id.displaytrackmap_imgZoomOut).setOnClickListener(
new OnClickListener() {
   @Override
   public void onClick(View v) {
      osmViewController.zoomOut();
   }
});
```

Si è scelto di adottare un "TMS" (Tile Map Service) il quale è un protocollo per servire (server) le mappe come piastrelle (tiles) cioè per dividere la mappa verso l'alto in una piramide di immagini a più livelli di zoom. Come Map Tile server si è scelto di utilizzare "OSM Mapnik" il quale, scritto in C++, essenzialmente fornisce la mappa di base OpenStreetMap.

```
return TileSourceFactory.MAPNIK;
```

Per poter registrare un percorso si è reso necessario implementare la geolocalizzazione in Android. La Geolocalizzazione consiste nel rilevare la posizione geografica di un dispositivo esprimendola nelle coordinate bidimensionali denominate latitudine e longitudine, le quali non sono altro che dei numeri frazionari che indicano un punto preciso sulla superficie

terrestre, ipotizzando che questa sia piatta ossia bidimensionale, altrimenti bisognerebbe considerare anche l'altezza.

Il tipo di localizzazione che si è scelto di adottare è il GPS**,** sistema leader nella geolocalizzazione basato su informazioni provenienti da una rete di satelliti, molto accurato e disponibile sulla maggior parte dei dispositivi Android; il suo utilizzo nell'applicazione richiede l'aggiunta del seguente permesso nel AndroidManifest.xml:

```
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Essenzialmente, sfruttare la localizzazione in Android significa gestire un dialogo tra due entità: il LocationManager ed il LocationListener. Il primo, il LocationManager, è un servizio di sistema, rintracciabile mediante Context, che svolge il ruolo di gestore unico dei sistemi di localizzazione. Il modo per reperire il LocationManager è:

```
lmgr=(LocationManager)
context.getSystemService(Context.LOCATION_SERVICE);
```

Quello di LocationListener è il ruolo svolto dall'elemento della nostra applicazione che desidera ricevere informazioni sulla posizione del dispositivo. L'implementazione dell'interfaccia LocationListener comporta l'implementazione obbligatoria di quattro metodi astratti:

- onStatusChanged: riferisce lo stato del provider di localizzazione che si sta utilizzando. Gli stati possibili sono tre, identificati da apposite costanti intere: TEMPORARILY_UNAVAILABLE (temporaneamente non disponibile), OUT_OF_SERVICE (fuori servizio), AVAILABLE (disponibile);
- onProviderDisabled: notifica che il provider è stato disabilitato;
- onProviderEnabled: notifica opposta alla precedente: il provider è stato appena abilitato dall'utente sul dispositivo;

-onLocationChanged è invece il metodo centrale del listener, quello che riceve le informazioni di localizzazione reperite sotto forma di un oggetto di classe Location.

Il modo in cui la localizzazione entra nel progetto Android segue dei passi ben precisi. In sostanza, si recupera un riferimento del LocationManager e gli si chiede di fornire aggiornamenti periodici (in base alle condizioni dettate) sulla posizione del dispositivo. Tutte le informazioni che si recuperano vengono impacchettate all'interno di un oggetto Location. La classe Location rappresenta il tipo di nodo informativo centrale al sistema di localizzazione Android. Esso contiene necessariamente latitudine e longitudine, reperibili mediante i metodi getLatitude e getLongitude.

L'attivazione degli aggiornamenti avviene con la seguente riga di codice:

```
lmgr.requestLocationUpdates(LocationManager.GPS_PROVIDER,   0,   0,
this);
```

I valori passati al metodo rappresentano rispettivamente: la stringa indicante il tipo di provider, il tempo minimo in millisecondi che deve intercorrere tra due aggiornamenti, la distanza in metri minima che deve essere percorsa affinché si abbia un nuovo aggiornamento, l'oggetto che svolge il ruolo di LocationListener.

```
private LocationManager lmgr;
private long lastGPSTimestampStatus = 0;
private long lastGPSTimestampLocation = 0;
private final long gpsLoggingInterval;

if (context instanceof RegistraPercorso) {
   activity = (RegistraPercorso) context;
   lmgr = (LocationManager)
context.getSystemService(Context.LOCATION_SERVICE);
}

public void requestLocationUpdates(boolean request) {
      if (request) {
      lmgr.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0,
0,    this);
      lmgr.addGpsStatusListener(this);
```

```
    } else {
       lmgr.removeUpdates(this);
       lmgr.removeGpsStatusListener(this);
    }
}

@Override
public void onLocationChanged(Location location) {
   if((lastGPSTimestampLocation + gpsLoggingInterval) <
System.currentTimeMillis()){
       lastGPSTimestampLocation = System.currentTimeMillis();
       Log.v(TAG, "Location received " + location);
       if (! gpsActive) {
          gpsActive = true;
          activity.onGpsEnabled();
       }

     }

}

@Override
public void onProviderDisabled(String provider) {
   Log.d(TAG, "Location provider " + provider + " disabled");
   gpsActive = false;
   activity.onGpsDisabled();
}

@Override
public void onProviderEnabled(String provider) {
   Log.d(TAG, "Location provider " + provider + " enabled");
}

@Override
public void onStatusChanged(String provider, int status, Bundle
extras) {
   Log.d(TAG, "Location provider " + provider + " status changed
to: " + status);

   switch (status) {
   case LocationProvider.OUT_OF_SERVICE:
      gpsActive = false;
      activity.onGpsDisabled();
      break;
   case LocationProvider.TEMPORARILY_UNAVAILABLE:
      gpsActive = false;
      break;
   }
}
```

A questo punto è stata creata una classe che estende la classe SQLiteOpenHelper. SQLiteOpenHelper è una classe di supporto per gestire

le attività di creazione del database. SQLiteOpenHelper ha due metodi:

1.onCreate(): viene chiamato quando viene creato il database per la prima volta;

2.onUpgrade(): viene chiamato per aggiornare il contenuto del database (aggiungere, modificare,eliminare).

Il metodo execSQL() , invece, viene chiamato per creare una tabella. Il metodo richiede una query in forma di stringa come parametro. Il metodo open ed il metodo close vengono usati rispettivamente per aprire e chiudere la connessione con il database.

```
private static final String SQL_CREATE_TABLE_TRACKPOINT = ""
   + "create table " + Schema.TBL_TRACKPOINT + " ("
   + Schema.COL_ID    + " integer primary key autoincrement,"
   + Schema.COL_TRACK_ID + " integer not null,"
   + Schema.COL_LATITUDE + " double not null,"
   + Schema.COL_LONGITUDE + " double not null,"
   + Schema.COL_SPEED + " double null,"
   + Schema.COL_ELEVATION + " double null,"
   + Schema.COL_ACCURACY + " double null,"
   + Schema.COL_TIMESTAMP + " long not null,"
   + Schema.COL_COMPASS + " double null,"
   + Schema.COL_COMPASS_ACCURACY + " integer null"+ ")";

@Override
public void onCreate(SQLiteDatabase db) {
   db.execSQL("drop table if exists " + Schema.TBL_TRACKPOINT);
   db.execSQL(SQL_CREATE_TABLE_TRACKPOINT);
   db.execSQL(SQL_CREATE_IDX_TRACKPOINT_TRACK);
   db.execSQL("drop table if exists " + Schema.TBL_TRACK);
   db.execSQL(SQL_CREATE_TABLE_TRACK);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
   switch(oldVersion){
   case 1:
      onCreate(db);
      break;
   case 2:
      manageNewStoragePath(db);
   case 3:
      db.execSQL("alter table " + Schema.TBL_TRACK + " add column
" + Schema.COL_DESCRIPTION + " text");
      db.execSQL("alter table " + Schema.TBL_TRACK + " add column
" + Schema.COL_TAGS + " text");
      db.execSQL("alter table " + Schema.TBL_TRACK + " add column
```

```
" + Schema.COL_OSM_VISIBILITY
            + " text default '"+OSMVisibility.Private+"'");
   case 4:
      db.execSQL("alter table " + Schema.TBL_TRACKPOINT + " add
column " + Schema.COL_SPEED + " double null");
   case 5:
      db.execSQL("alter table " + Schema.TBL_TRACKPOINT + " add
column " + Schema.COL_COMPASS + " double null");
      db.execSQL("alter table " + Schema.TBL_TRACKPOINT + " add
column " + Schema.COL_COMPASS_ACCURACY + " integer null");
   }

}
```

Una volta che l'applicazione è stata dotata di un database funzionante, è stato possibile implementare un content provider per gestire l'utilizzo dei dati in esso immagazzinati. Si è quindi creata una nuova classe che estende la classe ContentProvider. In essa è stata creata una variabile privata per contenere una istanza del database ed è stato istanziato il database nel metodo onCreate():

```
private SupportoDB dbHelper;
@Override
public boolean onCreate() {
   dbHelper = new SupportoDB(getContext());
   return true;
}
```

I Content providers operano con dati al livello URI. Le classi Content provider generalmente forniscono alcune costanti pubbliche che possono essere usate dalle applicazioni per identificare i dati che esse vogliono interrogare. Di seguito si riportano le costanti definite nella classe che estende la classe ContentProvider ed utilizzate da classi esterne che utilizzano il Content Provider:

```
public static final Uri CONTENT_URI_TRACK = Uri.parse("content://"
+ AUTHORITY + "/" + Schema.TBL_TRACK);
public static final Uri CONTENT_URI_TRACK_ACTIVE =
Uri.parse("content://" + AUTHORITY + "/" + Schema.TBL_TRACK +
```

```
"/active");
```

Per determinare quali tipi di indirizzi URI sono passati al content provider si è sfruttata una classe di supporto chiamata UriMatcher per definire specifici URI patterns che il content provider supporta:

```
private static final UriMatcher uriMatcher = new
UriMatcher(UriMatcher.NO_MATCH);
static {
   uriMatcher.addURI(AUTHORITY, Schema.TBL_TRACK,
Schema.URI_CODE_TRACK);
   uriMatcher.addURI(AUTHORITY, Schema.TBL_TRACK + "/active",
Schema.URI_CODE_TRACK_ACTIVE);
   uriMatcher.addURI(AUTHORITY, Schema.TBL_TRACK + "/#",
Schema.URI_CODE_TRACK_ID);
   uriMatcher.addURI(AUTHORITY, Schema.TBL_TRACK + "/#/start",
Schema.URI_CODE_TRACK_START);
   uriMatcher.addURI(AUTHORITY, Schema.TBL_TRACK + "/#/end",
Schema.URI_CODE_TRACK_END);
   uriMatcher.addURI(AUTHORITY, Schema.TBL_TRACK + "/#/" +
Schema.TBL_TRACKPOINT + "s", Schema.URI_CODE_TRACK_TRACKPOINTS);
   }
```

Un content provider ha numerosi metodi di cui bisogna fare l'override: query(),delete(),getType(),insert(),update().
In particolare il metodo query()ha cinque parametri, tra cui due arrays. Android SDK ha una classe di supporto che semplifica l'implementazione di questo metodo: la classe SQLiteQueryBuilder.

Innanzitutto si ottiene una nuova istanza della classe SQLiteQueryBuilder. Po si utilizza il metodo setTables() per specificare le tabelle con cui si sta lavorando. Successivamente si utilizza la classe UriMatcher per determinare se la query è per una singola voce (entry) o tutte le voci. Quindi si restituisce il cursore, creato chiamando il metodo query() di SQLiteQueryBuilder. .
Il metodo Cursor.setNotificationUri() infine consente al cursore di sapere

per quale URI del Content Provider è stato creato.

```
@Override
public Cursor query(Uri uri, String[] projection, String
selectionIn, String[] selectionArgsIn, String sortOrder) {
   Log.v(TAG, "query(), uri=" + uri);
   SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
   String selection = selectionIn;
   String[] selectionArgs = selectionArgsIn;

   String groupBy = null;
   String limit = null;

   switch (uriMatcher.match(uri)) {
   case Schema.URI_CODE_TRACK_TRACKPOINTS:
      String trackId = uri.getPathSegments().get(1);
      qb.setTables(Schema.TBL_TRACKPOINT);
      selection = Schema.COL_TRACK_ID + " = ?";

      if (null != selectionIn) {
         selection += " AND " + selectionIn;
      }

      List<String> selctionArgsList = new ArrayList<String>();
      selctionArgsList.add(trackId);

      if (null != selectionArgsIn) {
         for (String arg : selectionArgsIn) {
            selctionArgsList.add(arg);
         }
      }
      selectionArgs = selctionArgsList.toArray(new String[0]);

      selctionArgsList.clear();
      selctionArgsList = null;
      break;
   case Schema.URI_CODE_TRACK_START:
      if (selectionIn != null || selectionArgsIn != null) {

         throw new UnsupportedOperationException();
      }
      trackId = uri.getPathSegments().get(1);
      qb.setTables(Schema.TBL_TRACKPOINT);
      selection = Schema.COL_TRACK_ID + " = ?";
      selectionArgs = new String[] {trackId};
      sortOrder = Schema.COL_ID + " asc";
      limit = "1";
      break;
   case Schema.URI_CODE_TRACK_END:
      if (selectionIn != null || selectionArgsIn != null) {

         throw new UnsupportedOperationException();
      }
      trackId = uri.getPathSegments().get(1);
      qb.setTables(Schema.TBL_TRACKPOINT);
      selection = Schema.COL_TRACK_ID + " = ?";
      selectionArgs = new String[] {trackId};
```

```
            sortOrder = Schema.COL_ID + " desc";
            limit = "1";
            break;
        case Schema.URI_CODE_TRACK:
            qb.setTables(TRACK_TABLES);
            if (projection == null)
                projection = TRACK_TABLES_PROJECTION;
            groupBy = TRACK_TABLES_GROUP_BY;
            break;
        case Schema.URI_CODE_TRACK_ID:
            if (selectionIn != null || selectionArgsIn != null) {

                throw new UnsupportedOperationException();
            }
            trackId = uri.getLastPathSegment();
            qb.setTables(TRACK_TABLES);
            if (projection == null)
                projection = TRACK_TABLES_PROJECTION;
            groupBy = TRACK_TABLES_GROUP_BY;
            selection = Schema.TBL_TRACK + "." + Schema.COL_ID + " = ?";
            selectionArgs = new String[] {trackId};
            break;
        case Schema.URI_CODE_TRACK_ACTIVE:
            if (selectionIn != null || selectionArgsIn != null) {

                throw new UnsupportedOperationException();
            }
            qb.setTables(Schema.TBL_TRACK);
            selection = Schema.COL_ACTIVE + " = ?";
            selectionArgs = new String[]
{Integer.toString(Schema.VAL_TRACK_ACTIVE)};
            break;
        default:
            throw new IllegalArgumentException("Unknown URI: " + uri);
        }
        Cursor c = qb.query(dbHelper.getReadableDatabase(), projection,
selection, selectionArgs, groupBy, null, sortOrder, limit);
        c.setNotificationUri(getContext().getContentResolver(), uri);
        return c;
}
```

Per poter utilizzare il content provider si è resa necessaria la sua registrazione nel AndroidManifest.xml attraverso l'aggiunta della seguente sezione:

```
<provider android:name=".database.ContentProviderPercorso"
    android:authorities="com.example.scouty.provider"
    android:exported="false" />
```

Si riportano di seguito i principali usi del content provider da parte di classi esterne alla classe che estende ContentProvider.

- Salvataggio nel database delle nuove informazioni relative ai dettagli di un percorso:

```
/**
* Salva la nuova informazione nel database
* @return false se il salvataggio non ha avuto luogo, true
*altrimenti.
*/
protected boolean save() {
   // Salva le modifiche nel database, se presenti.
   etDescription.setError(null);
   if (fieldsMandatory) {
      if (etDescription.getText().length() < 1) {
         etDescription.setError(getResources().getString(R.string.
trackdetail_description_mandatory));
         return false;
      }
   }

   Uri trackUri =
ContentUris.withAppendedId(ContentProviderPercorso.CONTENT_URI_TR
ACK, trackId);
   ContentValues values = new ContentValues();

   // Salva il campo nome, se modificato, nel database.
   String enteredName = etName.getText().toString().trim();
   if ((enteredName.length() > 0)) {
      values.put(Schema.COL_NAME, enteredName);
   }

   // Tutti gli altri valori aggiornati anche se vuoti
   values.put(Schema.COL_DESCRIPTION,
etDescription.getText().toString().trim());
   values.put(Schema.COL_TAGS,
etTags.getText().toString().trim());
   values.put(Schema.COL_OSM_VISIBILITY,
OSMVisibility.fromPosition(spVisibility.getSelectedItemPosition())
.toString());

   getContentResolver().update(trackUri, values, null, null);
   return true;
}
```

- Registrazione di un Content Observer per l'osservazione di eventuali

modifiche del percorso:

```
//Registra un content observer per eventuali modifiche del
//percorso
getContentResolver().registerContentObserver(
      ContentProviderPercorso.trackPointsUri(currentTrackId),
      true, trackpointContentObserver);
```

- Creazione di un nuovo percorso nel database e sulla scheda SD:

```
/**
* Crea un nuovo percorso, nel DB e sulla SD card
 * @returns l' ID del nuovo percorso
 * @throws CreaEccezionePercorso
 */
private long createNewTrack() throws CreaEccezionePercorso {
   Date startDate = new Date();

   // Crea una entry nella tabella TRACK
   ContentValues values = new ContentValues();
   values.put(Schema.COL_NAME, "");
   values.put(Schema.COL_START_DATE, startDate.getTime());
   values.put(Schema.COL_ACTIVE, Schema.VAL_TRACK_ACTIVE);
   Uri trackUri =
getContentResolver().insert(ContentProviderPercorso.CONTENT_URI_T
RACK, values);
   long trackId = ContentUris.parseId(trackUri);
   // imposta il percorso attivo
   setActiveTrack(trackId);

   return trackId;
}
```

Si è implementato inoltre il tracciamento del percorso sulla mappa; in particolare, a tale scopo:

1. si è creato un Content Observer per essere avvertiti riguardo a qualunque nuovo punto del percorso e per ridisegnare di conseguenza lo schermo;

2. si è implementato il metodo VistaPercorso() per impostare il disegno del percorso e per recuperare risorse utili al disegno;

3. si è utilizzato il metodo onDraw() il quale consente di disegnare una view personalizzata ricevendo come parametro un oggetto Canvas che la view può utilizzare per disegnare se stessa.

```
/**
 * id del percorso corrente
 */
private long currentTrackId;
/**
* ContentObserver per essere avvertiti riguardo a qualunque nuovo
* punto del percorso e per ridisegnare lo schermo
*/
private class TrackPointContentObserver extends ContentObserver {
   public TrackPointContentObserver(Handler handler) {
      super(handler);
   }
}

/**
 * Istanza di TrackpointContentObserver
 */
private TrackPointContentObserver trackpointContentObserver;
public VistaPercorso(Context context) {
   super(context);
}
public VistaPercorso(Context context, long trackId) {
   super(context);
   currentTrackId = trackId;

   // Imposta il disegno del percorso
   trackPaint.setColor(getCurrentTextColor());
   trackPaint.setStyle(Paint.Style.FILL_AND_STROKE);

   // Recupera alcune risorse che saranno usate nel disegno
   meterLabel =
getResources().getString(R.string.various_unit_meters);
   northLabel =
getResources().getString(R.string.displaytrack_north);

   trackpointContentObserver = new TrackPointContentObserver(new
Handler());
   context.getContentResolver().registerContentObserver(
         ContentProviderPercorso.trackPointsUri(currentTrackId),
         true, trackpointContentObserver);
}


@Override
protected void onDraw(Canvas canvas) {
   super.onDraw(canvas);
   // Se si hanno dati da rappresentare
   if (pixels != null && pixels.length > 0) {
      int length = pixels.length;
      for (int i = 1; i < length; i++) {
```

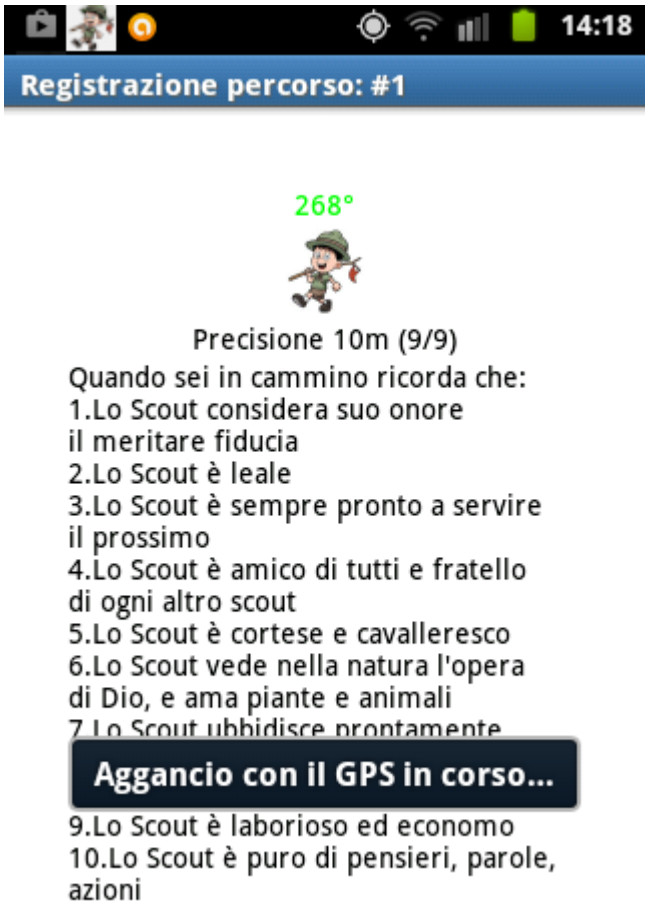
```
            // Disegna una linea tra ciascun punto
            canvas.drawLine(
                  PADDING + pixels[i - 1][ProiezioneMercator.X],
                  PADDING    + pixels[i - 1][ProiezioneMercator.Y],
                  PADDING + pixels[i][ProiezioneMercator.X],
                  PADDING    + pixels[i][ProiezioneMercator.Y],
trackPaint);
      }
      // Disegna il marker della posizione corrente
      canvas.drawBitmap(marker, pixels[length - 1]
[ProiezioneMercator.X],
            pixels[length - 1][ProiezioneMercator.Y],
this.getPaint());
   }
}
```

Per concludere si riportano i permessi aggiunti nel AndroidManifest.xml necessari alla piattaforma di registrazione e gestione dei percorsi realizzata:

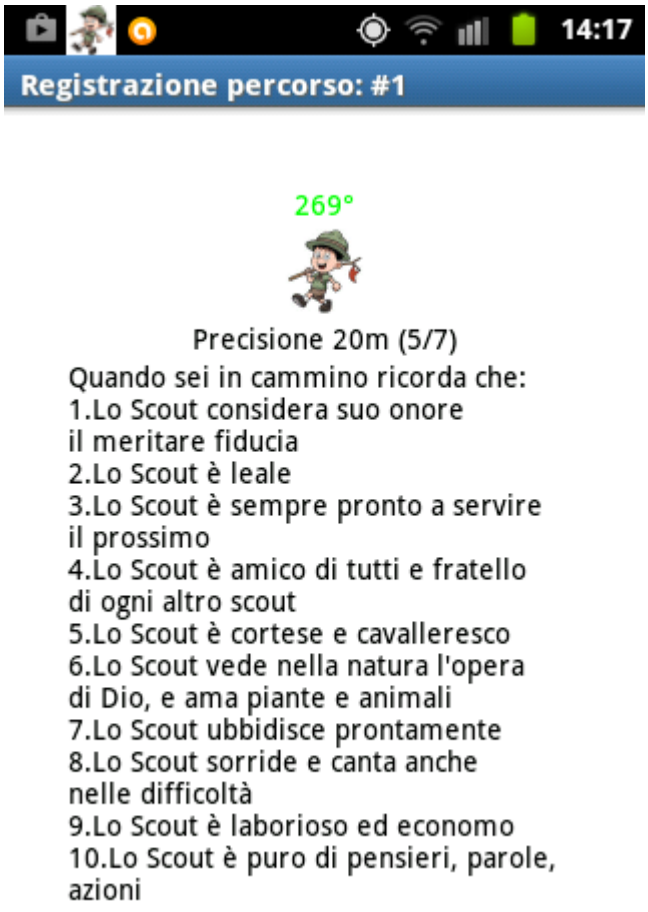
```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission
android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Screenshot Itinerari fig.1

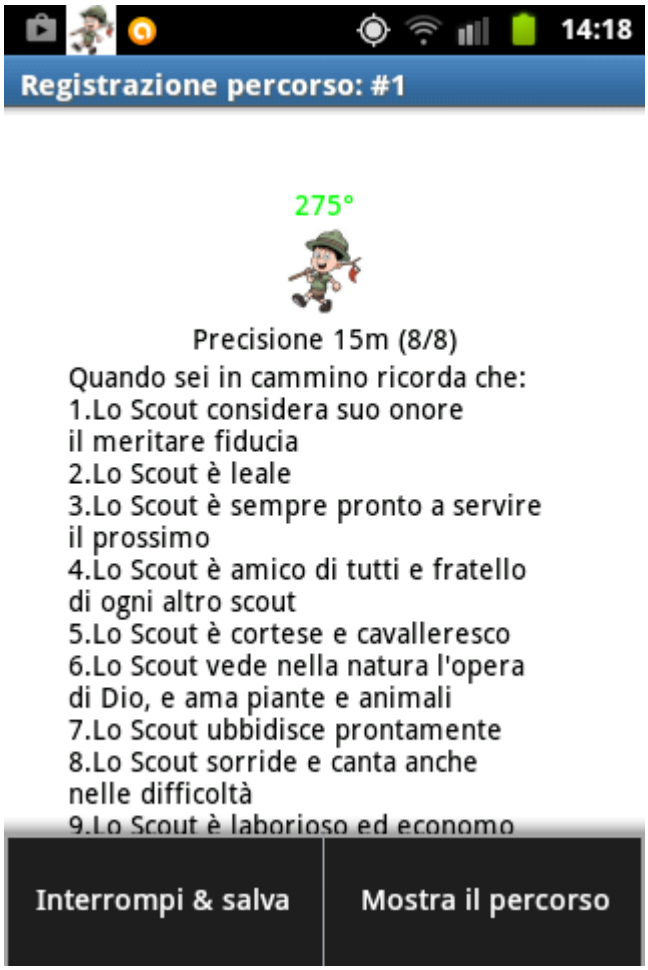
Screenshot Itinerari fig.2



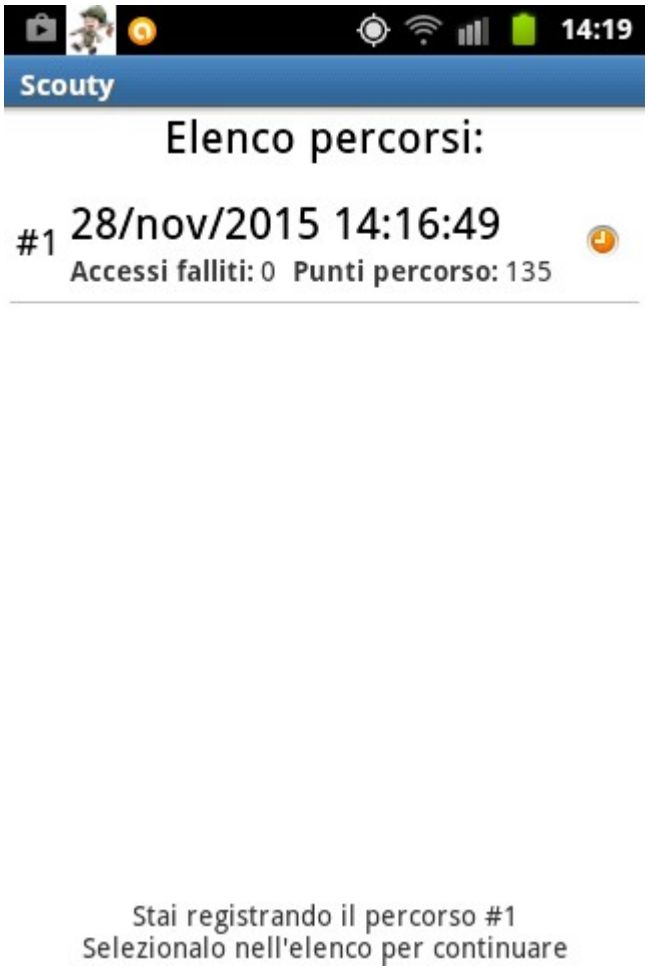
Screenshot Itinerari fig.3



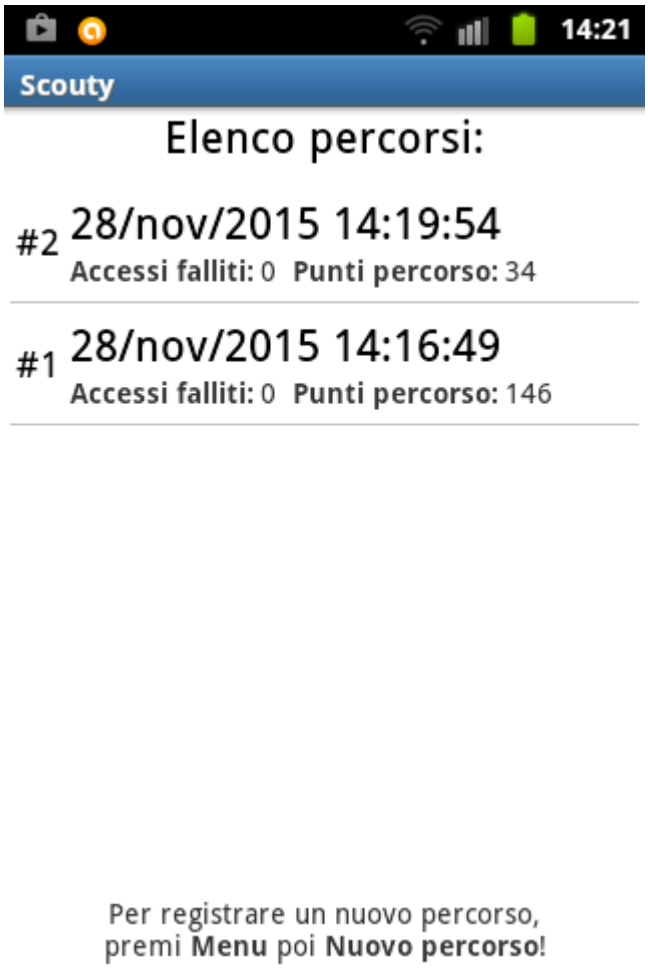
Screenshot Itinerari fig.4



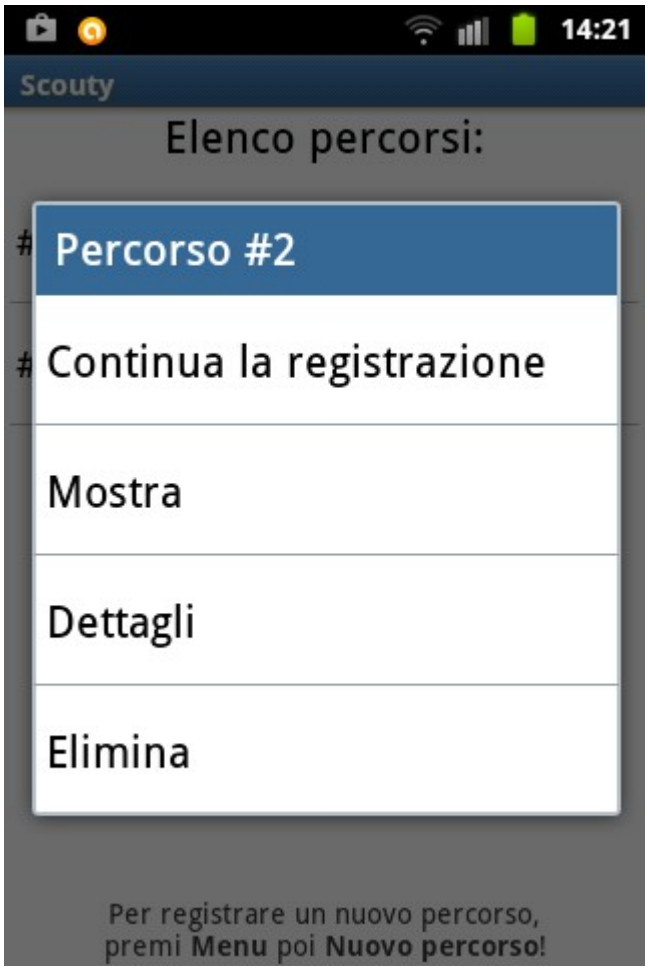
Screenshot Itinerari fig.5



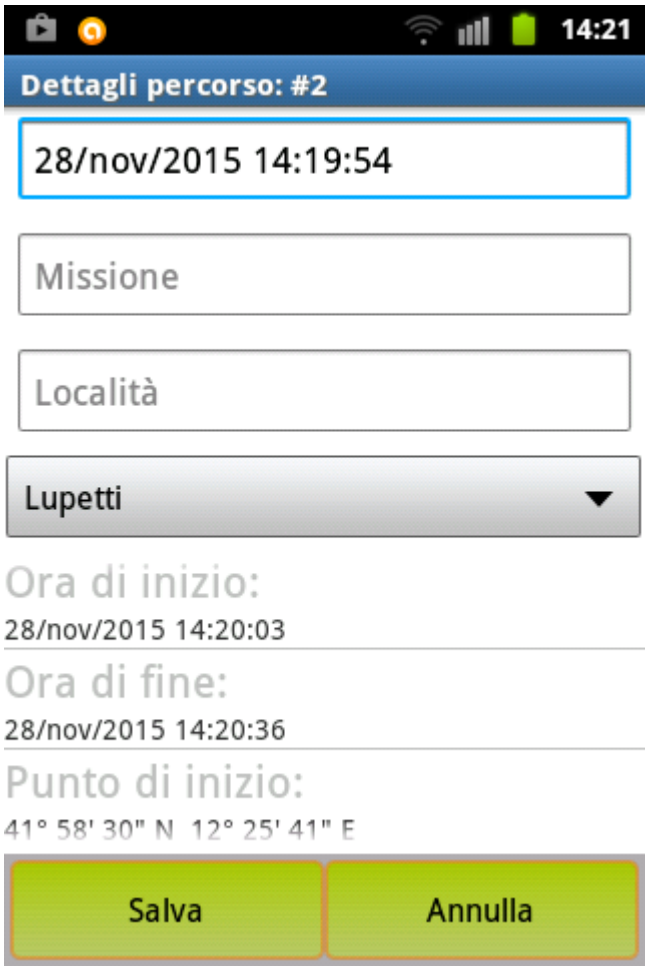
Screenshot Itinerari fig.6



Screenshot Itinerari fig.7



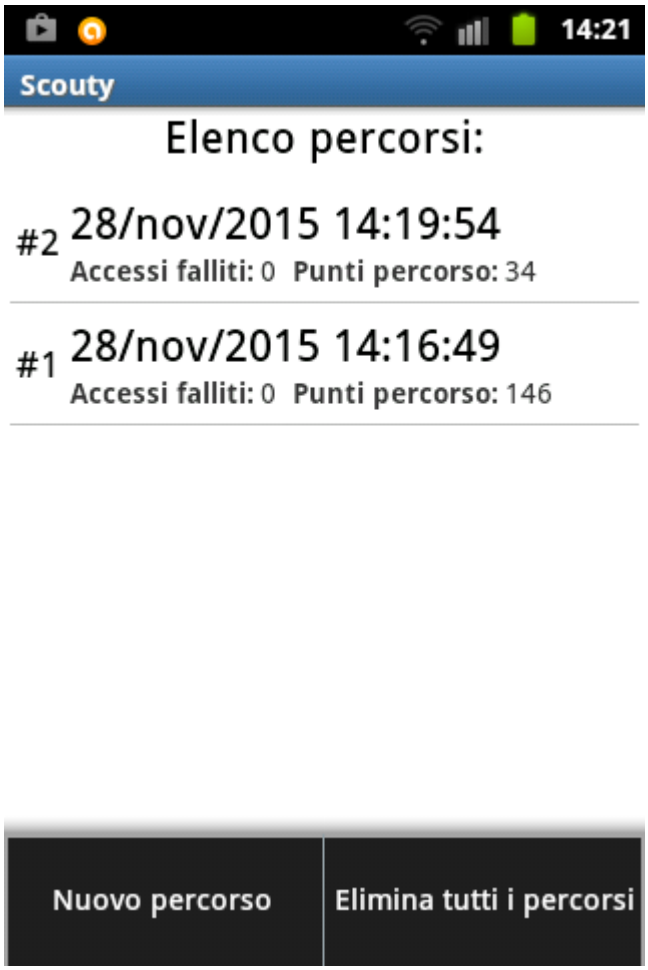
Screenshot Itinerari fig.8



Screenshot Itinerari fig.9

Screenshot Itinerari fig.10

Screenshot Itinerari fig.11



Screenshot Itinerari fig.12

Conclusioni

Grazie agli strumenti messi a disposizione dalla piattaforma Android, l'applicazione realizzata implementa numerose funzionalità tra cui alcune di livello avanzato quali l'autenticazione, la visualizzazione di mappe, la geolocalizzazione. Essa è predisposta all'integrazione di ulteriori funzionalità quali, ad esempio, VoIP e scambio dati.