G53MDP Coursework
Android Running App

Report

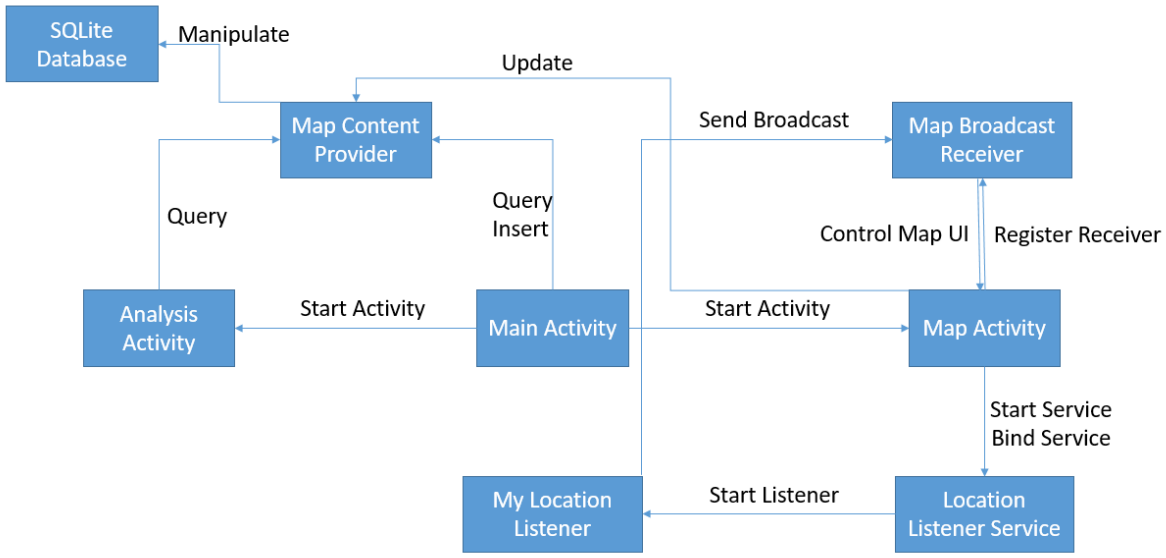
Wang Jinhao 4302178

Basic Functionalities:

The user is able to:

1. Start a track on his/her movement and stop the track at anywhere and anytime;
2. View a list of historical running records, with date, start time, finish time, running (walking) distance and speed displayed on it;
3. Sort the records in ascending or descending order, using selected attribute in the spinner;
4. View detailed statistical information categorized in daily and monthly;
5. View the movement route during tracking.

Structure Overview:



The whole application is composed of 3 activities, 1 service, 1 content provider and 1 broadcast receiver.

Explanation of the Diagram:

MainActivity:

- Queries data in database via MapContentProvider and displays them on the ListView;

```
public void queryAll() {
    Cursor cursor;
    cursor = getContentResolver().query(MapProviderContract.ALL_URI, projection,  selection: null,  selectionArgs: null,  sortOrder: null);

    displayRecord(cursor);
}
```

  In queryAll() method, calls method query(…) provided by MapContentProvider, which returns a Cursor object containing required data.

displayRecord(Cursor) is a helper method which displays corresponding data on the ListView.

- Inserts new data records into database via MapContentProvider when the user presses the START button to start a track;

```
public void onStartClick(View v) {

    String date = new SimpleDateFormat( pattern: "yyyy-MM-dd").format(new Date());
    String startTime = new SimpleDateFormat( pattern: "HH:mm:ss").format(new Date());

    ContentValues newValues = new ContentValues();
    newValues.put(MapProviderContract.DATE, date);
    newValues.put(MapProviderContract.START, startTime);
    Uri result = getContentResolver().insert(MapProviderContract.ALL_URI, newValues);
    int id = Integer.parseInt(result.getLastPathSegment());
```

When onStartClick(View) method is called, it means that user tends to start a new track record. Therefore, calls method insert(…) provided by MapContentProvider to add a new row of data into the database table.

- Starts MapActivity when user presses the START button to start a track;

```
    Bundle bundle = new Bundle();
    bundle.putInt("selectedID", id);
    bundle.putString("startTime", startTime);

    Intent intent = new Intent( packageContext: MainActivity.this, MapActivity.class );
    intent.putExtras(bundle);
    startActivity(intent);
}
```

When user starts a new track, the application should display MapActivity for user to view movement route. Therefore, calls method startActivity(Intent) to start MapActivity.

Since MapActivity needs to updates database as well, the new record ID should be passed to MapActivity. The startTime value is passed for MapActivity to do corresponding calculation.

- Starts Analysis Activity when user presses the View Data Analysis button.

```
public void onViewClick(View v) {
    Intent intent = new Intent( packageContext: MainActivity.this, AnalysisActivity.class);
    startActivity(intent);
}
```

Calls startActivity(Intent) to start AnalysisActivity.

MapActivity:

- Explicitly starts LocationListenerService and binds LocationListenerService to MapActivity when the map fragment is ready;

```
this.startService(new Intent( packageContext: this, LocationListenerService.class));
this.bindService(new Intent( packageContext: this, LocationListenerService.class), serviceConnection, Context.BIND_AUTO_CREATE);
```

startService(…) allows the service to run indefinitely in the background. Even when the service is unbound from the activity, it won't be destroyed by system unless stopservice() is called.

bindService(…) allows the activity bound to it can control the service behaviour.

- Creates a new MapBroadcastReceiver object and registers the receiver to MapActivity;

```
receiver = new MapBroadcastReceiver(myMap);
IntentFilter filter = new IntentFilter( action: "Tracking");
LocalBroadcastManager.getInstance(this).registerReceiver(receiver, filter);
```

IntentFilter specifies the action of Broadcast receiver. When other components are broadcasting, the receiver will only listen to and receives broadcast which sets its action.
Uses LocalBroadcastManager to prevent data leakage and improve efficiency.

```
this.unbindService(serviceConnection);
this.stopService(new Intent( packageContext: this, LocationListenerService.class));
LocalBroadcastManager.getInstance(this).unregisterReceiver(receiver);
```

Since LocalBroadcast is dynamically registered in code fragments rather than statically registered in the manifest, the receiver should be unregistered when it is no longer needed. That is, when the user finishes the current track and presses the STOP button.
Similarly, service should also be stopped and unbound.

- Updates existed records in database via MapContentProvider when the user presses the STOP button to end a track.

```
ContentValues values = new ContentValues();
values.put("finish", finishTime);
values.put("distance", totalDistance);
values.put("velocity", velocity);
values.put("duration", diffInSec);
getContentResolver().update(MapProviderContract.ALL_URI, values,  where: MapProviderContract._ID + "=" + selectedID, selectionArgs: null);
```

When user finishes the track, updates the new record with corresponding data.

- When MapActivity is visible on top of the stack, the application should prevent user from returning to MainActivity by pressing the system back button. Only when the user finishes tracking, he/she can go back to MainActivity by pressing STOP button directly.

```
@Override
public void onBackPressed() {
    moveTaskToBack( nonRoot: true);
    //super.onBackPressed();
}
```

Therefore, I override the onBackPressed() method to make the system back button acts like home button when MapActivity is visible on top of the stack. So there will be no influence on tracking service functioning normally.

AnalysisActivty:

- Queries data in database via MapContentProvider and displays them in corresponding TextViews.

```
public void dailyAnalysis() {
    String date = new SimpleDateFormat( pattern: "yyyy-MM-dd").format(new Date());
    Cursor cursor = getContentResolver().query(MapProviderContract.ALL_URI, projection,  selection: MapProviderContract.DATE + "= '" + date +"'",  selectionArgs: null,  sortOrder: null);

    cursor.moveToFirst();
    while (cursor.isAfterLast()==false) {
        dailyDistance = dailyDistance + cursor.getInt(cursor.getColumnIndex( s: "distance"));
        if (cursor.getFloat(cursor.getColumnIndex( s: "velocity")) > dailyBestSpeed) {
            dailyBestSpeed = cursor.getFloat(cursor.getColumnIndex( s: "velocity"));
        }
        cursor.moveToNext();
    }

    temp = (TextView) findViewById(R.id.textView11);
    temp.setText(String.valueOf(dailyDistance));

    temp = (TextView) findViewById(R.id.textView13);
    temp.setText(String.valueOf(dailyBestSpeed));
}
```

Same as how MainActivity shows data on ListView.
Calls moveToFirst() method because the default position of the cursor is on row 0, which indicates column attributes.

LocationListenerService:

- Creates a new MyLocationListener object;

- Requests system service LocationManager to request regular update on location via MyLocationListener object.

```
locationListener = new MyLocationListener( myContext: this);
locationManager =
        (LocationManager)getSystemService(Context.LOCATION_SERVICE);

try {
    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
            minTime: 5, // minimum time interval between updates
            minDistance: 5, // minimum distance between updates, in metres
            locationListener);
} catch(SecurityException e) {
    Log.d( tag: "g53mdp", e.toString());
}
```

LocationManager is a system service that enables locationListener to run indefinitely in the background. Hence, unless the user presses STOP button to finish a track, the application can be minimized to background while it is still tracking user's movement.

```
@Override
public void onDestroy() {
    locationManager.removeUpdates(locationListener);
    super.onDestroy();
}
```

Therefore, when the user finishes tracking, LocationListenerService is stopped and unbound, it should requests locationManager to close locationListener. Otherwise, the locationListener will keep listening to updated location as long as the locationManager exists.

MapContentProvider:

- Provides methods for other components to call to manipulate database. Overrides 4 basic operations, insert, query, update and delete (not used).

MapBroadcastRecevier:

- Gains control on MapActivity's UI component – map fragment when created, and draws movement route while tracking;

```
public MapBroadcastReceiver(GoogleMap googleMap) { myMap = googleMap; }
```

When the receiver is created in MapActivity, map fragment in MapActivity is passed to the receiver via the constructor.

- Recevies updated location information from MyLocationListener object.

```
@Override
public void onReceive(Context context, Intent intent) {
    points = intent.getParcelableArrayListExtra( name: "routePoint");
    totalDistance = totalDistance + intent.getFloatExtra( name: "perDistance",  defaultValue: 0);
    redrawLine();
}
```

Calls onReceive(…) method each time receiver receives a broadcast. redrawLine() is a helper method which draws movement route on the map fragment.

MyLocationListener:

- Sends broadcast containing current location information to broadcast receiver.

```
@Override
public void onLocationChanged(Location location) {
    if (points.isEmpty() == false) {
        distance = location.distanceTo(previousLocation);
    }

    previousLocation = location;
    latitude = location.getLatitude();
    longitude = location.getLongitude();
    LatLng latLng = new LatLng(latitude, longitude);
    points.add(latLng);

    Bundle bundle = new Bundle();
    bundle.putParcelableArrayList("routePoint", points);
    bundle.putFloat("perDistance", distance);

    Intent intent = new Intent( action: "Tracking");
    intent.putExtras(bundle);
    LocalBroadcastManager.getInstance(myContext).sendBroadcast(intent);

    Log.d( tag: "CW4",  msg: location.getLatitude() + " " + location.getLongitude());
}
```

Calls onLocationChanged(Location) method every 5 minutes or location changes at least 5 meters. This is controlled by system service LocationManager which is requested in LocationListenerService.
Every time there is an update in location, it will send a broadcast to broadcast receiver with updated location data and data needed to be used for AnalysisActivity.

Explanation on Other Parts of Application:

Spinner_options.xml:

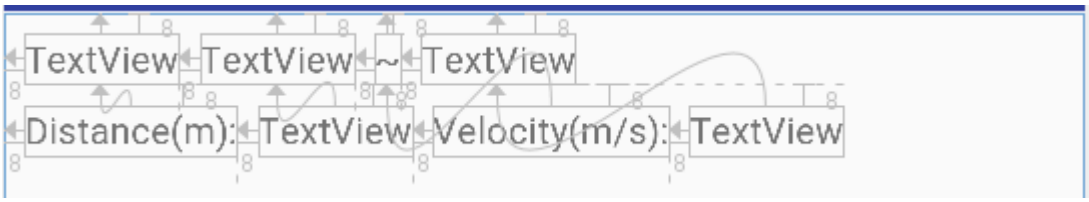
```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="spinner_array">
        <item>Duration</item>
        <item>Distance</item>
        <item>Speed</item>
    </string-array>
</resources>
```

Since available choices for the user to sort data is pre-determined, I used string resource file Spinner_options.xml to define a string array containing these choices.

```
spinner = (Spinner) findViewById(R.id.spinner);
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource( context: this,
        R.array.spinner_array, android.R.layout.simple_spinner_item);
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinner.setAdapter(adapter);
```

The spinner in MainActivity uses string-array name to specify choices values.

Record_lv.xml:



This layout xml file is used to control how data is displayed in MainActivity ListView.

```
dataAdapter = new SimpleCursorAdapter(
        context: this,
        R.layout.record_lv,
        c,
        colsToDisplay,
        colResIds,
        flags: 0);

lv = (ListView) findViewById(R.id.Record);
lv.setAdapter(dataAdapter);
```

In MainActivity, ListView's dataAdapter specifies the layout, meaning that each element in the ListView should display data in this format.

Google Map:

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="AIzaSyBL7u4EON9X0SofpiDO0azqbb1dF2b9xhw" />
```

In manifest, add the Google Map API_KEY required from Google Cloud Platform in order to successfully request map service from Google.

```
implementation 'com.google.android.gms:play-services-maps:16.0.0'
```

In the application build.gradle, add Google Map Service to dependencies.

```
repositories {
    google()
    jcenter()
}
```

In the project build.gradle, add "google()" to repositories.

Permission:

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

These permissions should be added to the manifest.

Since there is a need to request for corresponding permissions for map fragment (Google Map Service) and LocationListener to work properly.

```
if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_COARSE_LOCA
    ActivityCompat.requestPermissions( activity: this, new String[]{Manifest.permission.ACCESS_FINE_LOCATION},  requestCode: 1);
}
myMap.setMyLocationEnabled(true);
```

For example, when enabling 'locating myself' button on the map fragment, it requires to check whether the application has gained proper permissions to have the right to do so. Otherwise it will result in an error.