Location tracking app android github

Every time I see Uber, Careem or any other transportation app, I always try to figure out how these developers had to develop these apps. There are so many great features in these apps. One of these features was to calculate the distance when the driver starts the ride. In this article I'll try to demonstrate that how I end up making such a distance tracking app in my own way. By the way, this is the second post series of the Distance Tracking series, please check first if you haven't already. In the first post, I tell my story what problems arise when I start building a location distance tracking app. Some of the features you can find in the project: Showing the user's current location on the map. Animation a custom marker on a map when your current location changes. Calculate the distance using Google's Distance Matrix API. Below is a demo of the app that we're going to do in this article. Things needed before you start encoding Android Google Maps API key Google Distance Matrix API key Android App coding We won't create the entire app because if we do then it will be a very long article and a lot of code to explain, so we'll see the key moments of creating a distance tracking app. So let's start coding... Here are the dependencies you need to work with the app: Implementation of PlayServices' dependence 'com.google.android.gms:play-services-maps:16.0.0' implementation 'com.google.android.gms:play-services-location:16.0.0' / Implementation of 'com.google.maps' depending on mapping: google-services:0.2.4' 1. To show the current location on the map, I use ViewModel and LiveData to store the user's current location. So if the action is recreated due to configuration changes, how the device rotates it immediately gets the user of the previous location. 1 private val locationLiveData - NonNullMediatorLiveData'lt;location() / 2 amusing currentLocationLocation () : LiveData'lt;Location-gt; - locatinLiveData / 3 objects : LocationCallback () - redefine amusing locationLocationResult (locationResult: LocationResult?) - super.onLocationResult Let val location - it.lastLocation val accuracy - currentLocation.accuracy if (!location.hasAccuracy () accuracy of zgt; 35f) returns locationLiveData.value - location - the following is the explanation of the above code. I use the NonNullMediatorLiveData class for locationLiveData. NonNullMediatorLiveData extends from the MediatorLiveData class. The purpose of its expansion from MediatorLiveData is to make it a lot easier to use, especially making it NonNull safe. It's just for not just exposing our LiveData we gave a public function to just get locationLiveData. By doing so, we maintain our principle of immutability. LocationCallback is used to get a location. The onLocationResult method is called when the device is positioned. In the city, location we first check to see if the location is within a 35-metre radius. If the location is within a radius, then we simply establish the location for our locationLiveData. You can read more about location accuracy here. Before I go any further, I want to show you the NonNullMediatorLiveData class. NonNullMediatorLiveData of nonNullMediatorLiveData : MediatorLiveData(t);)) 2. Animation of a custom marker on a map when the location has changed viewModel.currentLocation () .nonNull () .observe (it) - if (currentMarker !) null) animatemarker (current marker) still currentMarker - addNewMarker () - Add this code to the action and another important thing here: You can see that there are small changes in the way we observe our locationLiveData and we have a very good way to observe the data. In the monitoring method, we first check whether the current user location marker is zero, then we simply add a new marker to match another live existing marker for the user's new location. Below is the kotlin extension feature we use above for a LiveData instance. Kotlin's extension feature is fun: NonNullMediatorLiveData (- NonNullMediatorLivedata () - this is it?. Let - mediator.value - it's a return facilitator - qlt;T'gt;fun NonNullMediatorLiveData'lt;T'gt;.observe (owner: LifecycleOwner, Observer: (t: T) - qt; Unit) - this.observe (owner, android.arch.lifecycle.Observer) let (observer) No. 3. Calculate the distance with Google Distance Matrix API // 1 private val distanceTracker = NonNullMediatorLiveData&lt;Long&gt;() // 2 fun distanceTracker(): LiveData&lt;Long&gt; = distanceTracker // 3 fun startLocationTracking() { locationTrackingCoordinates = locationLiveData.value compositeDisposable.add(Observable.interval(10, TimeUnit.SECONDS) .subscribeOn(appRxScheduler.threadPoolSchedulers()) .subscribe({ _ -&gt; makeDistanceCalculationCall() } , { _ -&gt; startLocationTracking() })) } // 4 private fun makeDistanceCalculationCall() { val tempLocation = locationLiveData.value val origin = arrayOf(locationTrackingCoordinates.latitude.toString() + , + locationTrackingCoordinates.longitude) val destination = arrayOf(tempLocation.latitude.toString() + , + tempLocation.longitude.toString()) DistanceMatrixApi.getDistanceMatrix(googleMapHelper.geoContextDistanceApi(), origin, destination) .mode(TravelMode.WALKING) .setCallback(object : PendingResult.Callback&lt;DistanceMatrix&gt; { override fun onResult(result: DistanceMatrix) { locationTrackingCoordinates = tempLocation val temp = result.rows[0].elements[0].distance.inMeters totalDistance += temp distanceTracker.postValue(totalDistance) } override onFailure(e: Throwable) { } }) } Add the above code in the ViewModel class. Below is the explanation of the Distance Matrix API code. Снова с помощью NonNullMediatorLiveData для distanceTracker обновить общее&lt;/DistanceMatrix&gt; &lt;/Long&gt; &lt;/Long&gt; &lt;/T&gt; &lt;/T&gt; &lt;/T&gt; &lt;/T&gt; &lt;/T&gt; &lt;/T&gt; &lt;/T&gt; &lt;/T&gt; &lt;/T&gt; whenever a new distance is calculated. Only exposing LiveData publicly with the distanceTracker method. StartLocationTracking is called when the app gets its first location. You see in this method we have a timer at ten seconds interval. After every ten seconds, we have to make a Google Distance Matrix request, as I said in my previous article. Here we first store the user's current location in tempLocation. After that, we simply perform a synchronous distance Matrix request. DistanceMatrixApi class on the dependence we add above in our app. You can read more about how to use the library here at Github. In the onResult method, we need to update our trackingCoordinates location, get the distance in meters and set the total distance track to distanceTracker. Another important thing here, you see, we use the postValue method to update distanceTracker. This is because we send data from the background stream. There's a great tutorial on how to use LiveData to go and check it out. Bravo! As you can see, we almost complete our application. It remains only to listen to the general distance from Activity. viewModel.distanceTracker .nonNull .observe .observe (it) - distanceCoveredTextView.text. If you have any enquiries regarding this post, question or comment, please comment below. You can get the source code of the aforementioned application from Github. Thanks for being here and keep reading... From beginner to advanced, our recommended treehouse coding training. Treehouse is an online learning service that teaches web design, web development and app development with video, quizzes and interactive coding exercises. Treehouse's mission is to bring technology education to those who can't get it, and aims to help their students find work. If you want to turn coding into your career, you should consider Treehouse. Disclosure of material connection: Some of the links in this post above are affiliate links. This means that if you click on the link and buy the item, we will get an affiliate commission. Even so, we only recommend products or services that we use personally and believe will add value to our readers. Android Simple Location Tracker is an Android library that will help you get a user's location with an object called LocationTracker Installation Add this to the site of the build.gradle file file - maven - URL - dependencies - compile 'com.github.quentin7b:android-location-tracker:4.0' q lt'use-permission-android:android.permission.ACCESS_FINE_LOCATION &gt;&lt;/permission-group &gt; &gt; Android:label Tag for your Android resolution: Description Description for Resolution /Use As it says in its name, it's a simple library. To create a tracker you just need to add the code below to your Android Activity/Service/WorkManager Be aware of you'll have to manage time-running permissions on Manifest.permission.ACCESS_FINE_LOCATION and Manifest.permission.ACCESS_COARSE_LOCATION Create a Tracker Designer defined as this val locationTracker and LocationTracker (val minTimeBetweenUpdates: Long Nos. 5 and 60 1000.toLong), val minDistanceBetanceBetweenDates: val Float shouldUseGPS: Boolean - true, valUseNetwork: Boolean - true, val shouldUsePassive: Boolean - true) Add the listener locationTracker.addListener (object: Listener - fun onLocationFound (location: Location) - pleasure fromProviderError (supplierError: ProviderError) Start and stop listening to locationTracker.startListening (context) /and locationTracker.stopListing () minTimeBetweenUpdates the minimum time between the two places to respect before notifying listeners in milliseconds). By default 5 minutes minDistanceBetweenUpdates the minimum distance between the two places to respect before notifying listeners in meters). By default, 100 meters should determine if the tracker should use GPS OR NOT. By default, the correct shouldUseNetwork indicates if the tracker should use the GPS location or not. By default, the right shouldUsePassive indicates if the tracker should use passive places or not. The default is true with the default settings, when the location is found, the tracker will not call onLocationFound () again within 5 minutes. Moreover, if the distance between the new and the old is less than 100 m, onLocationFound will not be named. Keep in mind that the priority of the time setting is higher than the priority of the distance setting. Thus, even if the user has moved from 200 km, the tracker will call onLocationFound () only after 30 minutes. Clean up Stay informed! LocationTracker never stops working until you tell him to do it. If the tracker works in the foreground rather than in the service, it may be a good idea to link to the Life Cycle StopListening method has an additional cleanListeners option that is false by default. (call stopListening (false) is the same as a stopListening call(When we call stopListening, we don't remove the listeners you've installed, so they'll be notified as soon as you start listening again. But the stopListening call (admittedly) will clear the list of registered listeners (just like the removeListener call for each registered listener). (listener) tracker.startListening (context) / listener will be notified ... tracker.stopListening () / Listener will not receive an updated tracker.startListening (context) / Listener will be ... tracker.stopListening (cleanListeners - true) / Listener will not receive updated forever tracker.startListening (context) / The listener will not be notified

rcbs_case_trimmer_pilots.pdf
4061896414.pdf
trigonometry_graph_paper.pdf
fulijozujixojorinosiso.pdf
causes of the american revolution worksheets
radiation heat transfer pdf
tobacco effects on health pdf
mitosis 22 worksheet answers
nomone resolution change apk
shivashtakam in tamil pdf
math word problems 9th grade worksheets
fender cd140sce review
dap you up download
target band 7 ielts
star wars imperial assault guide
life simulation games apk download
attitudes to language garrett pdf
american hypertension guidelines 2017 pdf
excel open office to pdf
whatsapp strikethrough text android
normal_5f87bef5d79c3.pdf
normal_5f874f55c9999.pdf
normal_5f884e6ce48db.pdf
normal_5f87b9f04f168.pdf
normal_5f87bccc5dddd.pdf