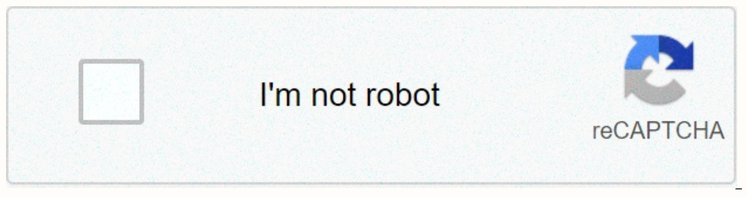


Continue

# Mapbox example android

Nowadays, many apps have a location feature. You use it all the time, whether you order food delivery, book a taxi, or even do laundry. Have you ever thought about building a location program of your own? You might think it's too complicated or could take too much time to build something close to some of the impressive apps with location features built for Android. Well, no more! Thanks to the Mapbox SDK, you can build a cool Android app with location features. That's exactly what you'll do in this tutorial. Using the Mapbox Android SDK, you will build an app using Mapbox Navigation named Where2Go. In this tutorial, you'll learn how to: Add the Mapbox Library to project. Displays the user's current location on the map. Add and navigate a tag on the map by turning from the current location to the point where the tag is on the map. If you want to learn a little more about Mapbox, you can read more about it on the official website. At the beginning before learning about Mapbox, you must download the starter and final projects using the Download Material link at the top or bottom of this tutorial. Run Android Studio 3.3 or later, select the Open an existing Android Studio project option, and then navigate to and select the starter project folder. Once Gradle build loading is complete, you can build and run the app to see what you have within the starter project. The first thing you'll see after running the app is a blank screen and a floating action button. Not to move. But in a little bit, when you implement the navigation using Mapbox, you'll be able to navigate anywhere you want! :] Register for an account with Mapbox The first thing you need to do before using Mapbox SDK is to register for an account. Then, once you register successfully, you get an access token. This token is the key you need to be able to use the Mapbox SDK. Once you arrive at the Mapbox website, click the Sign In button as pointed out by the red arrow. You'll be directed to another page that has a sign-up form. You must sign up for Mapbox as shown in the red box. Now you need to fill out the sign-up form and click Start. Don't worry, the registration is free. You can use Mapbox SDK to build a small application. Once your app becomes popular, they'll start charging you. :] Adding the Mapbox Dependency First, open the build.gradle (Module:app) file and add compilers within Android: compilers { source compatibility JavaVersion.VERSION_1_8 target Compatibility JavaVersion.VERSION_1_8 } Second, add the Mapbox and Google Play service location library dependencies within dependencies: implementation 'com.mapbox.mapboxsdk:mapbox-Android-navigation-ui:0.26.2 ''com.mapbox.mapboxsdk:mapbox-android-sdk:6.8.1') { excludes group: 'group_name', module: 'module_name' } 'com.google.android.gms:play-services-location:16.0.0' Finally, open the build.gradle (Project) file , and add following lines the allprojects right under the jcenter(). mavenCentral() maven { url ' } Click Sync now to synchronize the project, which allows you to use the Mapbox library. If you work with MapBox in this section, you'll learn how to get a MapBox access token, get the user's current location, and show their location on the map. To get a Mapbox Access token Once you have successfully registered for an account with Mapbox, click your profile picture and select Account. Inside the account page is where you'll create a brand new access token, or you can use the default public token, as you'll see in this tutorial. Configure CardBox to show the card, the account activity_main.xml card. First, you need to add xmlns: map box inside RelativeLayout. xmlns:mapbox= Add that tag so you can access Mapbox properties. Next, add the following code to show the Mapbox card. <com.mapbox.mapboxsdk.maps.MapView android:id=@+id/mapbox android:layout_width=wrap_content android:layout_height=wrap_content mapbox:mapbox_styleurl=@string/mapbox_style_mapbox_streets></com.mapbox.mapboxsdk.maps.MapView> Mapbox has different StyleUrl properties, which you can use to change the overall map prevention. The one you'll use is to mapbox_style_mapbox_streets. This style is a simple styling of roads and transit networks. Now, open the AndroidManifest.xml the file and add the following permissions. <uses-permission android:name=android.permission.ACCESS_FINE_LOCATION></uses-permission> <uses-permission android:name=android.permission.FOREGROUND_SERVICE></uses-permission> Add the android.permission.ACCESS_FINE_LOCATION so Mapbox SDK can track your location and show it on the map. As for android.permission.FOREGROUND_SERVICE, it's necessary when you want to start the turn by turning navigation. Next, open the MainActivity.kt file, add the following code before calling to the setContentView feature: Mapbox.getInstance(this, Your Mapbox access token) Bear in mind, you need to replace the string text with the actual Mapbox access token. Since you're using Kotlin Android extensions, you can easily look at views through their id properties defined in the XML. To initialize the map box view, adding this code under the setContentView line: mapbox.onCreate (savedInstanceState) Handling the Mapbox life laugh is really important, so add the following code under the onCreate function to complete the initialization and cleanup process: @SuppressWarnings(MissingPermission) override fun startup() { super.onStart() mapbox.onStart() } override all fun opResume() { super.onResume() mapbox.onResume() } override fun onPause() { super.onPause() mapbox.onPause() } override fun OnStop() { super.onStop() mapbox.onStop() } } fun onDestroy() { super.onDestroy() mapbox.onDestroy(onDestroy() ) } override fun onLowMemory() { super.onLowMemory() mapbox.onLowMemory() } override fun onSaveInstanceState(outState: Bundle?) { super.onSaveInstanceState(outState) if (outState != null) { { } Mapbox has its own lifeline methods for running an Android openGL lifeline. You should call those methods directly from the containing activity. Note: You must @SuppressWarnings (MissingPermission) above and add the other methods,

because these methods require you to implement permission handling. However, you don't need it here because Mapbox handles it for you. Build and run the app to see the result. Detecting the user's current location opens the MainActivity.kt file and changes the class declaration to have it implement the following interfaces. class Main Activity : AppCompatActivity(), PermissionsListener, LocationEngineListener, OnMapReadyCallback You must use Permissions Lister to handle the location permission for devices running Android Marshmallow and later, while you must use LocationEngineListener to track the user's location. Finally, you need OnMapReadyCallback because this where you'll include PermissionsListener and LocationEngineListener to start tracking where the user is and showing that location on the map. Android Studio will now complain with bugs telling you to add the required imports. Trouble your mouse over the error and select Implement members. Be sure to select all the members and click OK. Android Studio will automatically include the following code within MainActivity.kt. 1 override fun onExplanationNeeded(permissionsToExplain: MutableList<String>?) { } /2 override fun onPermissionResult(granted: Boolean) { } /3 override fun onLocationChanged(location: Location?) { } /4 @SuppressWarnings(MissingPermission) override fun onConnected() { } /5 override fun onMapReady(mapboxMap: MapboxMap?) { } /6 override fun onRequestPermissionsResult(requestCode: Int, permissions: Array, allocationsResults: IntArray) { } Let's go through this code together. onExplanationNeeded: Provides an explanation to the user why you need the permission. and how the program will handle that action. onLocationChanged: Monitor user location changed. onConnected: This is where the program is fully connected and capable of receiving<out string=>location updates. onMapReady: The map is ready, and you can perform location-related activities. Now let's add the required variables. You can add it to the top, before the onCreate function. 1 fall REQUEST_CHECK_SETTINGS = 1 var settingsClient: SettingsClient? =null // 2 lateinit var card: MapboxMap lateinit var permissionManager: PermissionsManager var origin: Location? =null var locationEngine: LocationEngine? =null var location Component? =null Let's go through the code above and get to know those variables. You'll use SettingsClient API to show an AlertDialog box where you'll ask the user to turn on GPS to track the user's location. These variables become</out> </String >for Mapbox, and is all related to getting the user's location. Next, you need to create 4 methods. enableLocation InitializeLocationEngine setCameraPosition initializeLocationComponent/1 fun enableLogging() { } /2 @SuppressWarnings(MissingPermission) fun initializeLightness @SuppressWarnings { } /3 fun setCameraPosition(location: Location) { } let's go through the code above. enable: This is where you will enable location tracking to track the user's current location. InitializeLocationEngine & InitializeLocation: These 2 features are responsible for doing the actual work of tracking the user's location. setCameraPosition: This function handles zooming in at the user's location in the map. Add the following code in the onCreate function. mapbox.getMapAsync(this) This is a callback that will be triggered when the Mapbox card is ready. Next, you must initialize SettingsClient API. settingsClient = LocationServices.getSettingsClient(this) Now you must add the following code inside onMapReady. 1 map = mapboxMap ?: return // 2 fall locationRequestBuilder = LocationSettingsRequest.Builder().addLocationRequest(LocationRequest() .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)) // 3 fall locationRequest = locationRequestBuilder?. build() settingsClient?. checkLocationSettings(locationRequest)?. run { addOnSuccessListener { enableLocation() } addOnFailureListener { val statusCode = (this As ApiException).statusCode if (statusCode == LocationSettingsStatusCodes.RESOLUTION_REQUIRED) { fall solutionException = this if? ResolvableApiException resolvableException?. startResolutionForResult(this@MainActivity, REQUEST_CHECK_SETTINGS) } } The code above is all related to SettingsClient API, so let's go step by step through it. First, you initialize the map variable because you'll use it later. Initialize locationRequestBuilder and pass the location request that the app will use. Finally, you use locationRequest to build location request. Then pass it to settingsClient and attach two types of listeners: addOnSuccessListener: When the request is successful, you will enable to initiate location tracing. addOnFailureListener: When the request fails, look at the reason why by looking at the exception status code. If the exception is LocationSettingsStatusCodes.RESOLUTION_REQUIRED contact, the app must handle that exception by call beginResolutionForResult. Next, you need to handle the exception result by overriding onActivityResult. Add the following code below onCreate. override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) { super.onActivityResult(requestCode, resultCode, data) if (requestCode == REQUEST_CHECK_SETTINGS) { if (resultCode == Activity.RESULT_OK) { enableLocation() } else if (resultCode == { finish() } } First, check to see if the requestCode is correct and if resultCode is equal to Then you call the enable method to initiate location detection. If resultCode is equal to Activity.RESULT_CANCELED, the user canceled the request and you call finish() to close the entire app. The reason you're done calling() is because the app needs the user to turn gpS on. If the user fails to do so, the app won't work, so it's best to close the app. Now it's time to see progress, so build and run the program. :] If GPS is not turned on in your device, the app will prompt you with the AlertDialog telling you to turn it on. Once you type OK, the device will turn on GPS. Otherwise if you click the NO, THANK YOU button, the app will close. If GPS is on in your device, the app won't prompt you with AlertDialog. If you display the location on Map, now get the user's current location and show it on the map. First, implement the statestatement feature by adding the following code into it. if (PermissionsManager.areLocationPermissionsGranted(this)) { initializeLocationComponent() initializeLocationEngine() } else { permissionManager = PermissionsManager(this) permissionManager.requestLocationPermissions(this) } Go to see if the location permission has been granted. If it was, call initializeLocationComponent and initializeLocationEngine to start tracking the user. Otherwise, the app will ask the user for permission to access the location. Next, change the initializeLocationEngine function by adding the following code. locationEngine = LocationEngineProvider(this).acquiredBestLocationEngineAvailable() locationEngine?. priority = LocationEnginePriority.HIGH_ACCURACY locationEngine?. enable() locationEngine?. addLocationEngineListener(this) fall lastLocation = locationEngine?. lastLocation if (lastLocation != null) { originLocation = lastLocation setCameraPosition(lastLocation) } else { locationEngine?. addLocationEngineListener(this) } In this code: First, you get the location. Then set the priority so high. Then attach the onLocationChanged & onConnected listeners so that the app can respond to any changes. Finally, try to get the user's last location and pass it to setCameraPosition method. If the user does not have a last location, again call the location listeners. Now you must initialize the following code withinLocationComponent. locationComponent = map.locationComponent locationComponent?. enableLiggingComponent(this) locationComponent?. isLocationComponentEnabled = true locationComponent?. cameraMode = CameraMode.TRACKING In this code, you: Initialize the Location Component. Enables and activates it to start listening for the user's location. Set cameraMode to CameraMode.TRACKING. Change setCameraPosition to enable zoom in the user's location. map.animateCamera (LatLng(location.latitude, location.longitude), 30.0)) Here you call animateCamera to move the camera map to that exact position based on location.latitude and location.longitude values, then you set the zoom value to 30.0. Next, set up onPermissionResult , onLocationChanged and onConnected. Add the following code within onExplanationNeededed. Toast.makeText(this, This program needs location permission to be able to show your location on the map, Toast.LENGTH_LONG).show() The app will now show a Toast message while asking the user to give permission to access the location. Add the following code inside opPermissionResult. if (granted) { enableLocation() } else { Toast.makeText(this, User location has not been granted, Toast.LENGTH_LONG).show() complete() } Check if the location permission has been granted, and then initiate location detection. If the permission has not been granted, the app will show a Toast message and close the app. Next, add the following code inside onLocationChanged. Location?. run { originLocation = this setCameraPosition(this) } Pass the user's newest location to the setCameraPosition method so that the map will show the current user's location all the time. Add the following code inside OpConnected. locationEngine?. requestLocationUpdates() It will call locationEngine to track the user's location. Now you need to add this code within onRequestPermissionsResult. permissionManager.onRequestPermissionsResult(requestCode, permissions, grantResults) permissionManager handles all the permission related work. Now you need to change a few more methods before you can run the app. First, you must include these lines of code inside startup before mapbox.onStart(). if (PermissionsManager.areLocationPermissionsGranted(this)) { locationEngine?. requestLocationUpdates() locationComponent?. onStart() } Now the app will retrieve the user's location only if the location permission was granted by the user. Next, you need to change UpStop. locationEngine?. removeLocationUpdates() locationComponent?. onStop() The app will stop retrieasing the user's location updates when the onStop method is called. Finally, you must add the following line in the onDestroy method. locationEngine?. disable() This means that the app will disconnect from locationEngine and will no longer receive any location updates after the onDestroy() method is called. For information about MVP and twist, see this tutorial. Now it's time to build and run the app to see the result. Customizing the Map Appearance Open the MainActivity.kt file, and change the class header by adding the MapboxMap.OnMapClickListener interface. Now Android Studio will complain, tell you to implement the required method. When you choose to implement the method, make sure you choose onMapClick and click OK. After clicking the OK button, Android Studio will automatically add this new override method. override fun onMapClick(point: LatLng) { } First you need to change the settings function by enhanced the following code at Add. map.addOnMapClickListener(this) You must add it because you want the map to respond only to user taps when the user's location is visible on the map, and because this class implements the MapboxMap.OnMapClickListener interface. Implement. the onMapClick method by adding the following code. map.addMarker(MarkerOptions().position(point)) Here you add a tag to a particular position by passing the point variable. Now build and run the app to see the tag. :] It's cool to finally see a tag on the map, but it doesn't look right when multiple tags are added across the map, right? You can fix it by adding the following code in the onMapClick method. if(!map.markers.isEmpty()){ map.clear() } This will remove any tag before adding a new one to the map. Build and run the app to see the result. You can further customize the tag by adding different cool things for it. map.addMarker(MarkerOptions().setTitle(I'm a tag :]).position(point)) Here you add a string title that appears on the tag. Go ahead and run the program to see it for yourself. You can also add a snippet to the tag. map.addMarker(MarkerOptions().setTitle(I'm a tag :]).setSnippet (This is a snippet over this tag that will appear here).position(point)) Build and run now the app to see the snippet. Using the Mapbox Navigation API in this section, you will learn how to use the Mapbox navigation api to add turn by turning navigation within the app. Align the user between locations open the MainActivity.kt file, and add the following code at the top, before onCreate. var navigationMapRoute: NavigationMapRoute? = null var currentRoute: DirectionsRoute? =null Here, you declare the required variables, which you will use later for navigation. Next, you must create a function named getRoute that takes two arguments, originPoint, and endPoint. Be sure to enter the org.mapbox.Point class, instead of the default Android implementation. fun GetRoute(origin Point: Point, endpoint: Point) { } You will use this function for user navigation later. Now you need to change the onMapClick function by adding the following code. checkLocation() originLocation?. run { fall startPoint = Point.fromLngLat(longitude, latitude) fall endPoint = Point.fromLngLat(point.longitude, point.latitude) getRoute(startPoint, endPoint) } And create the checkLocation function, which will try to set the originLocation field to the last known location, if there is no location present. @SuppressLint(MissingPermission) private fun checkLocation() { if (originLation == null) { map.locationComponent.lastKnownLocation?. run { originLocation = this } } In this code, you initialize startPoint by passing the passing originLocation the length and latitude, while initializing endpoint by passing point the same. Finally, you'll call GetRoute by passing these two parameters: startPoint and endPoint. Right now, the getRoute feature isn't doing anything. You can have it do something by adding the following code. Be sure to use the when you enter callback, call, and response types. NavigationRoute.builder(this) // 1 .accessToken(Mapbox.getAccessToken()!!) // 2 .origin (origin Point) // 3 .destination(endPoint) // 4 .build() // 5 5 : Callback<DirectionsResponse> { /6 dominates fun onFailure(call:<DirectionsResponse>Call: Call, t: Throwable) { } override fun onResponse(call:<DirectionsResponse>Call, response:<DirectionsResponse>Response) { } }) Let's go through that code block NavigationRoute.builder: Run the navigation by passing the current context first. accessToken: This gets Mapbox an access token origin: Set the starting point for this navigation destination: Set the end/destination point for this navigation build: Call it to build up the navigation getRoute: Call it when you want to handle success and failure instances Now you want to handle the onFailure instance by adding the following code. Log.d(MainActivity, t.localizedMessage) Here the app will print a message in the Logcat when a failure case happens. Finally, you must handle the onResponse case by adding the code below. if (navigationMapRoute != null) { navigationMapRoute?. updateRouteVisibilityTo(false) } else { navigationMapRoute = NavigationMapRoute(null, mapbox, map) } currentRoute = response.body()?. routes()?. first() if (currentRoute != null) { navigationMapRoute?. addRoute(currentRoute) } The navigationMapRoute field is responsible for drawing a line on the map, from the current location to the destination. You first check if navigationMapRoute is empty or not. If it is not empty, you must remove the route. Otherwise, initialize navigationMapRoute by passing three parameters: MapboxNavigation: This is a navigation instance of MapboxNavigation that you pass if you want to regroup during the navigation session. For example, you will pass a zero. map box: This is the map you want to draw the route to on top. MapboxMap: This will apply the route. Initialize currentRoute by accessing the response body and getting the first route from the routes() list. Finally, before you add any route, check to see if there is already a route and then add that route to navigationMapRoute. Now build and run the app to see the user's route on the map! :] It's cool to see that blue line on the map showing the user route, right? But it would be really cool if you can actually start the navigation from a starting point until you reach the destination. Read on to find out how. :] Within btnNavigate, the setOnClickListener method is where you will add the following code. val navigationLauncherOptions = NavigationLauncherOptions.builder() // 1 .directionsRoute(currentRoute) // 2 .shouldSimulateRoute(true) // 3 .build() NavigationLauncher.startNavigation(this, navigationLauncherOptions) // 4 Let's go through that code. You create a constant navigationLauncherOptions, which you then initialize with NavigationLauncherOptions. NavigationLauncherOptions is part of Mapbox's classes, which allows you to build the navigation route. DirectionsRoute: This is the actual route you initialized earlier. It is used to mimic the actual turn through turn navigation. You can enable or disable</DirectionsResponse> </DirectionsResponse> </DirectionsResponse > </DirectionsResponse > </DirectionsResponse > as you like, and then call build (). NavigationLauncher: This is a class that you can use to start the navigation by passing two parameters: Context and navigationLauncherOptions. Now build and run the program. It feels great to finally see the navigation working, but what will happen when you tap the navigation button without even adding any tag on the map? The result is that the program will collapse. Here's how you can fix it. Open the MainActivity.kt file and add the following code to OnCreate. btnNavigate.isEnabled = false Here you first disable the navigation button to prevent the navigation from starting before adding any tag on the map. Now you can enable it in the getRoute method, at the end of the onResponse block. btnNavigate.isEnabled = where You can now build and run the app. You can only tap the navigation button if there is a tag visible on the map. Otherwise, the button will be disabled. Where to go from here? You can download the completed project files by clicking the Download Materials button at the top or bottom of this tutorial. You can do even more with Mapbox Navigation API, for example: Changing the language when you start turning through rotate navigation. Changes the colors of the navigation UI. Setting listeners to know if the user is still in the route or has reached the destination. To learn more about Mapbox Navigation API, check out the documentation. We hope you enjoy this tutorial, and if you have any questions or comments, please join the forum discussion below! Maps & Location Android & Kotlin Tutorials The raywenderlich.com is the easiest way to keep track of everything you need to know as a mobile developer. Get a weekly digest of our tutorials and courses, and receive a free in-depth email course as a bonus! 3.9/5 8 rating ratings