Near-By Android App in Kotlin

1. Introduction

a. Overview

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA . On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

● A flexible Gradle-based build system
● A fast and feature-rich emulator
● A unified environment where you can develop for all Android devices
● Apply Changes to push code and resource changes to your running app without restarting your app
● Code templates and GitHub integration to help you build common app features and import sample code
● Extensive testing tools and frameworks
● Lint tools to catch performance, usability, version compatibility, and other problems
● C++ and NDK support
● Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine This app helps you to make a list of groceryitems alongwith its price and quantity.

b. Purpose

People need the information about basic necessity like atms, hotels, and other govt. institutions in whatever area they are in.This application helps in finding out the near by places of their location with ease.

2. Literature Survey

a. Existing Problem

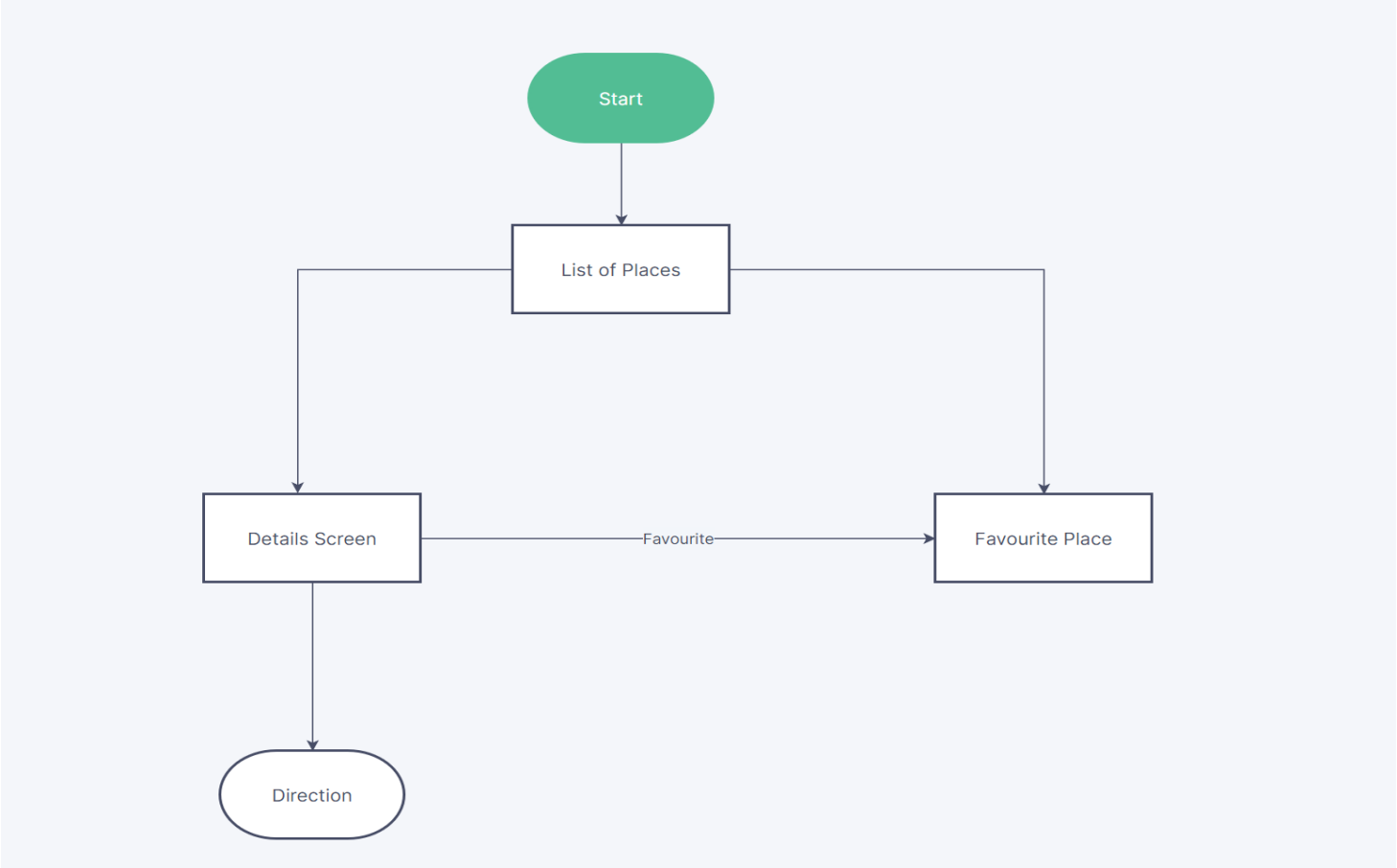
Users may need to find the information about nearby commodities and it is a hassle to find a person for direction in busy time.Sometimes, people tends to give wrong direction and it wastes a lot of time to find the right direction.

b. Proposed Solution

To overcome the existing problem, Near BY is created to help users to find commodities such as hotels and atms in 10KM radius using google cloud platform for accurate information.

3. Theoritical Analysis

a. Block Diagram



b. Hardware/ Software Designing

- CPU : Ryzen 7 5800H
- RAM : 16GB
- Operating System : Windows 10
- Developing tools : Android Studio, Google Cloud
- Language Used : Kotlin , Xml

4. EXPERIMENTAL INVESTIGATIONS

In this project MVVM (Model View ViewModel) was used for architectural patterns, Room for database, Coroutines and RecyclerView to display the list of items.

LiveData:
LiveData is an observable data holder class. Unlike a regular observable, LiveData is lifecycle-aware, meaning it respects the lifecycle of other app components, such as activities, fragments, or services. This awareness ensures LiveData only updates app component observers that are in an active lifecycle state.

ViewModel:
The ViewModel class is designed to store and manage UI-related data in a lifecycle conscious way. The ViewModel class allows data to survive configuration changes such as screen rotations.
The Android framework manages the lifecycle of UI controllers, such as activities and fragments. The framework may decide to destroy or re-create a UI controller in response to certain user actions or device events that are completely out of your control.

Repository:
A class that you create that is primarily used to manage multiple data sources.

Entity:
Marks a class as an entity. This class will have a mapping SQLite table in the database.

Each entity must have at least 1 field annotated with PrimaryKey. You can also use primaryKeys() attribute to define the primary key.

Each entity must either have a no-arg constructor or a constructor whose parameters match fields (based on type and name). Constructor does not have to receive all fields as parameters but if a field is not passed into the constructor, it should either be public

or have a public setter. If a matching constructor is available, Room will always use it. If you don't want it to use a constructor, you can annotate it with Ignore.

DAO:
Data Access Objects are the main classes where you define your database interactions. They can include a variety of query methods.

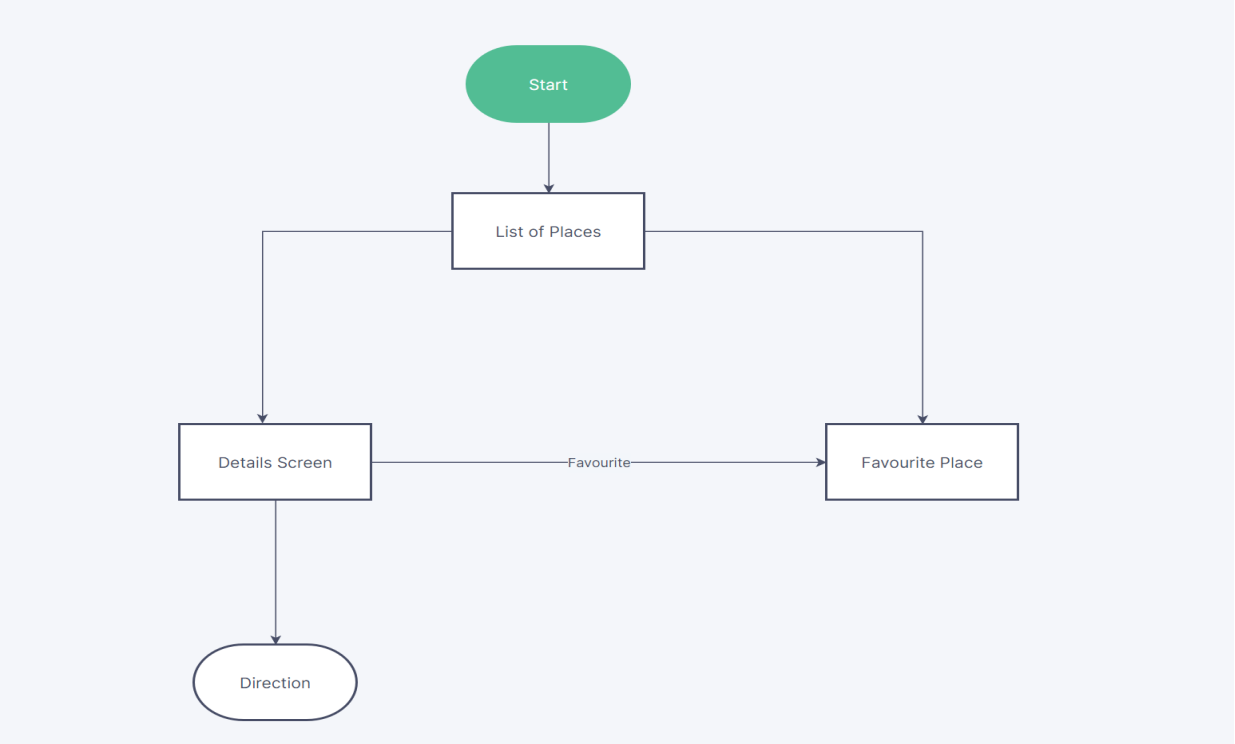
The class marked with @Dao should either be an interface or an abstract class. At compile time, Room will generate an implementation of this class when it is referenced by a Database.

An abstract @Dao class can optionally have a constructor that takes a Database as its only parameter.

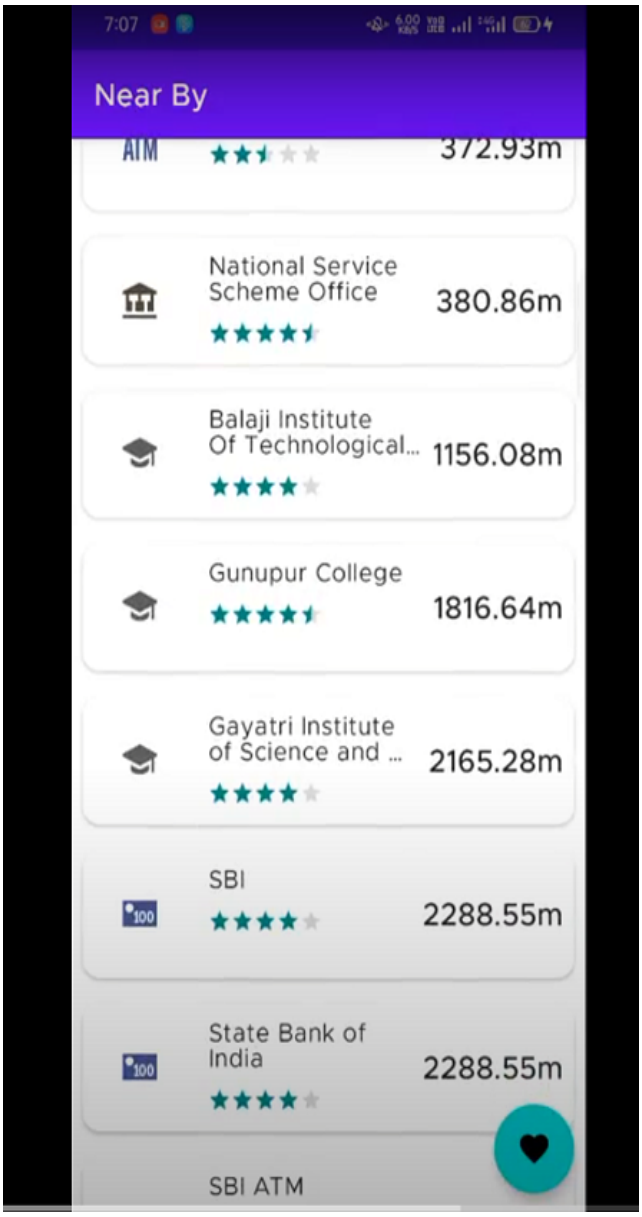
RecyclerView:
A flexible view for providing a limited window into a large data set.
It is a container and is used to display the collection of data in a large amount of dataset that can be scrolled very effectively by maintaining a limited number of views.

Coroutines:
Coroutines are lightweight thread, we use a coroutine to perform an operation on other threads, by this our main thread doesn't block and our app doesn't crash.

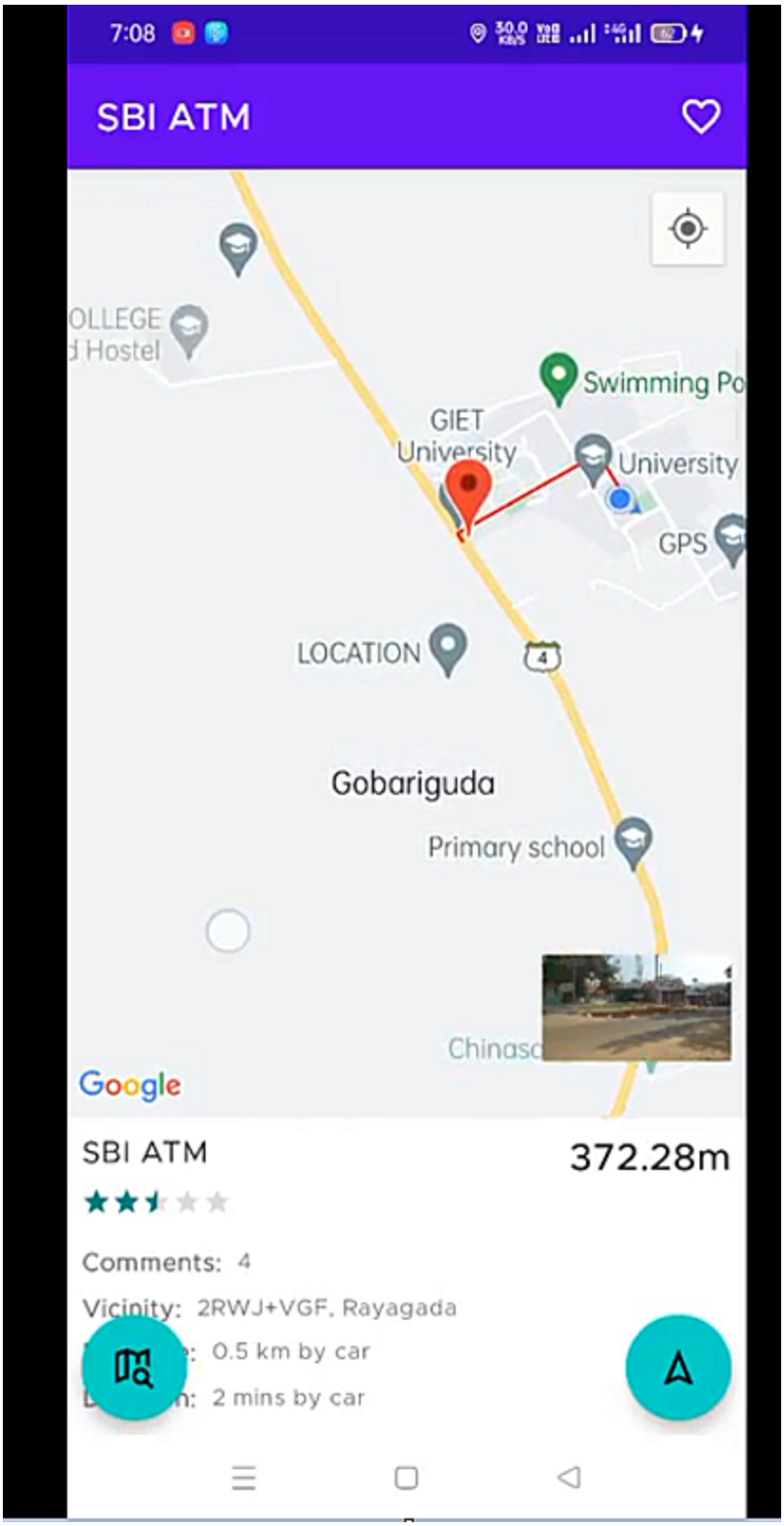
5.FLOWCHART



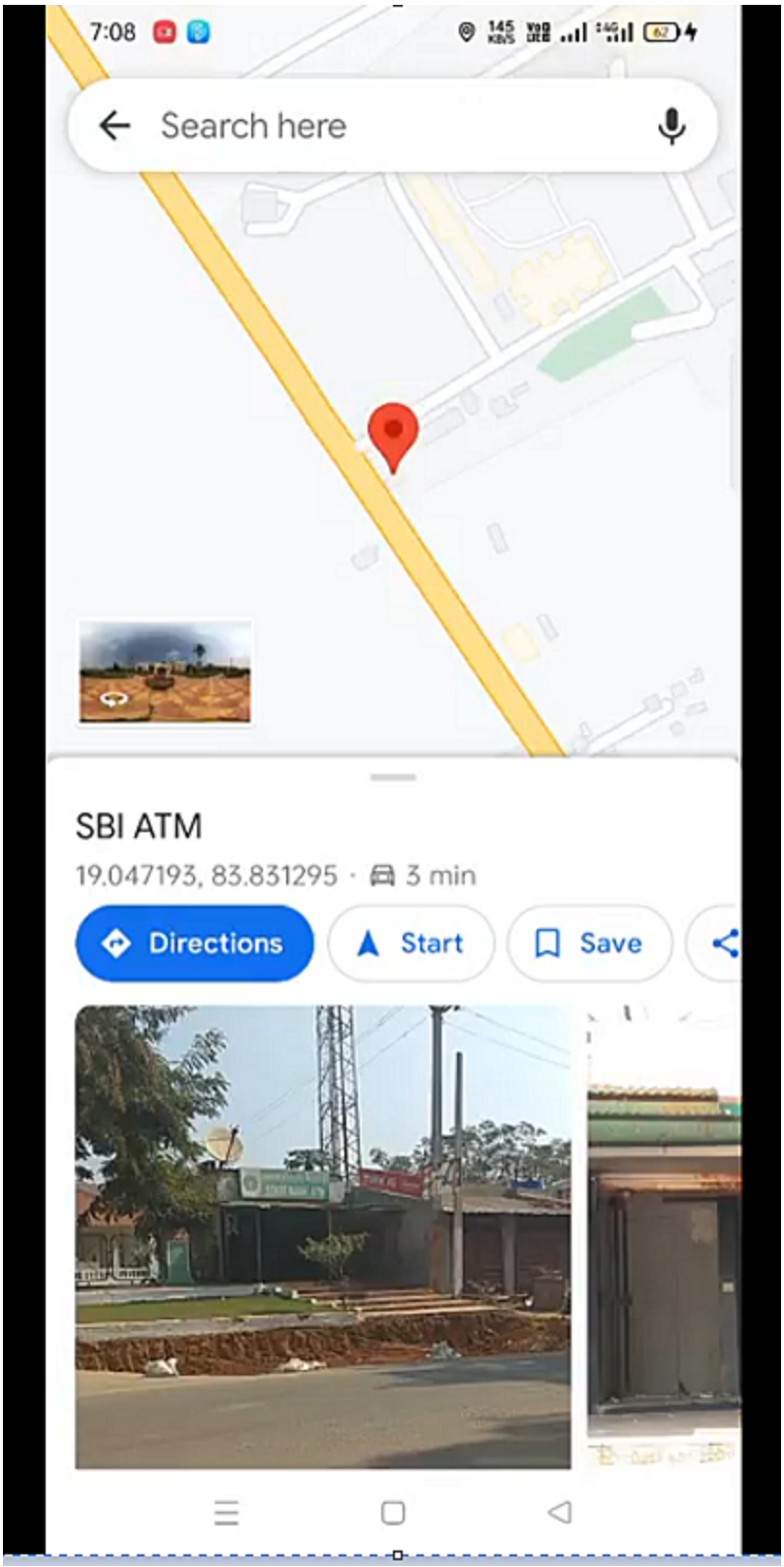
6.RESULT



List of Locations in 10KM radius



Detail Screen of a place



Direction page

7. Advantages and Disadvantages

Advantages:

- Near by places in 10KM radius
- App act as a guide

Disadvantages:

- This app shows only the places name , not the quality.
- People should be be careful while checking the place

8. Applications:

- This is widely used by old people for direction
- Travellers can use this application for hotels in 10km radius

9. Conclusion

This Android app development project helped me to learn concepts like Room Database, Coroutines, MVVM, etc. Working on this project made me confidentenough to apply my knowledge on android app development and to create an app for existing problem.
I have used Kotlin to build this application and used android studio as a medium.

10.Future Scope

This application can have option custom search.

11.BIBILOGRAPHY

- https://developer.android.com/guide
- https://developers.google.com/codelabs/maps-platform/maps-platform-101-android#5

Appendix

NearbyPlacesActivity

```
package com.example.nearby.ui.listNearbyPlaces

import android.Manifest
import android.annotation.SuppressLint
import android.content.Intent
import android.location.Location
import android.net.Uri
import android.os.Bundle
import android.provider.Settings
import android.view.View
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.nearby.BuildConfig
import com.example.nearby.R
import com.example.nearby.adapter.AdapterPlaces
import com.example.nearby.databinding.ActivityNearbyPlacesBinding
import com.example.nearby.models.Places
import com.example.nearby.ui.detailsPlace.DetailsPlaceActivity
import com.example.nearby.ui.favoritesPlaces.FavoritesPlacesActivity
import com.example.nearby.util.Constants.Companion.KEY_API
import com.example.nearby.util.Dialog
import com.example.nearby.util.Features
import com.example.nearby.util.NetworkResult
import com.google.android.gms.location.FusedLocationProviderClient
import com.google.android.gms.location.LocationRequest
import com.google.android.gms.location.LocationServices
import com.google.android.gms.tasks.CancellationTokenSource
import com.google.android.gms.tasks.Task
import com.google.android.material.snackbar.Snackbar
import com.karumi.dexter.Dexter
import com.karumi.dexter.MultiplePermissionsReport
import com.karumi.dexter.PermissionToken
import com.karumi.dexter.listener.PermissionRequest
import com.karumi.dexter.listener.multi.CompositeMultiplePermissionsListener
import com.karumi.dexter.listener.multi.DialogOnAnyDeniedMultiplePermissionsListener
import com.karumi.dexter.listener.multi.MultiplePermissionsListener
import dagger.hilt.android.AndroidEntryPoint
import es.dmoral.toasty.Toasty
import javax.inject.Inject


@AndroidEntryPoint
class NearbyPlacesActivity : AppCompatActivity(), MultiplePermissionsListener,
```

```
View.OnClickListener {

    private val TAG = "NearbyPlacesActivity"

    //Permissions
    private var allPermissionsListener: MultiplePermissionsListener? = null

    //Binding
    private lateinit var binding: ActivityNearbyPlacesBinding

    //ViewModel
    @Inject
    lateinit var nearbyPlacesViewModel: NearbyPlacesViewModel

    //Features
    private val features by lazy { Features() }

    //Dialog
    private val dialog by lazy { Dialog(this@NearbyPlacesActivity) }

    //Fused Location Provider API
    private val fusedLocationClient: FusedLocationProviderClient by lazy {
LocationServices.getFusedLocationProviderClient(applicationContext) }

    // Allows Cancel Location Request
    private var cancellationTokenSource = CancellationTokenSource()

    //Location
    private var location: Location? = null

    //Adapter
    private val adapterPlacesNearby by lazy { AdapterPlaces(arrayListOf()) }

    override fun onCreate(savedInstanceState: Bundle?) {
        binding = ActivityNearbyPlacesBinding.inflate(layoutInflater)
        setContentView(binding.root)
        super.onCreate(savedInstanceState)

        binding.recyclerViewPlacesNearby.adapter = adapterPlacesNearby
        binding.recyclerViewPlacesNearby.layoutManager =
LinearLayoutManager(this@NearbyPlacesActivity)
        binding.recyclerViewPlacesNearby.hasFixedSize()

        adapterPlacesNearby.placesClicklistener = object :
AdapterPlaces.PlacesClicklistener{
            override fun onPlacesClick(place: Places, position: Int, view: View) {

                if (features.checkPermission(this@NearbyPlacesActivity)) {

                    if(features.isConnected(this@NearbyPlacesActivity)){
```

```
                        val intent = Intent(this@NearbyPlacesActivity,
DetailsPlaceActivity::class.java)
                        intent.putExtra("place", place)
                        startActivity(intent)

                    }else{

                        Toasty.warning(this@NearbyPlacesActivity, "No internet.",
Toast.LENGTH_LONG, true).show()

                    }

                }else{

                    showSnackbar("Permission was denied, but is required for core
functionality.", "Settings") {
                        val intent = Intent()
                        intent.action = Settings.ACTION_APPLICATION_DETAILS_SETTINGS
                        val uri = Uri.fromParts("package", BuildConfig.APPLICATION_ID,
null)
                        intent.data = uri
                        intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK
                        startActivity(intent)
                    }

                }

            }
        }

        if (features.checkPermission(this@NearbyPlacesActivity)) {

            requestCurrentLocation()

        }else{

            checkPermissions()

        }

        binding.floatingActionFavorites.setOnClickListener(this@NearbyPlacesActivity)

    }

    @SuppressLint("MissingPermission")
    private fun requestCurrentLocation() {
        val currentLocationTask: Task<Location> =
fusedLocationClient.getCurrentLocation(
            LocationRequest.PRIORITY_HIGH_ACCURACY,
```

```
            cancellationTokenSource.token
        )

        currentLocationTask.addOnCompleteListener { task: Task<Location> ->
            if (task.isSuccessful && task.result != null) {
                location = task.result
                findPlacesNearby(task.result)
            } else {
                val exception = task.exception
                Toasty.info(this@NearbyPlacesActivity, "Location (failure):
$exception", Toast.LENGTH_LONG).show()
            }
        }

    }

    private fun findPlacesNearby(result: Location, type: String = ""){

        nearbyPlacesViewModel.getPlaces("${result.latitude},${result.longitude}",
"1000000", type, KEY_API)

        nearbyPlacesViewModel.places.observe(this@NearbyPlacesActivity) {

            when(it) {

                is NetworkResult.Success -> {

                    dialog.dismissDialog()

                    if(it.data?.results.isNullOrEmpty()){

                        Toasty.info(this@NearbyPlacesActivity, "There was a problem in
the request.", Toast.LENGTH_LONG, true).show()

                    }else{

                        adapterPlacesNearby.setData(it.data?.results?.sortedBy {
                                obj ->
features.distFrom(location?.latitude!!.toFloat(), location?.longitude!!.toFloat(),
obj.geometry?.location?.lat!!.toFloat(), obj.geometry?.location?.lng!!.toFloat()) }
                                as MutableList<Places>, location)

                    }

                }

                is NetworkResult.Error -> {

                    dialog.dismissDialog()
```

```
                    it.message?.let { it1 -> Toasty.error(this@NearbyPlacesActivity,
it1, Toast.LENGTH_LONG, true).show() }

                }

                is NetworkResult.Loading -> {

                    dialog.showDialog()

                }

            }

        }

    }

    override fun onClick(p0: View?) {

        when(p0?.id){

            R.id.floatingActionFavorites -> {

                val intent = Intent(this@NearbyPlacesActivity,
FavoritesPlacesActivity::class.java)
                startActivity(intent)

            }

        }

    }

    private fun showSnackbar(mainTextString: String, actionString: String, listener:
View.OnClickListener) {
        Snackbar
            .make(findViewById(android.R.id.content), mainTextString,
Snackbar.LENGTH_INDEFINITE)
            .setAction(actionString, listener).show()
    }

    private fun checkPermissions(){

        val dialogMultiplePermissionsListener: MultiplePermissionsListener =
            DialogOnAnyDeniedMultiplePermissionsListener.Builder
                .withContext(this@NearbyPlacesActivity)
                .withTitle("Permission")
                .withMessage("\n" +
                        "The requested permissions are necessary for the functions of
the app.")
```

```
                .withButtonText("Ok")
                .withIcon(R.mipmap.ic_launcher)
                .build()

        allPermissionsListener = CompositeMultiplePermissionsListener(
            this@NearbyPlacesActivity,
            dialogMultiplePermissionsListener
        )

        Dexter.withActivity(this@NearbyPlacesActivity)
            .withPermissions(Manifest.permission.ACCESS_FINE_LOCATION,
                Manifest.permission.ACCESS_COARSE_LOCATION)
            .withListener(allPermissionsListener)
            .check()

    }

    override fun onPermissionsChecked(report: MultiplePermissionsReport?) {
        if (report != null) {
            if (report.areAllPermissionsGranted()){
                requestCurrentLocation()
            }else{
                showSnackbar("Permission was denied, but is required for core
functionality.", "Settings") {
                    val intent = Intent()
                    intent.action = Settings.ACTION_APPLICATION_DETAILS_SETTINGS
                    val uri = Uri.fromParts("package", BuildConfig.APPLICATION_ID,
null)
                    intent.data = uri
                    intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK
                    startActivity(intent)
                }
            }
        }
    }

    override fun onPermissionRationaleShouldBeShown(permissions:
MutableList<PermissionRequest>?, token: PermissionToken?) {}

    override fun onStop() {
        super.onStop()
        cancellationTokenSource.cancel()
    }

}
```

Here is my GitHub link:
https://github.com/smartinternz02/SPSGP-87896-Virtual-Internship---Android-Application-Development-Using-Kotlin

Google Developer Profile:
https://g.dev/blez-coder

Demo Drive link:
https://drive.google.com/file/d/1FcGSKZ99qpV33VSotPe9Z9olzUncjwG0/view?usp=sharing