Experiment 2: Layout with Search Bar, List View, and Map Fragment

Objective

To design and implement a user interface that integrates a search bar, a list view, and a map fragment in a single Android screen.

Prerequisites

- Basic knowledge of Android Studio (or similar mobile development environment)
- Familiarity with XML layout design
- Understanding of Android Activities and Fragments
- Basic knowledge of Google Maps API (or another map service)
- Experience with Intents and data passing between components
- Knowledge of RecyclerView and adapters

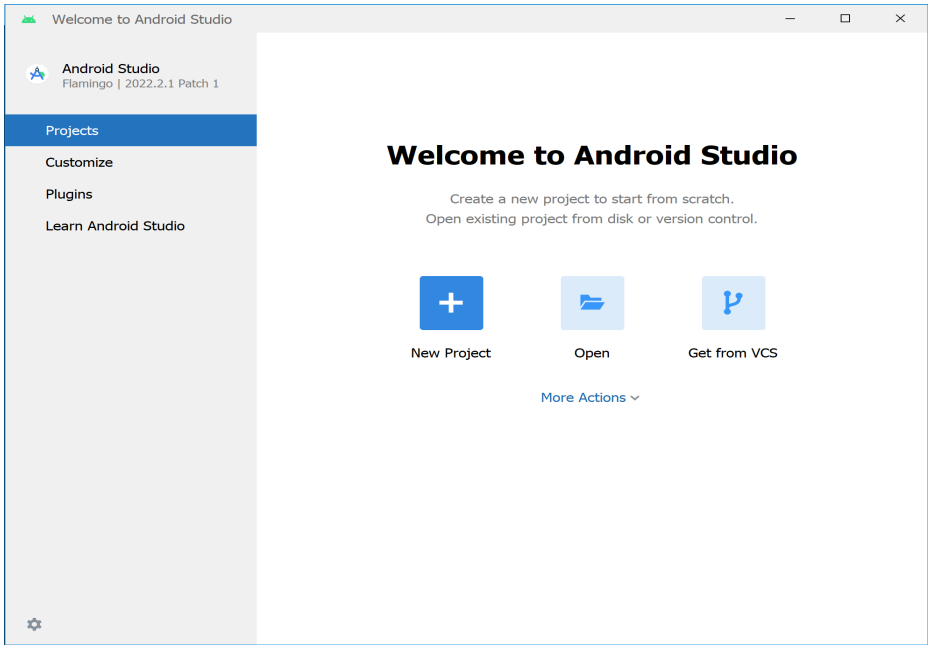
Expected Outcome

By the end of this lab, participants will be able to:

- Design and implement a cohesive layout containing a search bar, list view, and map fragment
- Capture and process user input from the search bar
- Integrate map functionality to display locations based on user selection
- Dynamically update the list view based on search results
- Manage data flow between the search bar, list/list-adapter, and map fragment
- Design an intuitive and user-friendly interface using Activities and Fragments
- Gain practical experience working with fragment lifecycles

Procedure

Step 1: Create a New Android Project

1. Open Android Studio and create a new project (Empty Activity / Empty Compose Activity or appropriate template).
2. Name the project (for example, `SearchMapDemo`) and configure package name, language (Java/Kotlin), and minimum SDK.
3. Wait for Gradle sync to complete before proceeding.
4. 

Step 2: Add Required Dependencies

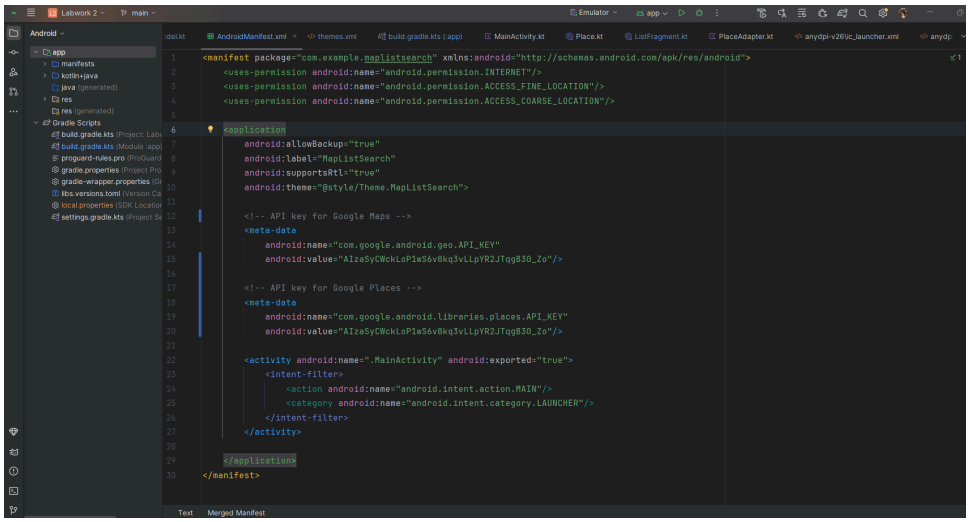
1. Open the `build.gradle` (Module: app) file.
2. Ensure that the dependencies for Google Maps (e.g., `com.google.android.gms:play-services-maps`) and RecyclerView are included.
3. Sync the project after adding or verifying dependencies.

Step 3: Design the Main Layout (XML)

1. Open `activity_main.xml` (or equivalent layout).
2. Create a parent layout (e.g., `CoordinatorLayout` or `LinearLayout`) with vertical orientation.
3. Add the following UI elements:
    - A `SearchView` or `EditText` at the top for user search input.

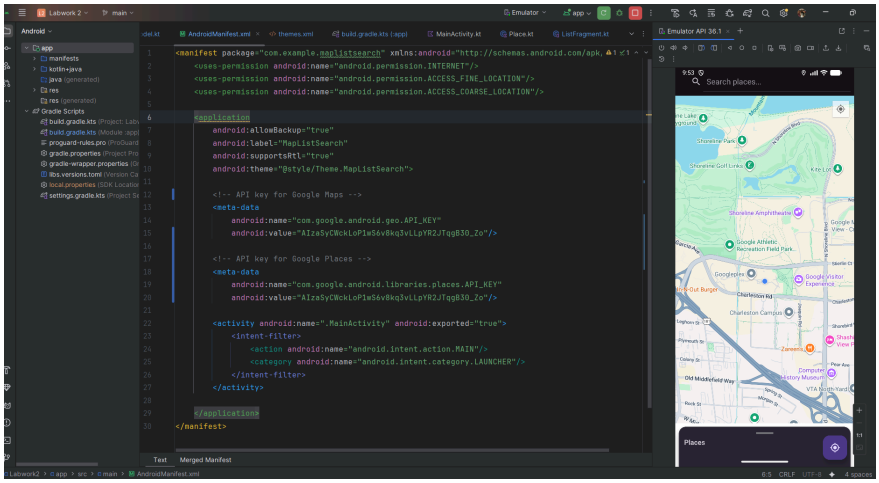
- A `RecyclerView` or `ListView` below the search bar to display search results or locations.
- A `FragmentContainerView` or `FrameLayout` at the bottom to host the `MapFragment` / `SupportMapFragment`.

4. Assign meaningful IDs to each element (e.g., `search_view`, `recycler_view_locations`, `map_fragment_container`).



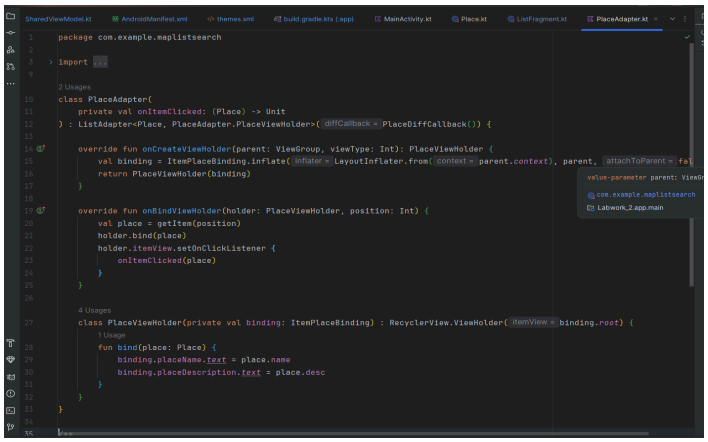
Step 4: Implement the Map Fragment

1. Create a new Fragment class (e.g., `LocationMapFragment`) extending `Fragment` and implementing `OnMapReadyCallback`.
2. In `onCreateView`, inflate a layout containing a `SupportMapFragment` or `MapView`.
3. Initialize the map in `onMapReady` and set default camera position.
4. Implement a method in the fragment (e.g., `showLocationOnMap(LatLng position)`) to update the marker and camera based on a selected location.



Step 5: Implement the List / RecyclerView

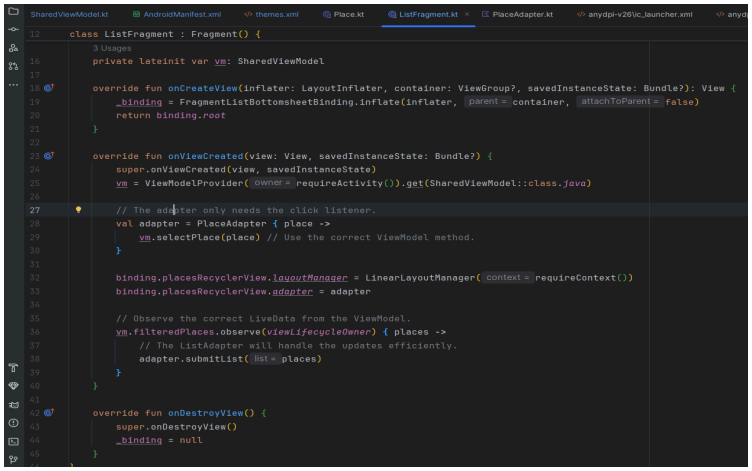
1. Define a data model class (e.g., `LocationItem`) with fields such as name, description, and coordinates.
2. Create a custom adapter for `RecyclerView` or `ListView` to bind the list of `LocationItem` objects.
3. In the adapter, handle item click events to notify the Activity or Map Fragment when a location is selected.



```
package com.example.maplistsearch

import ...

2 Usages
class PlaceAdapter(
    private val onItemClicked: (Place) -> Unit
) : ListAdapter<Place, PlaceAdapter.PlaceViewHolder>( diffCallback = PlaceDiffCallback()) {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): PlaceViewHolder {
        val binding = ItemPlaceBinding.inflate( inflater = LayoutInflater.from( context = parent.context), parent, attachToParent = fa
        return PlaceViewHolder(binding)
    }

    override fun onBindViewHolder(holder: PlaceViewHolder, position: Int) {
        val place = getItem(position)
        holder.bind(place)
        holder.itemView.setOnClickListener {
            onItemClicked(place)
        }
    }

    4 Usages
    class PlaceViewHolder(private val binding: ItemPlaceBinding) : RecyclerView.ViewHolder( itemView = binding.root) {
        1 Usage
        fun bind(place: Place) {
            binding.placeName.text = place.name
            binding.placeDescription.text = place.desc
        }
    }
}
```

Step 6: Integrate Search Functionality

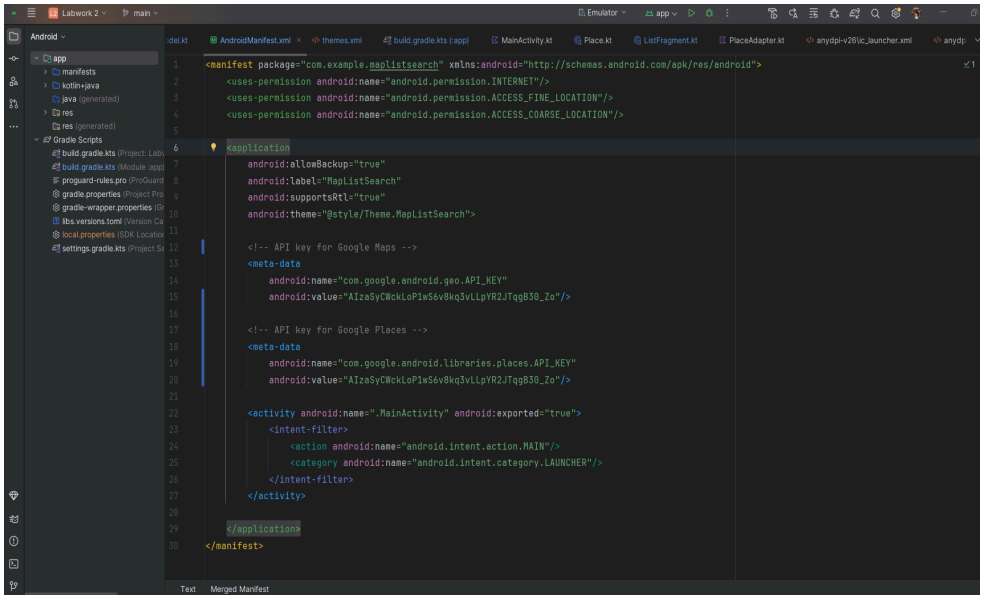
1. In `MainActivity`, set up a listener on the `SearchView` (e.g., `setOnQueryTextListener`).
2. Filter the data list based on the search query and update the adapter accordingly.
3. Ensure the list view or RecyclerView reflects filtered results in real time as the user types.



```
class ListFragment : Fragment() {
    3 Usages
    private lateinit var vm: SharedViewModel

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View {
        _binding = FragmentListBottomsheetBinding.inflate(inflater, parent = container, attachToParent = false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        vm = ViewModelProvider( owner = requireActivity()).get(SharedViewModel::class.java)

        // The adapter only needs the click listener.
        val adapter = PlaceAdapter { place ->
            vm.selectPlace(place) // Use the correct ViewModel method.
        }

        binding.placesRecyclerView.layoutManager = LinearLayoutManager( context = requireContext())
        binding.placesRecyclerView.adapter = adapter

        // Observe the correct LiveData from the ViewModel.
        vm.filteredPlaces.observe(viewLifecycleOwner) { places ->
            // The ListAdapter will handle the updates efficiently.
            adapter.submitList( list = places)
        }
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
```

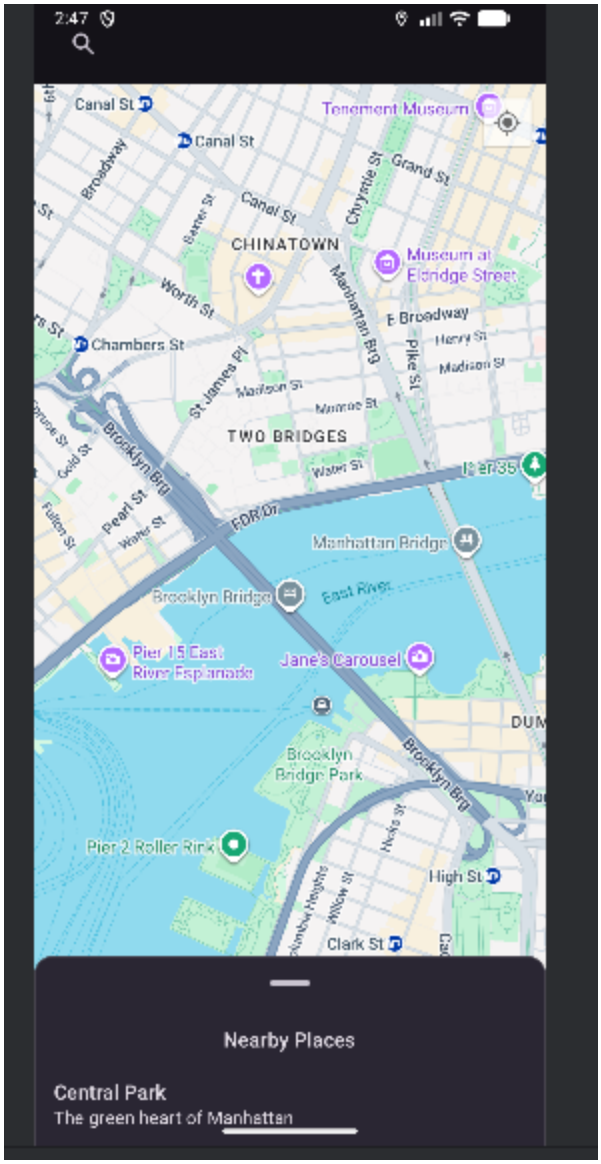
Step 7: Connect List Selections to the Map

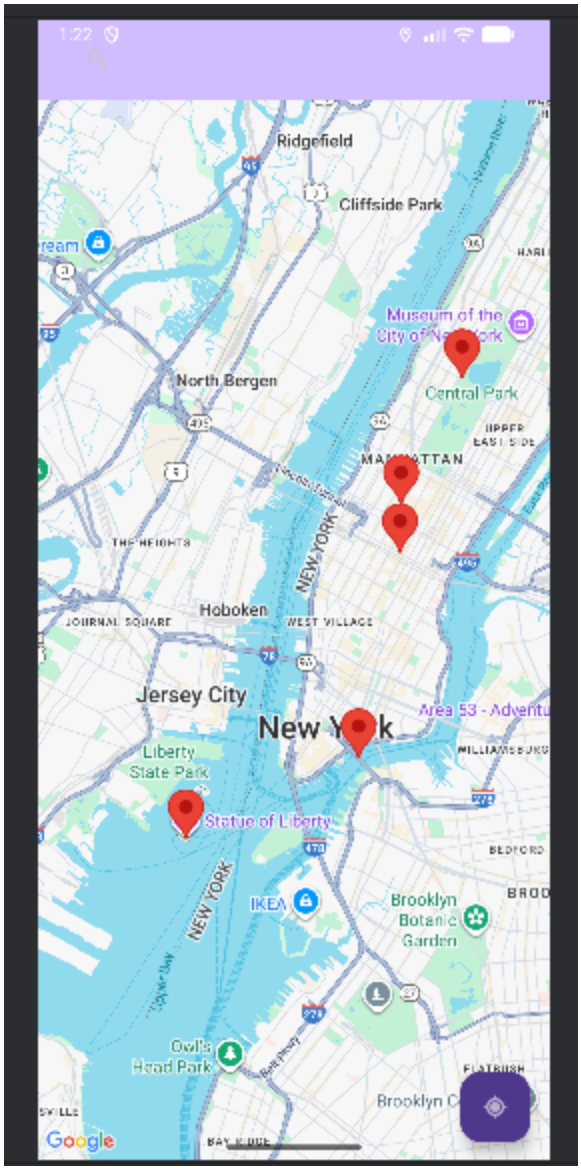
1. In the Activity, implement a callback interface for item selection events from the adapter.
2. When an item is selected, retrieve its coordinates and call the corresponding method on `LocationMapFragment` to update the marker and camera position.
3. Optionally, scroll or highlight the selected item in the list for better user feedback.

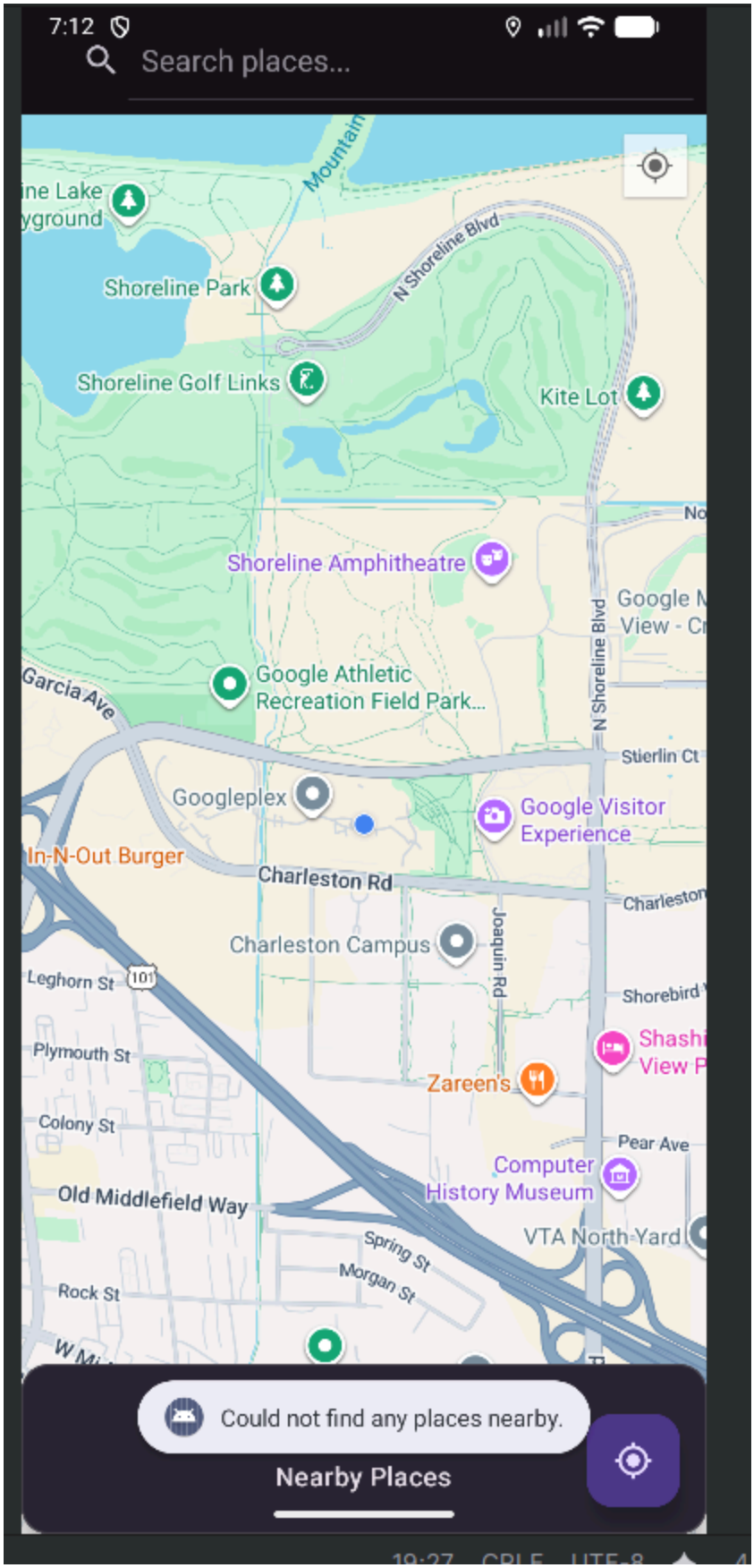


Step 8: Test and Refine the Interface

1. Run the project on an emulator or physical device.
2. Verify that:
    - The search bar accepts input and filters the list correctly.
    - Tapping an item in the list updates the map and displays the correct location.
    - Rotations or configuration changes preserve the state of the list and map (or are handled appropriately).
3. Refine padding, colors, and layout weights to ensure a clean and user-friendly UI.


2:47
Canal St
Canal St
Tenement Museum
Broadway
Canal St
Grand St
Chrystie St
CHINATOWN
Museum at Eldridge Street
Worth St
Manhattan Brg
E Broadway
Chambers St
Henry St
Pike St
Madison St
St James Pl
Madison St
Monroe St
TWO BRIDGES
Brooklyn Brg
Water St
I-278/35
Pearl St
Water St
FDR Dr
Manhattan Bridge
Brooklyn Bridge
East River
Pier 15 East River Esplanade
Jane's Carousel
Brooklyn Bridge Park
Brooklyn Brg
Pier 2 Roller Rink
Hicks St
High St
Willow St
Clark St
Nearby Places
Central Park
The green heart of Manhattan


1:22
Ridgefield
Cliffside Park
Museum of the
City of New York
North Bergen
Central Park
MANHATTAN
NEW YORK
Hoboken
JOURNAL SQUARE
WEST VILLAGE
Jersey City
New York
Liberty
State Park
Statue of Liberty
IKEA
Brooklyn
Botanic
Garden
Owl's
Head Park
Google


7:12
Search places...
Mountain
ine Lake
yground
Shoreline Park
N Shoreline Blvd
Shoreline Golf Links
Kite Lot
Shoreline Amphitheatre
Google M
View - Cr
Google Athletic
Recreation Field Park...
Garcia Ave
N Shoreline Blvd
Stierlin Ct
Googleplex
Google Visitor
Experience
In-N-Out Burger
Charleston Rd
Charleston
Charleston Campus
Joaquin Rd
Leghorn St
101
Shorebird
Plymouth St
Shashi
View P
Zareen's
Colony St
Pear Ave
Computer
History Museum
Old Middlefield Way
VTA North Yard
Spring St
Morgan St
Rock St
Could not find any places nearby.
Nearby Places

Observations

- Note how search input affects the list content and how quickly results appear.
- Observe responsiveness of the map when switching between different list items.
- Record any lifecycle issues encountered with the fragment (e.g., map re-initialization on rotation).

Result

A functional Android screen that combines a search bar, a dynamic list (ListView / RecyclerView), and a map fragment, with all components integrated to work together as a single, user-friendly interface.