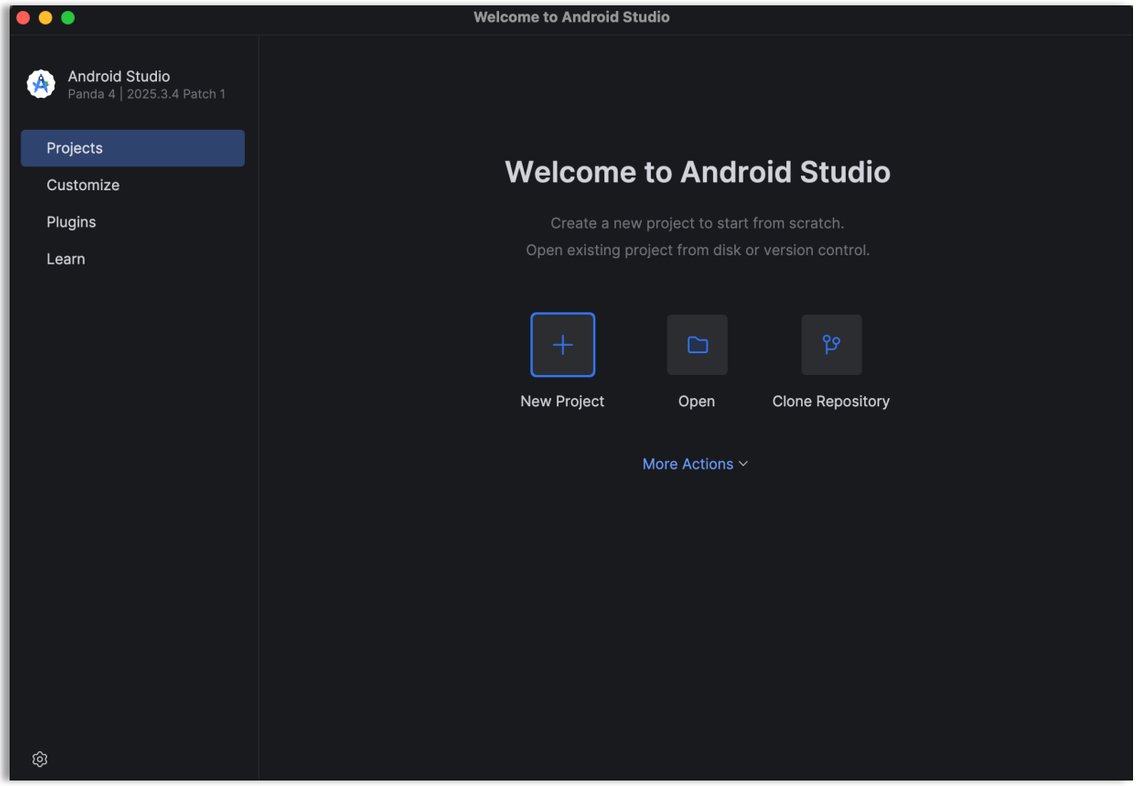
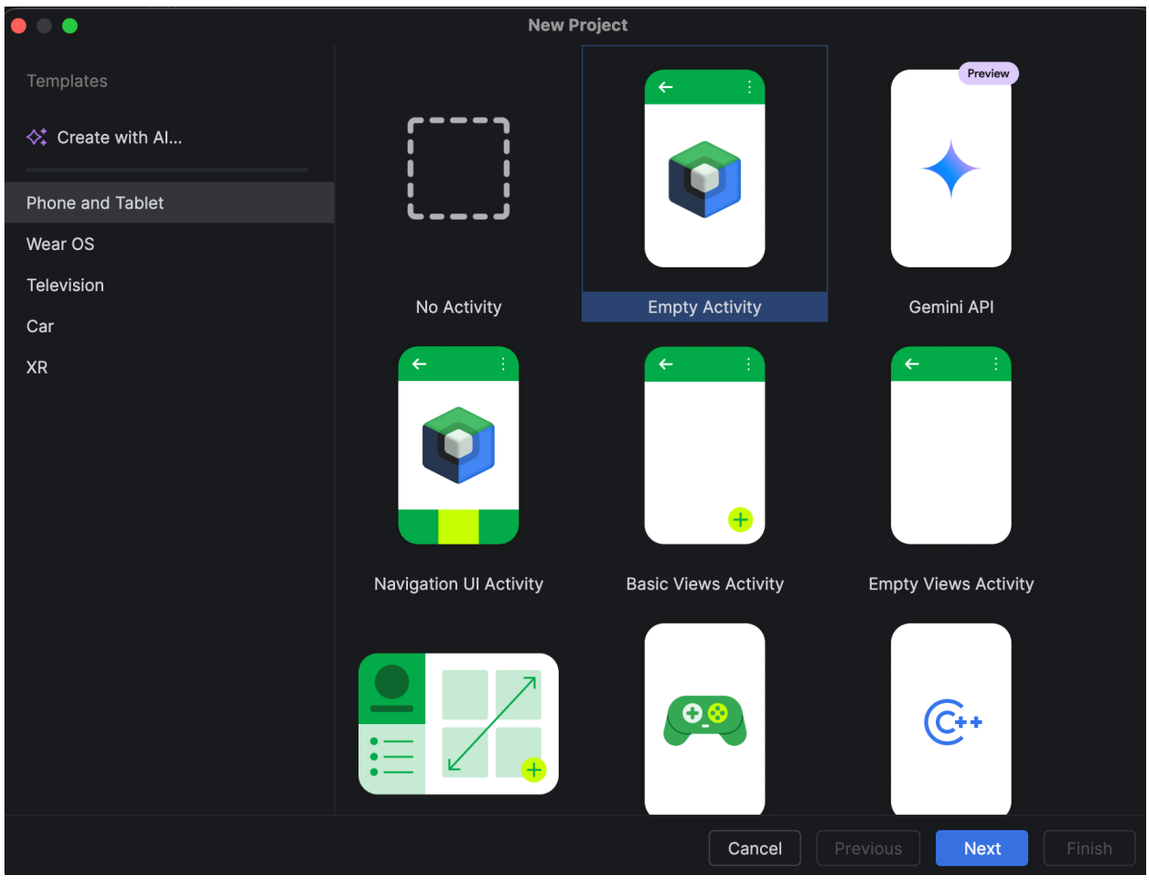
ListyCity – Instructions

1. Create a new project in Android Studio as shown below by clicking on ‘New Project’

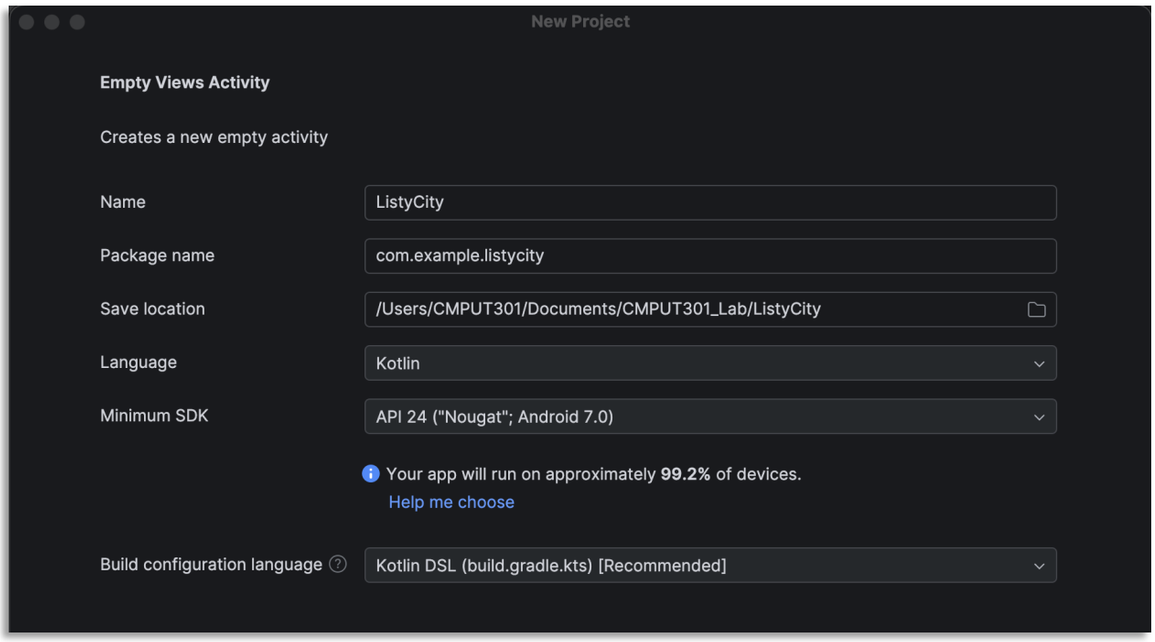


2. Select ‘Empty Activity’ under the ‘Phone and Tablet’ tab.
   Click Next.

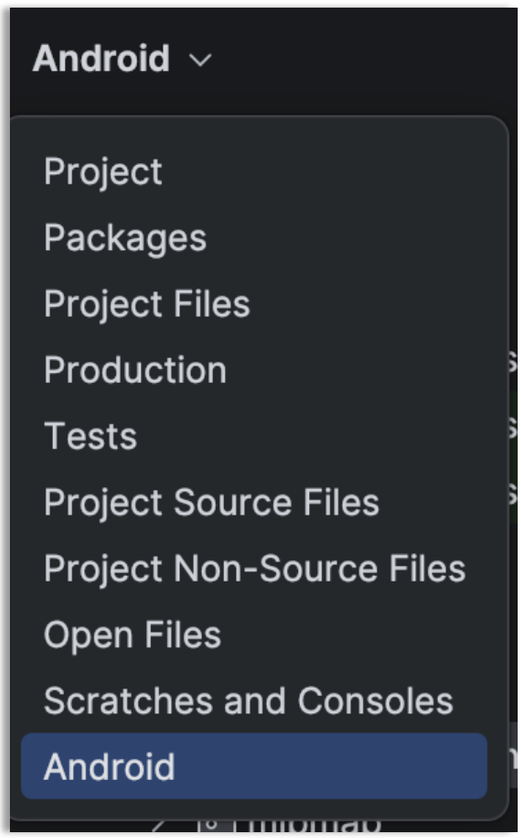


3. Configure your project
    I. Give the project a name
    II. Make sure the language is Kotlin
    III. The minimum API level should be enough so that the application runs on most devices

    Click Finish and wait for the project to build



4. Ensure the navigation view is set to Android



Navigate to ‘app / kotlin+java / com.[your package name] / MainActivity.kt’
This file contains the main Activity for the app. In a Jetpack Compose project, the UI is written using Kotlin using composable functions instead of XML layout files.

Look for the ‘onCreate()’ method. This is where the Activity starts and where Compose is told what UI to display using ‘setContent { … }’

5. Create the City Data
   In MainActivity.kt, create a simple class to store the city list. Add the following code below the MainActivity class:

```
class CityRepository {  1 Usage
    // Keep mutable app data private so other classes cannot change it directly
    private val _cities = mutableStateListOf(  2 Usages
        "Edmonton", "Vancouver", "Moscow",
        "Sydney", "Berlin", "Vienna",
        "Tokyo", "Beijing", "Osaka",
        "New Delhi"
    )

    // Get a read-only list for the UI to display
    val cities: List<String>  1 Usage
        get() = _cities

    fun addCity(city: String) {
        _cities.add(city)
    }
}
```

   This class stores the city names used by the app.

   The mutable list _cities is private, so other parts of the app cannot change it directly.
   The cities property gives the UI read-only access to the list.
   The addCity(city) function will later allow the app to add a city through a controlled function.

6. Connect the CityRepository to the Compose UI

   Inside MainActivity, look for the onCreate() method. The onCreate() method runs when the Android screen is created. Inside onCreate(), create a CityRepository object before setContent {…}:

```
class MainActivity : ComponentActivity() {  1 Usage

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        val cityRepository = CityRepository()
```

7. Pass City Data into CityListScreen
   I. Then, inside setContent { … }, replace the default Greeting(…) call inside with a call to CityListScreen:

```
        val cityRepository = CityRepository()

        setContent {
            ListyCityTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    CityListScreen(
                        cities = cityRepository.cities,
                        modifier = Modifier.padding( paddingValues = innerPadding)
                    )
                }
            }
        }
    }
}
```

Do not worry if Android Studio underlines CityListScreen in red. This is expected because we have not yet created that composable function yet.

8. Create a CityListScreen Composable

   In the previous step, MainActivity passed cityRepository.cities into CityListScreen. Now we need to create that function so Compose knows how to display the list. In MainActivity.kt after class MainActivity, type the following code:

```
// @Composable means this function describes part of the app's UI
@Composable  2 Usages
fun CityListScreen(
    // cities: List<String> is the list of city names that
    // this screen receives from MainActivity
    cities: List<String>,
    // modifier: Modifier = Modifier allows layout information,
    // such as padding, to be passed into this screen
    modifier: Modifier = Modifier
) {
    // LazyColumn is the Compose for a basic scrolling ListView
    LazyColumn(modifier = modifier.fillMaxSize()) {
        // items(cities) loops through the city list and
        // creates one UI row for each city.
        items(cities) { city ->
            CityRow(city = city)
        }
    }
}
```

9. Create the CityRow Composable
   Now, we will create a separate composable function for displaying one city in the list. Right now, CityListScreen is responsible for displaying the full list of cities. However, each individual city row should also have its own reusable UI function.

The CityRow composable will receive one city name as a String and display it using a Text composable.

We will also use a Modifier to format the row:

- fillMaxWidth() makes the row take up the full width of the screen
- padding(horizontal = 18.dp, vertical = 14.dp) adds spacing around the city name
- fontSize = 28.sp makes the text easier to read

Add the following code below CityListScreen:

```
@Composable  1 Usage
fun CityRow(city: String) {
    Text(
        text = city,
        fontSize = 28.sp,
        modifier = Modifier
            .fillMaxWidth()
            .padding(horizontal = 18.dp, vertical = 14.dp)
    )
}
```

10. Add a Text Field and Add City Button

Now we will allow the user to add a new city while the app is running. To do this, CityListScreen needs three things:

I. A text field to store the city name typed by the user
II. A button that adds the city to the list
III. A callback function that sends the new city name back to the CityRepository

First, update the CityListScreen function header so it receives an onAddCity function:

```
@Composable  2 Usages
fun CityListScreen(
    cities: List<String>,
    onAddCity: (String) -> Unit,
    modifier: Modifier = Modifier
```

onAddCity: (String) -> Unit means that CityListScreen receives a function as a parameter.

The function takes on String, which will be the city that is typed by the user.

The function returns Unit, which means it performs an action but does not return a value.

Then, pass the onAddCity argument to CityListScreen inside the setContent {...} block in onCreate(), which is found in the MainActivity class:

```
setContent {
    ListyCityTheme {
        Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
            CityListScreen(
                cities = cityRepository.cities,
                onAddCity = { cityRepository.addCity(it) },
                modifier = Modifier.padding( paddingValues = innerPadding)
            )
```

It is the city name that will be sent from CityListScreen.

cityRepository.addCity(it) adds that city to the list stored in CityRepository.

Next, create a state variable inside CityListScreen to store the text typed by the user:

```
fun CityListScreen(
    cities: List<String>,
    onAddCity: (String) -> Unit,
    modifier: Modifier = Modifier
) {
    var newCityName by remember {mutableStateOf( value = "") }
```

newCityName stores the current text in the input field and remember keeps the value available while the composable is on the screen.
MutableStateOf(“”) tells Compose to update the UI when the value changes.

Next, wrap the screen content in a Column, then add a Row containing an OutlinedTextField and a Button:

```
var newCityName by remember {mutableStateOf( value = "") }

Column(modifier = modifier.fillMaxSize()) {
    Row(modifier = Modifier.padding( all = 16.dp)) {
        OutlinedTextField(
            value = newCityName,
            onValueChange = { newCityName = it },
            label = { Text("City name") },
            modifier = Modifier.weight(1f)
        )

        Spacer(modifier = Modifier.width(8.dp))

        Button(
            onClick = {
                if (newCityName.isNotBlank()) {
                    onAddCity(newCityName)
                    newCityName = ""
                }
            }
        ) {
            Text("Add City")
        }
    }

    // LazyColumn is the Compose replacement for a basic scrolling ListView
    LazyColumn(modifier = Modifier.fillMaxSize()) {
        items(cities) { city ->
            CityRow(city = city)
        }
    }
}
```