

5

Essential Libraries – Ktor, Kotlin Serialization, and Coil

Activity 5.1 – displaying the current weather

Solution

We will use the free OpenWeatherMap API for this activity. The documentation can be found at <https://openweathermap.org/api>. To sign up for an API token, please go to https://home.openweathermap.org/users/sign_up. You can find your keys and generate new ones as needed at https://home.openweathermap.org/api_keys.

The steps for this activity are as follows:

1. Create a new project in Android Studio named Weather App with a package name of `com.example.weatherapp`.
2. Grant internet permissions to the app in order to be able to make API and image requests by adding the following to `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.INTERNET" />
```

3. Add the Kotlin Serialization plugin by updating the `app/build.gradle` file:

```
plugins {  
    ...  
  
    kotlin("plugin.serialization") version "1.9.0"  
}
```

4. Add Ktor, the Ktor CIO engine, the ContentNegotiation plugin, the Kotlin Serialization Ktor plugin, Coil, and the Coil Kotlin extension dependencies to the app. These are the dependencies that you should include:

```
ktor = { group = "io.ktor", name = "ktor-client-core",
        version.ref = "ktor" }
ktor-cio = { group = "io.ktor", name = "ktor-client-cio",
            version.ref = "ktor" }
ktor-client-content-negotiation = { group = "io.ktor",
    name = "ktor-client-content-negotiation", version.ref = "ktor" }
ktor-serialization-kotlinx-json = { group = "io.ktor",
    name = "ktor-serialization-kotlinx-json", version.ref = "ktor" }
coil = { group = "io.coil-kt", name = "coil", version.ref = "coil" }
coil-compose = { group = "io.coil-kt", name = "coil-compose",
                version.ref = "coil" }
```

5. Declare a `LoadedImage` composable that wraps around a Coil composable:

```
@Composable
fun LoadedImage(imageUrl: String, modifier: Modifier = Modifier) {
    AsyncImage(
        model = imageUrl,
        contentDescription = null,
        modifier = modifier
    )
}
```

6. Update the app composable to support the presentation of the weather in a textual form (a short and long description), as well as a weather icon:

```
@Composable
fun Weather(
    shortDescription: String,
    longDescription: String,
    weatherIconUrl: String,
    modifier: Modifier = Modifier
) {
    Box(modifier = modifier) {
        Row(modifier = Modifier.padding(8.dp)) {
            Column(Modifier.weight(1f)) {
```

```
        Text(
            text = shortDescription,
            fontSize = 18.sp
        )
        Text(text = longDescription)
    }
    if (weatherIconUrl.isNotEmpty()) {
        LoadedImage(imageUrl = weatherIconUrl)
    }
}
}
```

7. Define the model:

```
@Serializable
data class WeatherData(
    @SerializedName("main") val shortDescription: String,
    @SerializedName("description") val longDescription: String,
    @SerializedName("icon") val iconId: String
)
```

8. Create classes that will contain the server response:

```
@Serializable
data class ServerResponseData(
    @SerializedName("weather") val weather: List<WeatherData>
)
```

9. Add a Ktor HttpClient object to your main activity:

```
private val client = HttpClient(CIO)
```

10. Install the JSON ContentNegotiation plugin. Remember to set it to ignore unknown keys:

```
private val client = HttpClient(CIO) {
    install(ContentNegotiation) {
        json(Json { ignoreUnknownKeys = true })
    }
}
```

11. Add a function to read the weather from the OpenWeatherMap API at <https://api.openweathermap.org/data/2.5/weather>. Remember to add the appid parameter, as well as the desired lat and lon geocoordinates. Also, remember to make the function a suspend function:

```
private suspend fun weather(): ServerResponseData = client.get(
    "https://api.openweathermap.org/data/2.5/weather"
) {
    url {
        parameter("appid", "[ENTER TOKEN HERE]")
        parameter("lat", "51.6201654")
        parameter("lon", "0.3018662")
    }
}.body()
```

Important note

The documentation for the current weather API can be found here: <https://openweathermap.org/current>.

To find the coordinates for your desired location, see here: <https://openweathermap.org/api/geocoding-api>.

The icons can be fetched using the icon value from the API response: <https://openweathermap.org/img/wn/{icon}@2x.png>.



12. Call the function you created from your main activity every time it is composed. Store the result in a variable using remember:

```
var weather: WeatherData by remember {
    mutableStateOf(
        WeatherData(
            shortDescription = "",
            longDescription = "",
            iconId = ""
        )
    )
}

LaunchedEffect(true) {
```

```

        withContext(Dispatchers.IO) {
            weather = weather().weather.first()
        }
    }
}

```

13. Handle the successful server response:

```

WeatherAppTheme {
    Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
        Card(
            modifier = Modifier
                .padding(innerPadding)
                .padding(16.dp)
                .fillMaxWidth()
        ) {
            Weather(
                shortDescription = weather.shortDescription,
                longDescription = weather.longDescription,
                weatherIconUrl = weather.iconId.let { iconId ->
                    if (iconId.isEmpty()) {
                        ""
                    } else {
                        "https://openweathermap.org/img/
wn/$iconId@2x.png"
                    }
                },
                modifier = Modifier
                    .padding(16.dp)
                    .fillMaxWidth()
            )
        }
    }
}

```

14. Handle exceptions thrown when trying to fetch the weather:

```

LaunchedEffect(true) {
    withContext(Dispatchers.IO) {
        weather = try {

```

```
val response = weather()
val weather = response.weather.firstOrNull()
if (weather == null) {
    WeatherData(
        shortDescription = "No weather returned.",
        longDescription = "",
        iconId = ""
    )
} else {
    weather
}
} catch (exception: Exception) {
    WeatherData(
        shortDescription = "Something went wrong",
        longDescription = "$exception",
        iconId = ""
    )
}
}
```

15. Run the app. The expected output is shown here:



Figure 5.6 – The final weather app