Practical Instructions

Installation instructions [before workshop]:

For this workshop, you will need:

- Weather API key (**time sensitive: it may take some time to activate**)
- Android Studio **Hedgehog** installed & configured. (Iguana is the latest stable version, but all the practicals were tested with Hedgehog).
- JDK **17** installed & configured
- Cloned repository
- Additional iOS setup (macOS only)
    - Xcode 15 installed & configured. (I have Xcode 15.2)
    - KDoctor installed and checked
- **Setup verification done**

Please see sections below for details.

Questions?

Email: pamela.hill@jetbrains.com

Time-critical setup instructions (do **ASAP**):

1. Sign up for the Weather API.
    a. Navigate to https://home.openweathermap.org/users/sign_up.
    b. Fill in the form with your username, email address (the API key will be sent here), and password. Check the appropriate checkboxes and click on “Create account”.
    c. Next you will need to fill in the “How and where will you use our API?” dialog. I used “Education/Science” as “Purpose” and click the “Save” button.
    d. Go to your email and locate the OpenWeatherMap email that was sent to you. Click on the “Verify your email” button.
    e. The next email you will receive is the “API Instruction” in which you will receive your API key. **It may take some time to activate this key.**
    f. In Practical 2 Step 2, you will need to navigate to your local.properties file in the project, and add a line where the value is the same as your received API key. WEATHER_API_KEY={your api key goes here}. The instructions will remind you again of this.

Everyone: Android Studio setup

The instructions below describe the minimum setup to be able to run Android/Desktop. Please follow these instructions for macOS, Windows or Linux.

If you have a Mac and want to make sure the iOS components are set up properly, please also follow the Additional iOS setup section below. If not, you can skip that section and run only on Android/Desktop.

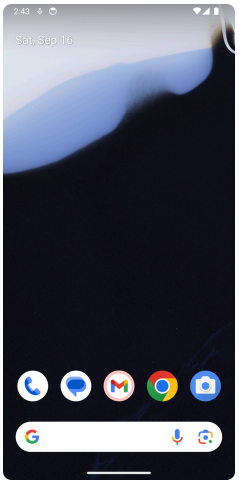
You can ignore all instructions related to Xcode and KDoctor if you're not planning on running the app on iOS.

Pre-steps/knowledge:

1. You will need to install Android Studio **Hedgehog**.
2. You will need to install the JDK **17** or higher (I used 18.0.2)**.**
3. If you're not sure whether your Mac has an Intel or Apple Silicon processor, follow the guide here.

Instructions:

1. Navigate to https://kotlinlang.org/docs/multiplatform-mobile-setup.html and follow the installation instructions.
2. In Android Studio, get the workshop repository from GitHub.
    a. Click on the "Get from VCS" button (assuming no other projects are open).
    b. In the "Get from Version Control" screen:
        i. Set "URL" to: https://github.com/pahill/kmp-workshop-appdevcon
        ii. Set "Directory" to where you want the project to be saved (and take note where you're saving to).
        iii. Click on the "Clone" button.
        iv. Note this may take some time.
4. In Android Studio, set up an emulator with API 24 or above. Assuming you have a project open:
    a. Navigate to "Tools" -> "Device Manager"
    b. In the "Device Manager" panel, click on "Create Device".
    c. In the "Select Hardware" dialog, select "Phone" for Category, and any phone that has the Play Store installed (the arrow icon ). I used Pixel 3a. Click the "Next" button.
    d. In the "System Image" panel, click on the download button next to the release name at the top of the list. The "SDK Component Installer" dialog box will appear and install the correct system image. I used API 34. This may take some time to download and install. Click the "Finish" button.
    e. Back in the "System Image" screen, click the "Next" button.
    f. In the "Android Virtual Device" screen, give your emulator a name in the "AVD Name" box, and click the "Finish" button.
    g. To verify that you have set up the emulator correctly:
        i. Go to "Tools" -> "Device Manager" again.
        ii. Click the launch button () corresponding to the emulator you created.
        iii. The device will then boot up, and you'll be taken to the home screen. Your home screen might look something like this:



Additional iOS setup

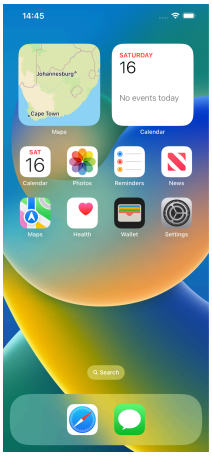
After you have installed and set up Android studio as described above, follow the instructions below to make sure iOS components are set up properly. Note: this is for macOS users only.

Pre-steps/knowledge:

1. If prompted to update your Ruby, follow the instructions here. I used RVM.
2. Select the default devices when prompted by Xcode what you want to develop for. You can add more if you'd like.

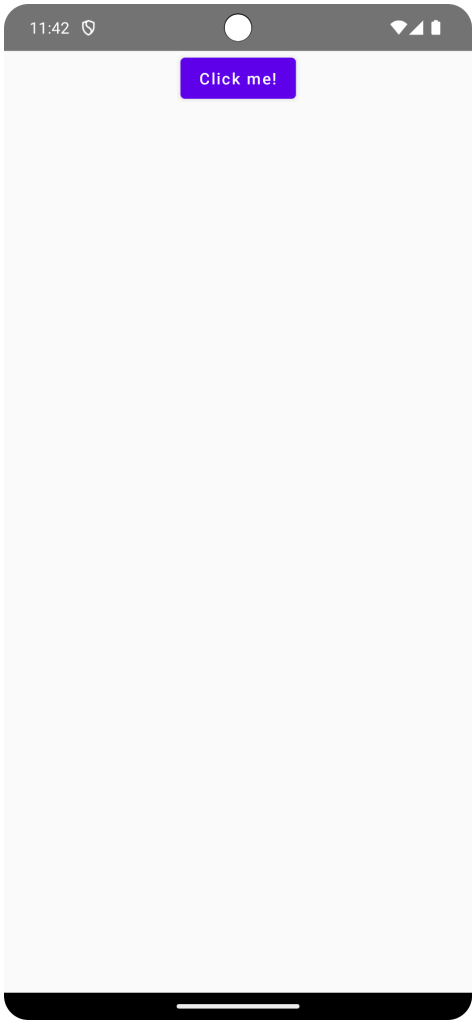
Installation steps:

1. Navigate to https://kotlinlang.org/docs/multiplatform-mobile-setup.html and follow the installation instructions, also follow the Xcode and KDoctor instructions. You may ignore KDoctor's warnings regarding the CocoaPods installation, we'll be using Direct Integration for simplicity.
2. In XCode, set up a simulator.
    a. Follow these instructions.
    b. Still in XCode, go to the menu "Xcode" -> "Open Developer Tool" -> "Simulator".
    c. In Simulator, go to the menu "File" -> "Open Simulator", then tick the simulator you'd like to launch. The simulator will boot and launch.
    d. Your home screen might look something like this:



Everyone: Workspace verification

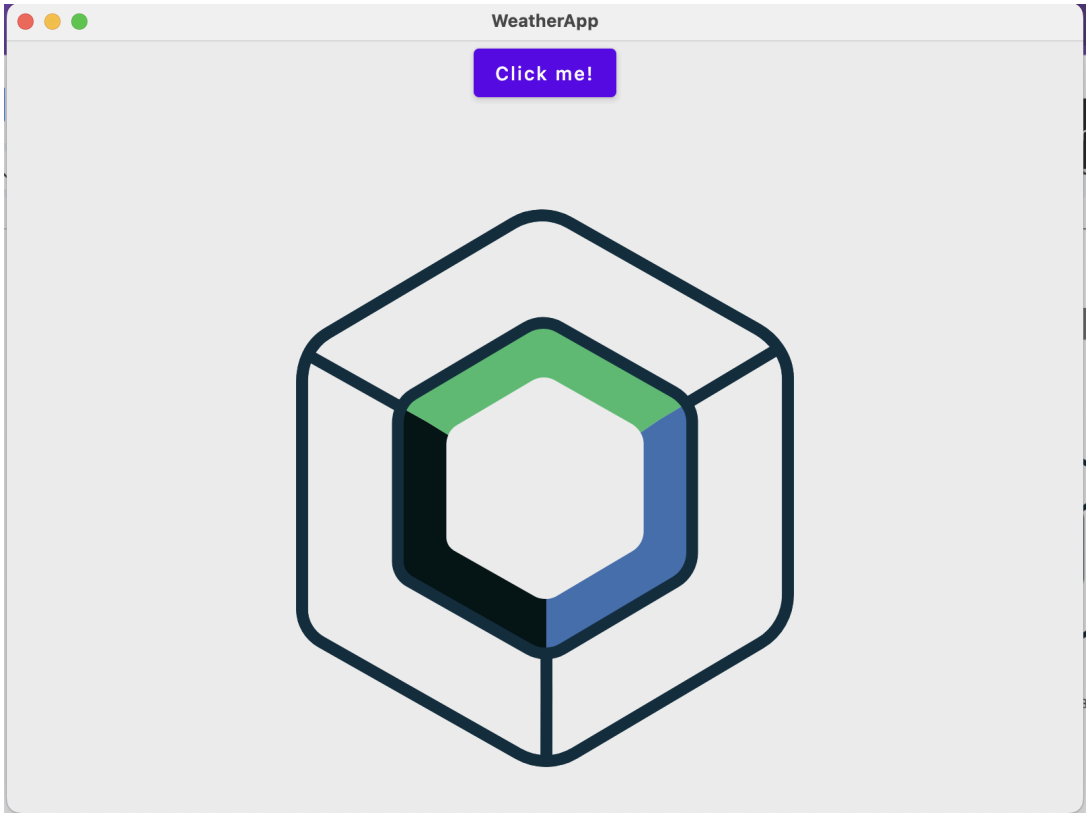
1. In Android Studio, navigate to “File” -> “Open”. In the file dialog, navigate to your local repository and then to the subdirectory /workspace-verification/final. The project should be compilable and runnable, but may need some time to finish syncing.
2. To run the project on an Android emulator, ensure that “composeApp” is selected in the “Run configurations” dropdown, that the emulator you created in the installation steps is selected, and click on the “Run composeApp” button (to the right). The app should start and the following screen should appear.



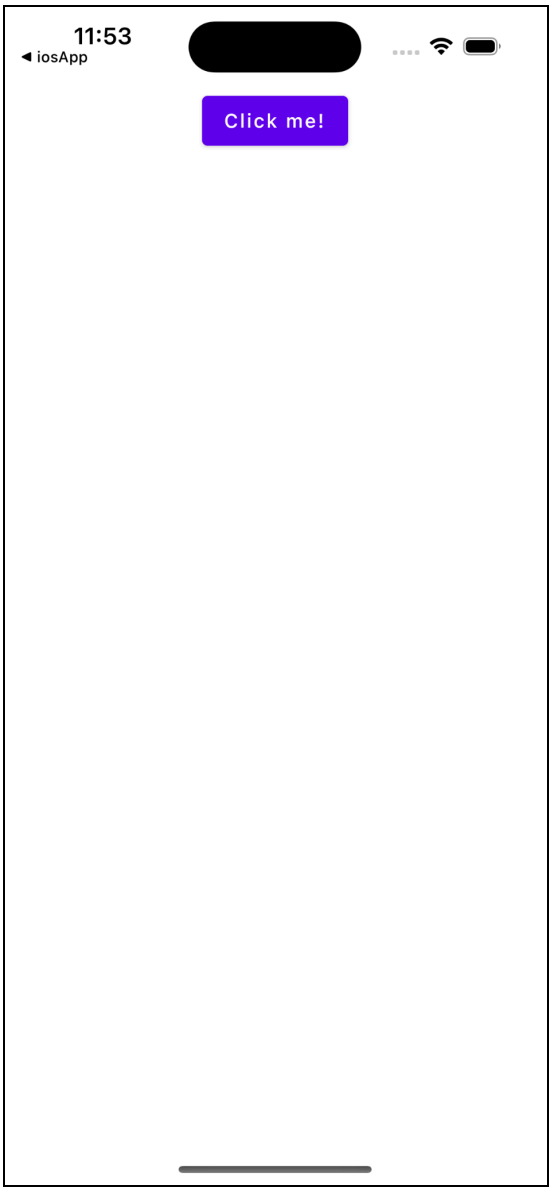
3. To run the project on Desktop, navigate to composeApp/src/desktopMain/kotlin/main.kt. In the code editor, there should be a green play gutter icon to run the application. Press the green gutter icon.

```
6  fun main() = application { this: ApplicationScope
7      Window(onCloseRequest = ::exitApplication, title = "WeatherApp") { this: FrameWindowScope
8          App()
9      }
10 }
```

The app should start and the following screen should appear as a desktop app.



4. **Close the desktop application before proceeding to the next step. There seems to be a bug in Android Studio that requires this action.**
5. To run the project on an iPhone simulator, ensure that “iosApp” is selected in the “Run configurations” dropdown, and click on the “Run ‘iosApp’” button (to the right). The app should start and the following screen should appear.

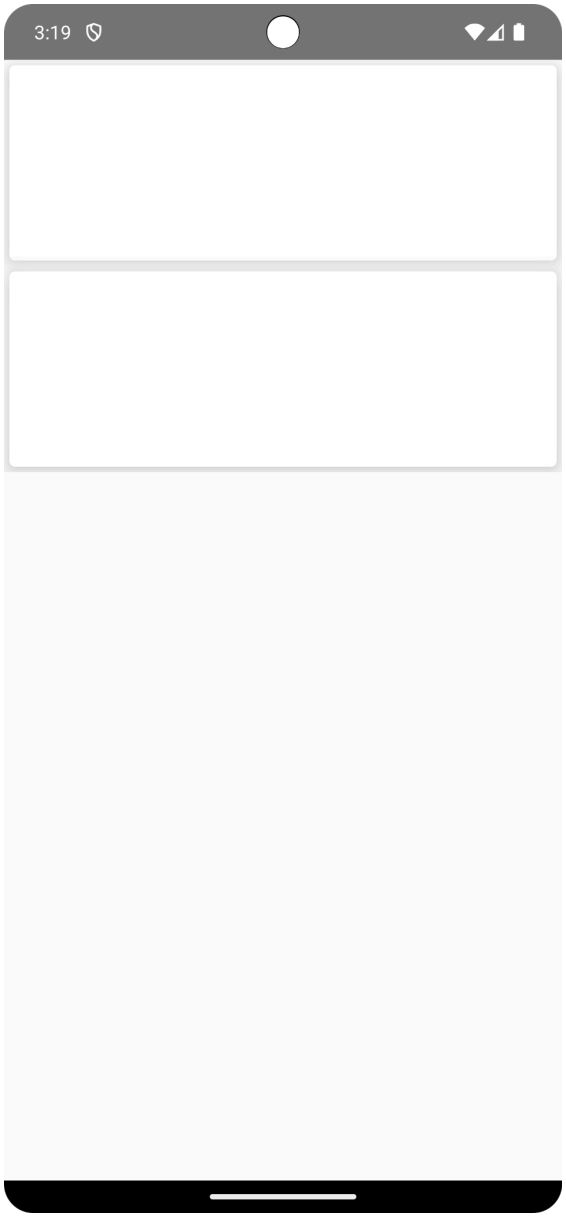


Troubleshooting:

- It might be that your iOS simulator isn't selected correctly in Android Studio.
    1. Click on the drop down menu arrow next to iosApp.
    2. Click on "Edit Configurations".
    3. Check that the "Execution target" is the right simulator.

Practical 1 [20 minutes]

1. In Android Studio, navigate to “File” -> “Open”. In the file dialog, navigate to your local repository and then to the subdirectory /practical1/starter. The project should be compilable and runnable and will look like this (two empty cards).

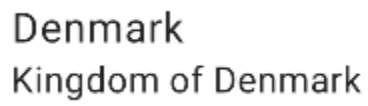


2. Navigate to the composeApp/src/commonMain/kotlin/com.jetbrains.weatherapp/App.kt file. You need to fill in the details of three Composable functions (their bodies are currently marked as TODO).

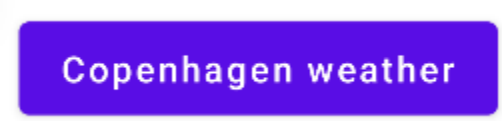
a. Flag - a Card to display the country's flag using the Image Loader library.

NB! *Make sure you use com.seiko.imageloader.rememberImagePainter for* ***network*** *images, but you can use org.jetbrains.compose.resources.painterResource for* ***local*** *images.*

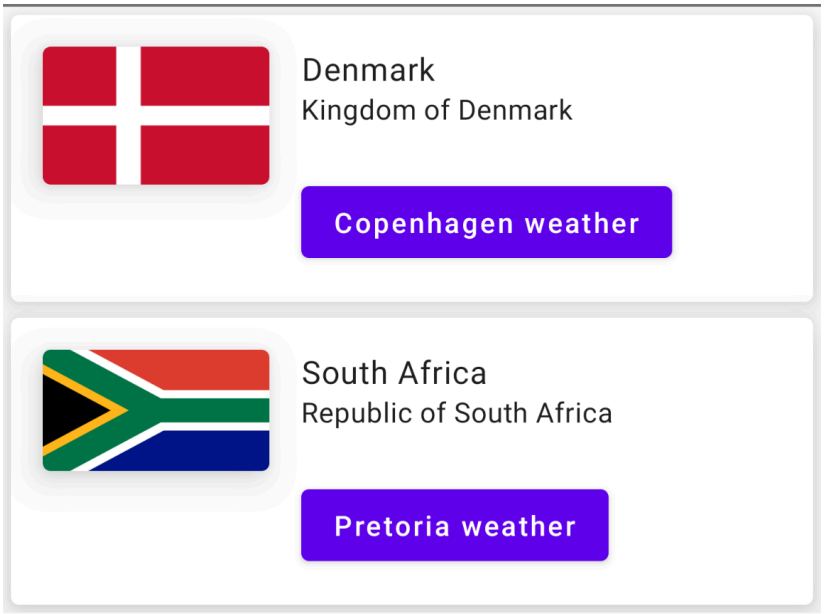
b. CountryNames - to display the country's common name and below it the official name.



c. WeatherButton - buttons to display the country's capital cities (there can be more than one).

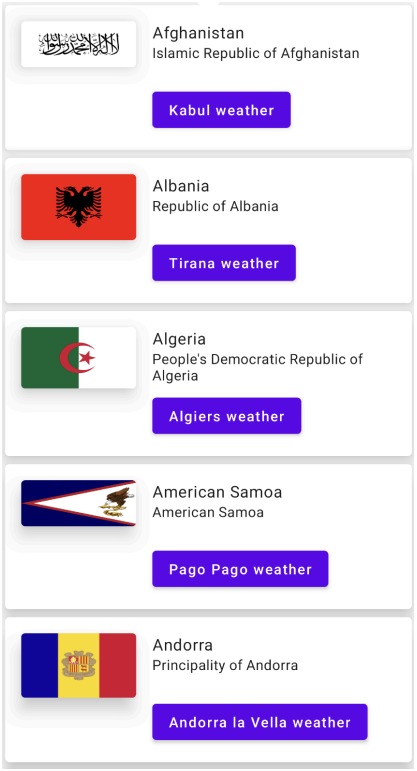


3. When you are finished, your application will look like this:



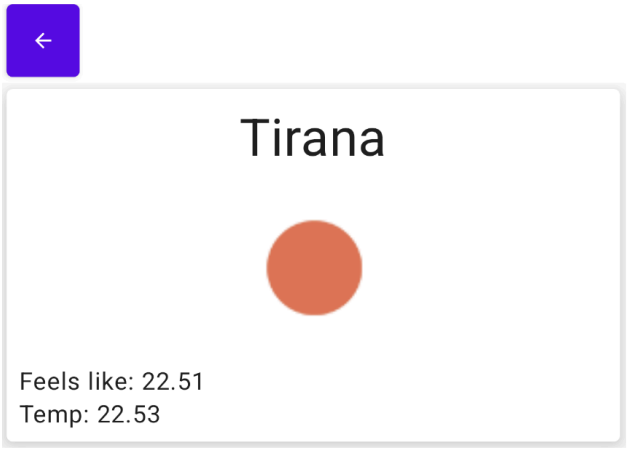
Practical 2 [20 mins]

1. Navigate to “File” -> “Open”. In the file dialog, navigate to your local repository and then to the subdirectory /practical2/starter.
2. **NB: Make sure you have configured your WEATHER_API_KEY in your project’s local.properties. It was part of the Pre-steps instructions. If “Config” is missing, try a Gradle sync.**
3. The project should be compilable and runnable and should look like this:



4. Navigate to the composeApp/src/commonMain/kotlin/com.jetbrains.weatherapp/network/WeatherApi.kt file. You need to fill in the two TODOs:
    a. Setting up the httpClient.
    b. Getting the weather data from the webservice using the http client. The URL is: https://api.openweathermap.org/data/2.5/weather?lat=**${lat}**&lon=**${long}**&appid=**${Config.WeatherApiKey}**&units=metric
    Where lat is the capital’s latitude, long is the capital’s longitude.
    c. Config.WeatherApiKey will automatically be populated for you from your local.properties. The full name is com.myapplication.Config.

    NB: If you get stuck, look at the CountryApi.kt file.
5. Navigate to the App.kt file, and find the TODO in the WeatherCard Composable. You need to call the WeatherApi class’ getWeather suspend function here. When you are finished, you should be able to click on a capital city’s weather button and see it’s current weather, for example:



Practical 3 [20 minutes]

1. Navigate to “File” -> “Open”. In the file dialog, navigate to your local repository and then to the subdirectory /practical3/starter. **The project should not compile, because there is an expect function with missing actual implementations.**
2. Navigate to composeApp/src/commonMain/kotlin/com.jetbrains.weatherapp/location/CountryCodeService.kt. Notice that you need to provide an implementation for the getCountryCode function and CountryCodeService on each platform.
3. Navigate to the composeApp/src/androidMain/kotlin/com.jetbrains.weatherapp directory and create a new location directory and a file called AndroidCountryCodeService.kt, and provide the implementations there. To get the country code on Android, you can use:

   ```
   import java.util.Locale
   Locale.getDefault().country
   ```

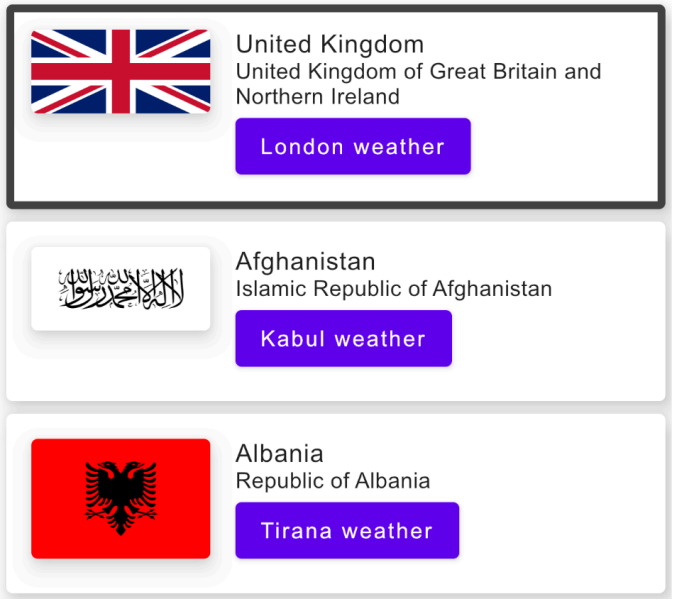
   NB: It’s very important that the `actual` function is in the same package as the `expect` function.
4. Navigate to the composeApp/src/desktopMain/kotlin/com.jetbrains.weatherapp directory and create a new location directory and a file called DesktopCountryCodeService.kt, and provide the implementations there. To get the country code on Desktop, you can use:

   ```
   import java.util.Locale
   Locale.getDefault().country
   ```
5. Navigate to the composeApp/src/iosMain/kotlin/com.jetbrains.weatherapp directory and create a new location directory and a file called iosCountryCodeService.kt, and provide the implementations there. To get the country code on iOS, you can use

   ```
   import platform.Foundation.*
   NSLocale.currentLocale().objectForKey(NSLocaleCountryCode).toString()
   ```
6. By default the locale for Android will be the US, so that card will be displayed at the top of the countries list. To change this:
   a. Open “Settings” and search for “System Languages”.

b. You may need to first add another language, let's say UK English.
c. Drag UK English above any other languages in the list.
d. Close and restart our app. The United Kingdom will then appear at the top of the countries list.

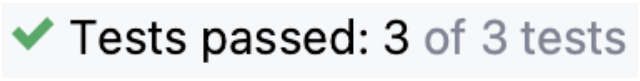
7. By default the locale for iOS will also be the US. To change this:
    a. Open "Settings" -> "General" -> "Language & Region".
    b. Click on "Region" and select a new region, let's say United Kingdom.
    c. Click on the "Change to United Kingdom" button. The phone will restart.
    d. Open the app. The United Kingdom will then appear at the top of the countries list. This is an example of what it could look like.



Practical 4 [20 minutes]

1. Navigate to "File" -> "Open". In the file dialog, navigate to your local repository and then to the subdirectory /practical4/starter. The project should compile and run.
2. Navigate to composeApp/src/commonMain/kotlin/com.jetbrains.weatherapp/weather/CelsiusToFahrenheit.kt. You will be writing three tests for the function celsiusToFahrenheit.
3. Navigate to composeApp/build.gradle.kts and uncomment the testing dependencies in the commonTest and androidUnitTest/desktopTest sections. Remember to sync Gradle.
4. Navigate to composeApp/src/commonTest/kotlin/com.jetbrains.weatherapp/weather/CelsiusToFahrenheitTest.kt. You need to write three functions to test the function mentioned in step 2.
    a. Test 0.0'C (it should be 32'F)
    b. Test 100.0'C (it should be 212'F)
    c. Test -100.0'C (it should be -148'F)

5. Run the tests by clicking on the green gutter icon next to test class, and select a platform to run the tests on. You should see something like this in the Run window:



Practical 5 [20 minutes]

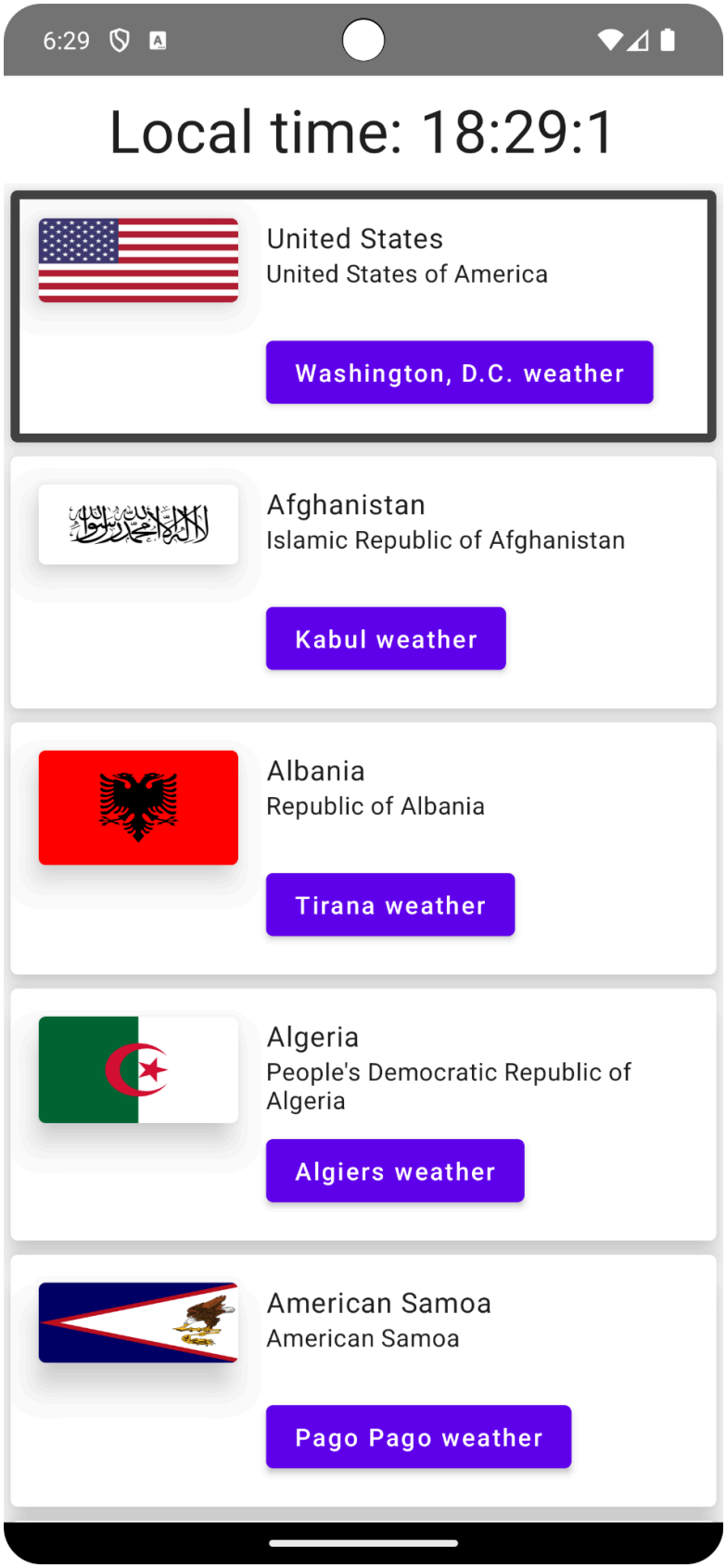
1. Navigate to “File” -> “Open”. In the file dialog, navigate to your local repository and then to the subdirectory /practical5/starter. **The project should not compile/run.**
2. There is no path to the cache configured for iOS (Android/Desktop are already configured). Navigate to composeApp/src/iosMain/kotlin/com.jetbrains.weatherapp and create a directory called cache. Now create a file called CountrySDK.kt and implement the actual function providing the path to the cache.

   Hint: You should use something like `${NSHomeDirectory()}/country_cache.json`
3. Navigate to composeApp/src/commonMain/kotlin/com.jetbrains.weatherapp/cache/CountrySDK.kt. You need to replace the TODOs:
   a. The cache variable should be assigned to a KStore store with a particular path.
   b. The getAllCountries function should implement normal caching logic working with the cache and api variables.
4. Run the app on both Android and iOS. After the first load, the countries list should display much faster as it is cached.

Practical 6 [20 minutes]

1. Navigate to “File” -> “Open”. In the file dialog, navigate to your local repository and then to the subdirectory /practical6/starter. **The project should not compile.**
2. Navigate to composeApp/build.gradle.kts and uncomment the kotlinx-datetime dependency in the commonMain section. Remember to sync Gradle.
3. Navigate to /composeApp/src/commonMain/kotlin/com.jetbrains.weatherapp/time/TimeApi.kt. You need to replace the TODO:
   a. The clockTime variable needs to be a Flow of Strings of the current time, with a delay in between each emission of the time.
4. Navigate to App.kt. You need to replace the TODOs:
   a. Collect the flow and assign it to a variable.

      Hint: Remember to `remember`.
   b. Display a Text Composable with the time.
5. The app should display the updating time, for example:


6:29
Local time: 18:29:1
United States
United States of America
Washington, D.C. weather
Afghanistan
Islamic Republic of Afghanistan
Kabul weather
Albania
Republic of Albania
Tirana weather
Algeria
People's Democratic Republic of Algeria
Algiers weather
American Samoa
American Samoa
Pago Pago weather