# Fotocamera, GPS, HTTP

Permessi, sensori e connettività di rete su Android

# Laboratorio di oggi

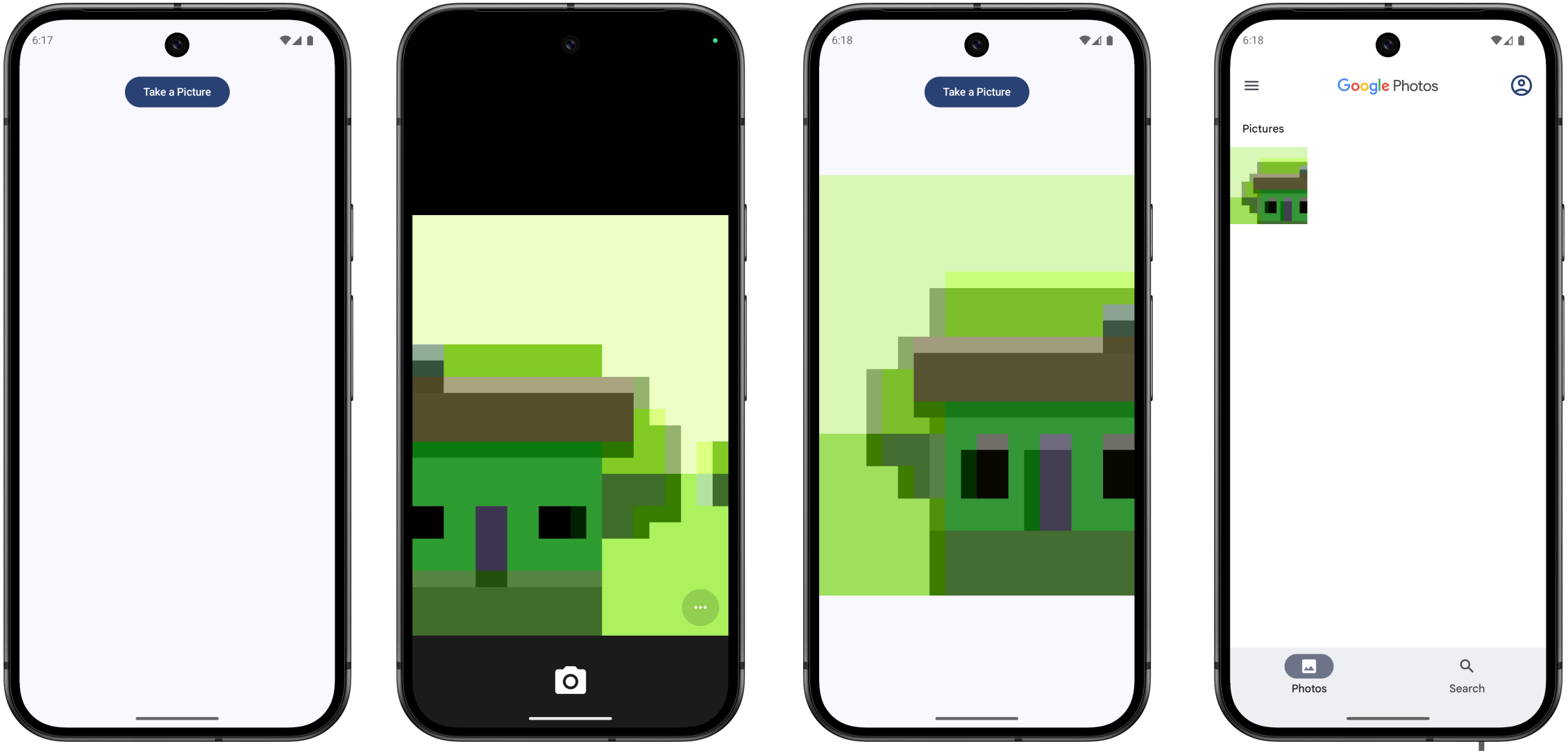
1. Fotocamera (esercitazione guidata)
   - Scattare foto e salvarle nella memoria del dispositivo
   - Progetto di partenza su Virtuale: **CameraBase.zip**
2. GPS (esercitazione guidata)
   - Ottenere la posizione del dispositivo
   - Progetto di partenza: nuovo progetto su Android Studio
3. HTTP (esercitazione guidata)
   - Inviare richieste HTTP a servizi esterni
   - Progetto di partenza su Virtuale: **HTTPBase.zip**
4. TravelDiary (esercitazione libera)
   - Aggiunta delle funzionalità delle esercitazioni guidate

# 1. Fotocamera

# 1. Fotocamera

- Creare un’applicazione che:
  - Permetta di scattare una foto
  - La visualizzi nella UI
  - La salvi nello storage del dispositivo

# 1. Fotocamera



# 1. Fotocamera

- Punto di partenza: progetto Android con:
  - Parte dell'interfaccia utente già realizzata
  - Alcune funzioni di utility per la gestione di immagini (vedi **utils/Image.kt**)
- Passaggi:
  1. Installazione dipendenze
  2. Aggiornamento manifest
  3. Creazione path provider
  4. Scatto della foto
  5. UI per la foto scattata

# 1.1. Installazione dipendenze

- Utilizzeremo il componente **`AsyncImage`** per mostrare un'immagine a partire da un URI
- Per farlo, è necessario aggiungere la seguente dipendenza al file **build.gradle.kts (modulo :app)**

```
implementation("io.coil-kt:coil-compose:2.3.0")
```

Replace with new library catalog declaration

# 1.2. Aggiornamento manifest

- Dobbiamo aggiornare il manifest per:
  - Salvare immagini nello storage del dispositivo
  - Salvare immagini nella cache dell'app
- E il permesso per utilizzare la fotocamera?
  - Non è necessario dal momento che accederemo ad essa tramite un Intent. Sarà l'app di sistema della fotocamera ad utilizzare il sensore

# 1.2. Aggiornamento manifest

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
    <uses-permission
        android:name="android.permission.READ_EXTERNAL_STORAGE"
        android:maxSdkVersion="32" />
    <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE"
        android:maxSdkVersion="28" />

    <application ...>
        <!-- ... -->

        <provider
            android:name="androidx.core.content.FileProvider"
            android:authorities="${applicationId}.provider"
            android:exported="false"
            android:grantUriPermissions="true">
            <meta-data
                android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/path_provider" />
        </provider>
    </application>
</manifest>
```

# 1.2. Aggiornamento manifest

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
    <uses-permission
        android:name="android.permission.READ_EXTERNAL_STORAGE"
        android:maxSdkVersion="32" />
    <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE"
        android:maxSdkVersion="28" />

    <application ...>
        <!-- ... -->

        <provider
            android:name="androidx.core.content.FileProvider"
            android:authorities="${applicationId}.provider"
            android:exported="false"
            android:grantUriPermissions="true">
            <meta-data
                android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/path_provider" />
        </provider>
    </application>
</manifest>
```

Accesso allo storage per dispositivi con Android <= 9

Configurazione del path provider per salvare file in cache

# 1.3. Creazione path provider

- Dobbiamo creare il path provider a cui abbiamo fatto riferimento nel manifest

res/xml/path_provider.xml

```
<?xml version="1.0" encoding="utf-8"?>
<paths>
    <external-cache-path name="my_images" path="/"/>
</paths>
```

# 1.4. Scatto della foto

- In **`utils/Camera.kt`**, creiamo un helper per l'utilizzo della fotocamera
- L'helper consiste in una funzione **`rememberCameraLauncher()`**, che restituisce un oggetto **`CameraLauncher`** con una funzione per scattare la foto e una variabile per memorizzarne l'URI

```
interface CameraLauncher {
    val capturedImageUri: Uri
    fun captureImage()
}

@Composable
fun rememberCameraLauncher(
    onPictureTaken: (imageUri: Uri) -> Unit = {}
): CameraLauncher {
    // ...
}
```

# 1.4. Scatto della foto

1. Creiamo due URI per le immagini

```
var imageUri by remember { mutableStateOf(Uri.EMPTY) }
var capturedImageUri by remember { mutableStateOf(Uri.EMPTY) }
```

# 1.4. Scatto della foto

1. Creiamo l’immagine in cache
2. Creiamo il launcher per l’activity della fotocamera

```
val cameraActivityLauncher = rememberLauncherForActivityResult(
    ActivityResultContracts.TakePicture()
) { pictureTaken ->
    if (!pictureTaken) return@rememberLauncherForActivityResult
    capturedImageUri = imageUri
    onPictureTaken(capturedImageUri)
}
```

# 1.4. Scatto della foto

1. Creiamo due URI per le immagini
2. Creiamo il launcher per l'activity della fotocamera
3. Creiamo l'oggetto **`CameraLauncher`** con **`remember`**, in modo che non venga re-istanziato ad ogni recomposition, ma solo al cambiare del valore di **`cameraActivityLauncher`**

```
val cameraLauncher = remember(cameraActivityLauncher) {
    object : CameraLauncher {
        override val capturedImageUri get() = capturedImageUri
        override fun captureImage() {
            val imageFile = File.createTempFile("tmp_image", ".jpg", ctx.externalCacheDir)
            imageUri = FileProvider.getUriForFile(ctx, ctx.packageName + ".provider", imageFile)
            cameraActivityLauncher.launch((imageUri))
        }
    }
}
```

# 1.4. Scatto della foto

utils/camera.kt

```
@Composable
fun rememberCameraLauncher(onPictureTaken: (imageUri: Uri) -> Unit = {}): CameraLauncher {
    val ctx = LocalContext.current

    var imageUri by remember { mutableStateOf(Uri.EMPTY) }
    var capturedImageUri by remember { mutableStateOf(Uri.EMPTY) }

    val cameraActivityLauncher =
        rememberLauncherForActivityResult(ActivityResultContracts.TakePicture()) { pictureTaken ->
            if (!pictureTaken) return@rememberLauncherForActivityResult
            capturedImageUri = imageUri
            onPictureTaken(capturedImageUri)
        }

    val cameraLauncher = remember(cameraActivityLauncher) {
        object : CameraLauncher {
            override val capturedImageUri = capturedImageUri
            override fun captureImage() {
                val imageFile = File.createTempFile("tmp_image", ".jpg", ctx.externalCacheDir)
                imageUri = FileProvider.getUriForFile(ctx, ctx.packageName + ".provider", imageFile)
                cameraActivityLauncher.launch((imageUri))
            }
        }
    }
    return cameraLauncher
}
```

Risultato

# 1.4. Scatto della foto

- Nella MainActivity, utilizziamo l'helper appena creato per scattare la foto e salvarla nello storage:

```
val cameraLauncher = rememberCameraLauncher(
    onPictureTaken = { imageUri -> saveImageToStorage(imageUri, ctx.contentResolver)
})
// ...
Button(onClick = cameraLauncher::captureImage) {
    Text("Take a Picture")
}
```

# 1.5. UI per la foto scattata

- Utilizziamo **`AsyncImage`** per caricare l'immagine nell'interfaccia a partire dal suo URI

```
if (capturedImageUri.path?.isNotEmpty() == true) {
    AsyncImage(
        ImageRequest.Builder(ctx)
            .data(capturedImageUri)
            .crossfade(true)
            .build(),
        "Captured image"
    )
}
```

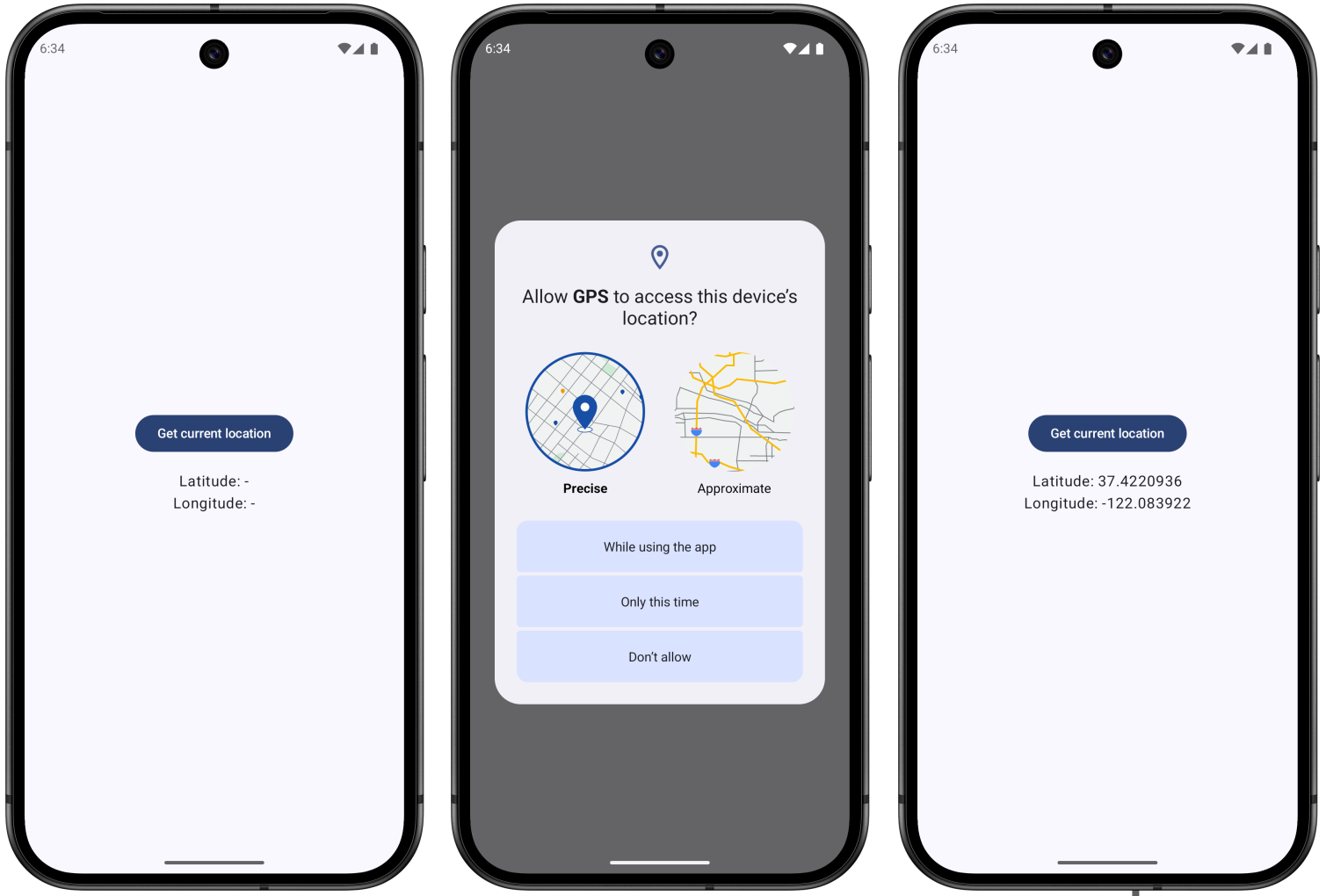
# Nota finale: salvataggio file con API ≤ 28

- Per API version ≤ 28 (Android ≤ 9), è necessario richiedere all'utente un permesso per il salvataggio di file sullo storage del dispositivo
- Il permesso è **`android.permission.WRITE_EXTERNAL_STORAGE`**, ed è ottenibile tramite l'helper che andremo a creare nel prossimo esercizio

# 2. GPS

# 2. GPS

- Creare un’applicazione che richieda l’accesso alla posizione del dispositivo e ne mostri le coordinate su schermo



# 2. GPS

- Punto di partenza: nuovo progetto Android
- Passaggi:
  1. Installazione dipendenze
  2. Aggiornamento manifest
  3. Helper per la richiesta dei permessi
  4. Classe per la gestione della posizione
  5. Reperimento della posizione nell'interfaccia utente
  6. Gestione casi particolari: permessi negati, GPS disattivato

# 2.1. Installazione dipendenze

- Per l'accesso alla posizione utilizzeremo le seguenti dipendenze (file **build.gradle.kts (modulo :app)**):

```
implementation("androidx.lifecycle:lifecycle-runtime-compose:2.8.7")
implementation("com.google.android.gms:play-services-location:21.3.0")
implementation("org.jetbrains.kotlinx:kotlinx-coroutines-play-services:1.7.1")
```

Replace with new library catalog declaration

# 2.2. Aggiornamento manifest

Dobbiamo aggiungere tre dichiarazioni al manifest:

1. Permesso per l'accesso alla posizione
   - Necessario per utilizzare il GPS
   - Dobbiamo richiedere obbligatoriamente **ACCESS_COARSE_LOCATION** per un'approssimazione di $3km^2$

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

   - Possiamo opzionalmente richiedere **ACCESS_FINE_LOCATION** per un'approssimazione di $50m^2$ (consigliato)

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

# 2.2. Aggiornamento manifest

Dobbiamo aggiungere tre dichiarazioni al manifest:

1. Permesso per l'accesso alla posizione
2. Intent per aprire le impostazioni del GPS
   – Se l'utente ha la posizione disattivata, vogliamo mostragli un tasto per aprire impostazioni del GPS, dove può abilitarla

```
<intent>
    <action android:name="android.settings.LOCATION_SOURCE_SETTINGS" />
</intent>
```

# 2.2. Aggiornamento manifest

Dobbiamo aggiungere tre dichiarazioni al manifest:

1. Permesso per l'accesso alla posizione
2. Intent per aprire le impostazioni del GPS
3. Intent per aprire le impostazioni di sistema alla pagina della nostra app
   - Se l'utente rifiuta il permesso per due volte non possiamo più richiederglielo. Possiamo però mostrargli un tasto per aprire le impostazioni della nostra app, dove può concedere il permesso

```
<intent>
    <action android:name="android.settings.APPLICATION_DETAILS_SETTINGS" />
</intent>
```

# 2.2. Aggiornamento manifest

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >

    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <queries>
        <intent>
            <action android:name="android.settings.LOCATION_SOURCE_SETTINGS" />
        </intent>
        <intent>
            <action android:name="android.settings.APPLICATION_DETAILS_SETTINGS" />
        </intent>
    </queries>

    <!-- ... -->

</manifest>
```

# 2.3. Gestione dei permessi

- In **`utils/Permission.kt`**, creiamo un helper per la gestione dei permessi
- Può essere utilizzato per richiedere qualsiasi tipo di permesso, non solo per quelli legati alla posizione
- Perché l'helper?
  - Purtroppo, all'avvio dell'app, Android non distingue tra un permesso non ancora richiesto e uno permanentemente negato 😕
  - È necessario richiedere il permesso almeno una volta per ottenere tale distinzione
  - Tramite il nostro helper ci "costringeremo" a richiederlo almeno una volta e terremo meglio traccia del suo stato

# 2.3. Gestione dei permessi

- Creiamo un enum per i possibili stati di un permesso:
  - **`Unknown`**: finché non l'abbiamo richiesto almeno una volta
  - **`Granted`**: se è stato concesso
  - **`Denied`**: se è stato negato ma possiamo ancora richiederlo
  - **`PermanentlyDenied`**: se è stato permanentemente negato e può essere modificato solo dalle impostazioni

utils/permission.kt

```
enum class PermissionStatus {
    Unknown,
    Granted,
    Denied,
    PermanentlyDenied;

    val isGranted get() = this == Granted
    val isDenied get() =
        this == Denied ||
        this == PermanentlyDenied
}
```

# 2.3. Gestione dei permessi

- Creiamo un'interfaccia **`MultiplePermissionHandler`** che contiene:
  - Una mappa da **`String`** a **`PermissionStatus`**
    - La chiave è il permesso richiesto e il valore è il corrispondente stato
  - Una funzione per richiedere i permessi

utils/permission.kt

```
interface MultiplePermissionHandler {
    val statuses: Map<String, PermissionStatus>
    fun launchPermissionRequest()
}
```

# 2.3. Gestione dei permessi

- Creiamo anche un composable **`rememberMultplePermissions`**
  - Prende come parametro la lista di permessi da richiedere e una funzione da eseguire al cambiamento di stato di tali permessi
  - Restituisce un'istanza di **`MultiplePermissionHandler`**
  - Non genera interfaccia utente → nome con l'iniziale minuscola

utils/permission.kt

```
@Composable
fun rememberMultiplePermissions(
    permissions: List<String>,
    onResult: (status: Map<String, PermissionStatus>) -> Unit
): MultiplePermissionHandler {
    // ...
}
```

# 2.3. Gestione dei permessi

- All'interno del composable abbiamo bisogno di:
  - Un riferimento all'activity attuale

```
val activity = LocalActivity.current!!
```

# 2.3. Gestione dei permessi

- All'interno del composable abbiamo bisogno di:
  - Un riferimento all'activity attuale
  - Un **`MutableState`** contenente la mappa **`String -> PermissionStatus`**

```
var statuses by remember {
    mutableStateOf(
        permissions.associateWith { permission ->
            if (ContextCompat.checkSelfPermission(
                activity,
                permission
            ) == PackageManager.PERMISSION_GRANTED)
                PermissionStatus.Granted
            else
                PermissionStatus.Unknown
        }
    )
}
```

# 2.3. Gestione dei permessi

- All'interno del composable abbiamo bisogno di:
  - Un riferimento all'activity attuale
  - Un **`MutableState`** contenente la mappa **`String -> PermissionStatus`**

```
var statuses by remember {
    mutableStateOf(
        permissions.associateWith { permission ->
            if (ContextCompat.checkSelfPermission(
                activity,
                permission
            ) == PackageManager.PERMISSION_GRANTED)
                PermissionStatus.Granted
            else
                PermissionStatus.Unknown
        }
    )
}
```

Alla prima composition creiamo, a partire dalla lista di permessi da chiedere, una mappa contenente lo stato di ciascuno. Per i limiti discussi in precedenza, a questo punto è possibile solo distinguere tra Granted e Unknown.

# 2.3. Gestione dei permessi

- All'interno del composable abbiamo bisogno di:
  - Un riferimento all'activity attuale
  - Un **`MutableState`** contenente la mappa **`String -> PermissionStatus`**
  - Un launcher per avviare la richiesta dei permessi

```
val permissionLauncher = rememberLauncherForActivityResult(
    ActivityResultContracts.RequestMultiplePermissions()
) { newPermissions ->
    statuses = newPermissions.mapValues { (permission, isGranted) ->
        when {
            isGranted -> PermissionStatus.Granted
            activity.shouldShowRequestPermissionRationale(permission) -> PermissionStatus.Denied
            else -> PermissionStatus.PermanentlyDenied
        }
    }
    onResult(statuses)
}
```

# 2.3. Gestione dei permessi

- All'interno del composable abbiamo bisogno di:
  - Un riferimento all'activity attuale
  - Un **`MutableState`** contenente la mappa **`String -> PermissionStatus`**
  - Un launcher per avviare la richiesta dei permessi

```
val permissionLauncher = rememberLauncherForActivityResult(
    ActivityResultContracts.RequestMultiplePermissions()
) { newPermissions ->
    statuses = newPermissions.mapValues { (permission, isGranted) ->
        when {
            isGranted -> PermissionStatus.Granted
            activity.shouldShowRequestPermissionRationale(permission) -> PermissionStatus.Denied
            else -> PermissionStatus.PermanentlyDenied
        }
    }
    onResult(statuses)
}
```

Utilizziamo shouldShowRequestPermissionRationale per distinguere tra negazione temporanea e permenanete

# 2.3. Gestione dei permessi

- All'interno del composable abbiamo bisogno di:
  - Un riferimento all'activity attuale
  - Un **`MutableState`** contenente la mappa **`String -> PermissionStatus`**
  - Un launcher per avviare la richiesta dei permessi

```
val permissionLauncher = rememberLauncherForActivityResult(
    ActivityResultContracts.RequestMultiplePermissions()
) { newPermissions ->
    statuses = newPermissions.mapValues { (permission, isGranted) ->
        when {
            isGranted -> PermissionStatus.Granted
            activity.shouldShowRequestPermissionRationale(permission) -> PermissionStatus.Denied
            else -> PermissionStatus.PermanentlyDenied
        }
    }
    onResult(statuses)
}
```

Esecuzione della callback passata come parametro dopo ogni richiesta di permesso

# 2.3. Gestione dei permessi

- All'interno del composable abbiamo bisogno di:
  - Un riferimento all'activity attuale
  - Un **`MutableState`** contenente la mappa **`String -> PermissionStatus`**
  - Un launcher per avviare la richiesta dei permessi
  - Un **`derivedState`** che utilizza gli elementi precedenti per creare un'istanza di **`MultiplePermissionHandler`**

```
val permissionHandler = remember(permissionLauncher) {
    object : MultiplePermissionHandler {
        override val statuses get() = statuses
        override fun launchPermissionRequest() =
            permissionLauncher.launch(permissions.toTypedArray())
    }
}
return permissionHandler
```

# 2.3. Gestione dei permessi

- All'interno del composable abbiamo bisogno di:
  - Un riferimento all'activity attuale
  - Un **`MutableState`** contenente la mappa **`String -> PermissionStatus`**
  - Un launcher per avviare la richiesta dei permessi
  - Un **`derivedState`** che utilizza gli elementi precedenti per creare un'istanza di **`MultiplePermissionHandler`**

```
val permissionHandler = remember(permissionLauncher) {
    object : MultiplePermissionHandler {
        override val statuses get() = statuses
        override fun launchPermissionRequest() =
            permissionLauncher.launch(permissions.toTypedArray())
    }
}
return permissionHandler
```

Con remember, MultiplePermissionHandler viene ricreato solo se le sue dipendenze cambiano (permissionLauncher)

# 2.3. Gestione dei permessi

```
activity.shouldShowRequestPermissionRationale(permission)
```

- **Nota**: **`shouldShowRequestPermissionRationale`** restituisce **`true`** se l'utente ha *temporaneamente* negato il permesso
- Questo significa che, alla prossima richiesta di permesso, dovremmo fornire una breve spiegazione, o *rationale*, sul perché l'app ha bisogno del permesso
- Lo vedremo meglio nell'ultimo step dell'esercizio

# 2.3. Gestione dei permessi

- Esempio di utilizzo dell'helper:

```
val locationPermissions = rememberMultiplePermissions(
    listOf(Manifest.permission.ACCESS_COARSE_LOCATION, Manifest.permission.ACCESS_FINE_LOCATION)
) { statuses ->
    when {
        statuses.any { it.value == PermissionStatus.Granted } ->
            // Do something with the GPS
        else ->
            // User has not granted location access
    }
}
```

  - In questo caso, ci basta che uno dei permessi sia **`Granted`** per procedere con l'uso del GPS
  - In altre situazioni, è possibile richiedere permessi totalmente scollegati tra loro (ad esempio **`CAMERA`** e **`READ_CONTACTS`**) e gestire ciascuno individualmente tramite **`statuses.get(PERMISSION_CODE)`**.

# 2.4. Classe per la gestione della posizione

- In **`utils/LocationService.kt`**, creiamo una classe **`LocationService`** per il monitoraggio della posizione
- La classe deve consentire le seguenti operazioni:
  - Reperimento della posizione del dispositivo
  - Apertura delle impostazioni del GPS

# 2.4. Classe per la gestione della posizione

- Creiamo una data class per memorizzare una coppia di coordinate GPS

```
data class Coordinates(val latitude: Double, val longitude: Double)
```

# 2.4. Classe per la gestione della posizione

- Accettiamo un **`Context`** come parametro del costruttore

```
class LocationService(private val ctx: Context) {
}
```

# 2.4. Classe per la gestione della posizione

- Accettiamo un **`Context`** come parametro del costruttore
- Reperiamo un'istanza di **`FusedLocationProviderClient`**

```
class LocationService(private val ctx: Context) {
    private val fusedLocationClient =
        getFusedLocationProviderClient(ctx)
}
```

# 2.4. Classe per la gestione della posizione

- Accettiamo un **`Context`** come parametro del costruttore
- Reperiamo un'istanza di **`FusedLocationProviderClient`**
- Reperiamo un'istanza di **`LocationManager`**

```
class LocationService(private val ctx: Context) {
    private val fusedLocationClient =
        getFusedLocationProviderClient(ctx)
    private val locationManager = ctx
        .getSystemService(Context.LOCATION_SERVICE)
        as LocationManager
}
```

# 2.4. Classe per la gestione della posizione

- Accettiamo un **`Context`** come parametro del costruttore
- Reperiamo un'istanza di **`FusedLocationProviderClient`**
- Reperiamo un'istanza di **`LocationManager`**
- Creiamo uno **`StateFlow`** per memorizzare le coordinate

```
class LocationService(private val ctx: Context) {
    private val fusedLocationClient =
        getFusedLocationProviderClient(ctx)
    private val locationManager = ctx
        .getSystemService(Context.LOCATION_SERVICE)
        as LocationManager

    private val _coordinates =
        MutableStateFlow<Coordinates?>(null)
    val coordinates = _coordinates.asStateFlow()
}
```

# 2.4. Classe per la gestione della posizione

- Accettiamo un **`Context`** come parametro del costruttore
- Reperiamo un'istanza di **`FusedLocationProviderClient`**
- Reperiamo un'istanza di **`LocationManager`**
- Creiamo uno **`StateFlow`** per memorizzare le coordinate
- Creiamo uno **`StateFlow`** che indica se stiamo caricando la posizione

```
class LocationService(private val ctx: Context) {
    private val fusedLocationClient =
        getFusedLocationProviderClient(ctx)
    private val locationManager = ctx
        .getSystemService(Context.LOCATION_SERVICE)
        as LocationManager

    private val _coordinates =
        MutableStateFlow<Coordinates?>(null)
    val coordinates = _coordinates.asStateFlow()

    private val _isLoadingLocation =
        MutableStateFlow(false)
    val isLoadingLocation = _isLoadingLocation.asStateFlow()
}
```

# 2.4. Classe per la gestione della posizione

```
class LocationService(private val ctx: Context) {
    // ...

    suspend fun getCurrentLocation(usePreciseLocation: Boolean = false): Coordinates? {
        val locationEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)
        if (!locationEnabled) throw IllegalStateException("Location is disabled")

        val permissionGranted = ContextCompat.checkSelfPermission(
            ctx,
            Manifest.permission.ACCESS_COARSE_LOCATION
        ) == PackageManager.PERMISSION_GRANTED
        if (!permissionGranted) throw SecurityException("Location permission not granted")

        _isLoadingLocation.value = true
        val location = withContext(Dispatchers.IO) {
            fusedLocationClient.getCurrentLocation(
                if (usePreciseLocation) Priority.PRIORITY_HIGH_ACCURACY
                else Priority.PRIORITY_BALANCED_POWER_ACCURACY,
                CancellationTokenSource().token
            ).await()
        }
        _isLoadingLocation.value = false

        _coordinates.value =
            if (location != null) Coordinates(location.latitude, location.longitude)
            else null
        return coordinates.value
    }
}
```

# 2.4. Classe per la gestione della posizione

```
class LocationService(private val ctx: Context) {
    // ...

    suspend fun getCurrentLocation(usePreciseLocation: Boolean = false): Coordinates? {
        val locationEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)
        if (!locationEnabled) throw IllegalStateException("Location is disabled")

        val permissionGranted = ContextCompat.checkSelfPermission(
            ctx,
            Manifest.permission.ACCESS_COARSE_LOCATION
        ) == PackageManager.PERMISSION_GRANTED
        if (!permissionGranted) throw SecurityException("Location permission not granted")

        _isLoadingLocation.value = true
        val location = withContext(Dispatchers.IO) {
            fusedLocationClient.getCurrentLocation(
                if (usePreciseLocation) Priority.PRIORITY_HIGH_ACCURACY
                else Priority.PRIORITY_BALANCED_POWER_ACCURACY,
                CancellationTokenSource().token
            ).await()
        }
        _isLoadingLocation.value = false

        _coordinates.value =
            if (location != null) Coordinates(location.latitude, location.longitude)
            else null
        return coordinates.value
    }
}
```

Tramite locationManager, lanciamo un'eccezione se il GPS è disattivato

Lanciamo un'eccezione se l'utente non ha fornito i permessi necessari

# 2.4. Classe per la gestione della posizione

```
class LocationService(private val ctx: Context) {
    // ...

    suspend fun getCurrentLocation(usePreciseLocation: Boolean = false): Coordinates? {
        val locationEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)
        if (!locationEnabled) throw IllegalStateException("Location is disabled")

        val permissionGranted = ContextCompat.checkSelfPermission(
            ctx,
            Manifest.permission.ACCESS_COARSE_LOCATION
        ) == PackageManager.PERMISSION_GRANTED
        if (!permissionGranted) throw SecurityException("Location permission not granted")

        _isLoadingLocation.value = true
        val location = withContext(Dispatchers.IO) {
            fusedLocationClient.getCurrentLocation(
                if (usePreciseLocation) Priority.PRIORITY_HIGH_ACCURACY
                else Priority.PRIORITY_BALANCED_POWER_ACCURACY,
                CancellationTokenSource().token
            ).await()
        }
        _isLoadingLocation.value = false

        _coordinates.value =
            if (location != null) Coordinates(location.latitude, location.longitude)
            else null
        return coordinates.value
    }
}
```

Settiamo isLoading a true prima del caricamento della posizione e a false una volta terminata l'operazione

Per il reperimento della posizione, spostiamo l'esecuzione su un thread dedicato all'I/O

# 2.4. Classe per la gestione della posizione

- Apertura delle impostazioni del GPS
  - È un semplice Intent implicito

```
fun openLocationSettings() {
    val intent = Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS).apply {
        flags = Intent.FLAG_ACTIVITY_NEW_TASK
    }
    if (intent.resolveActivity(ctx.packageManager) != null) {
        ctx.startActivity(intent)
    }
}
```

# 2.5. Reperimento della posizione nell’interfaccia utente

- Creiamo delle variabili per i messaggi di warning che andremo a mostare in caso di permessi negati o posizione disabilitata

MainActivity.kt - Composable GPSScreen

```
var showLocationDisabledAlert by remember { mutableStateOf(false) }
var showPermissionDeniedAlert by remember { mutableStateOf(false) }
var showPermissionPermanentlyDeniedSnackbar by remember { mutableStateOf(false) }
```

# 2.5. Reperimento della posizione nell’interfaccia utente

- Creiamo un'istanza di **`LocationService`**

```
val ctx = LocalContext.current
val locationService = remember { LocationService(ctx) }
```

- Convertiamo le sue proprietà **`coordinates`** e **`isLoadingLocation`** in **`State`**

```
val coordinates by locationService.coordinates.collectAsStateWithLifecycle()
val isLoading by locationService.isLoadingLocation.collectAsStateWithLifecycle()
```

# 2.5. Reperimento della posizione nell'interfaccia utente

- Creiamo una funzione **`getCurrentLocation`** che viene eseguita in una coroutine nello scope del composable, gestendo eventuali errori

```
val scope = rememberCoroutineScope()
fun getCurrentLocation() = scope.launch {
    try {
        locationService.getCurrentLocation()
    } catch (_: IllegalStateException) {
        showLocationDisabledAlert = true
    }
}
```

# 2.5. Reperimento della posizione nell'interfaccia utente

- Utilizziamo **`rememberMultiplePermissions`** per richiedere accesso alla posizione

```
val locationPermissions = rememberMultiplePermissions(
    listOf(
        Manifest.permission.ACCESS_COARSE_LOCATION,
        Manifest.permission.ACCESS_FINE_LOCATION
    )
) { statuses ->
    when {
        statuses.any { it.value == PermissionStatus.Granted } ->
            getCurrentLocation()
        statuses.all { it.value == PermissionStatus.PermanentlyDenied } ->
            showPermissionPermanentlyDeniedSnackbar = true
        else ->
            showPermissionDeniedAlert = true
    }
}
```

# 2.5. Reperimento della posizione nell'interfaccia utente

- Utilizziamo **`rememberMultiplePermissions`** per richiedere accesso alla posizione

```
val locationPermissions = rememberMultiplePermissions(
    listOf(
        Manifest.permission.ACCESS_COARSE_LOCATION,
        Manifest.permission.ACCESS_FINE_LOCATION
    )
) { statuses ->
    when {
        statuses.any { it.value == PermissionStatus.Granted } ->
            getCurrentLocation()
        statuses.all { it.value == PermissionStatus.PermanentlyDenied } ->
            showPermissionPermanentlyDeniedSnackbar = true
        else ->
            showPermissionDeniedAlert = true
    }
}
```

Se l'utente concede anche solo uno dei due permessi, possiamo utilizzare il GPS.

# 2.5. Reperimento della posizione nell'interfaccia utente

- Utilizziamo **`rememberMultiplePermissions`** per richiedere accesso alla posizione

```
val locationPermissions = rememberMultiplePermissions(
    listOf(
        Manifest.permission.ACCESS_COARSE_LOCATION,
        Manifest.permission.ACCESS_FINE_LOCATION
    )
) { statuses ->
    when {
        statuses.any { it.value == PermissionStatus.Granted } ->
            getCurrentLocation()
        statuses.all { it.value == PermissionStatus.PermanentlyDenied } ->
            showPermissionPermanentlyDeniedSnackbar = true
        else ->
            showPermissionDeniedAlert = true
    }
}
```

Se tutti i permessi sono stati permanentemente negati , mostreremo una snackbar per accedere alle impostazioni

Se alcuni permessi sono stati temporaneamente negati, mostreremo un'alert per spiegarne l'utilizzo

# 2.5. Reperimento della posizione nell'interfaccia utente

- Creiamo una funzione **`getLocationOrRequestPermission`** che chiama **`getCurrentLocation`** se si dispone dei permessi necessari, mentre li chiede se mancanti
- Chiamiamo la funzione al click sul relativo bottone

```
fun getLocationOrRequestPermission() {
    if (locationPermissions.statuses.any { it.value.isGranted }) {
        getCurrentLocation()
    } else {
        locationPermissions.launchPermissionRequest()
    }
}
```

```
Button(onClick = ::getLocationOrRequestPermission) {
    Text("Get current location")
}
```

# 2.5. Reperimento della posizione nell’interfaccia utente

- Mostriamo un indicatore di caricamento mentre stiamo reperendo la posizione

```
if (isLoading) {
    LinearProgressIndicator(modifier = Modifier.fillMaxWidth())
}
```

# 2.5. Reperimento della posizione nell’interfaccia utente

- Mostriamo su schermo le coordinate ottenute

```
Text("Latitude: ${locationService.coordinates?.latitude ?: "-"}")
Text("Longitude: ${locationService.coordinates?.longitude ?: "-"}")
```

# 2.6. Gestione casi particolari

- Se la **posizione** è **disabilitata**, vogliamo mostrare un AlertDialog con un tasto per andare alle impostazioni del GPS, dove l'utente può abilitarla

```
if (showLocationDisabledAlert) {
    AlertDialog(
        title = { Text("Location disabled") },
        text = { Text("Location must be enabled to get your
coordinates in the app.") },
        confirmButton = {
            TextButton(onClick = {
                locationService.openLocationSettings()
                showLocationDisabledAlert = false
            }) {
                Text("Enable")
            }
        },
        dismissButton = {
            TextButton(onClick = {
                showLocationDisabledAlert = false
            }) {
                Text("Dismiss")
            }
        },
        onDismissRequest = {
            showLocationDisabledAlert = false
        }
    )
}
```

# 2.6. Gestione casi particolari

```
if (showPermissionDeniedAlert) {
    AlertDialog(
        title = { Text("Location permission denied") },
        text = { Text("Location permission is required to get
your coordinates in the app.") },
        confirmButton = {
            TextButton(onClick = {
                locationPermissions.launchPermissionRequest()
                showPermissionDeniedAlert = false
            }) {
                Text("Grant")
            }
        },
        dismissButton = {
            TextButton(onClick = {
                showPermissionDeniedAlert = false

            }) {
                Text("Dismiss")
            }
        },
        onDismissRequest = { showPermissionDeniedAlert = false }
    )
}
```

- Se l'**accesso** alla posizione è stato **negato**, ma può ancora essere richiesto, vogliamo mostrare un AlertDialog per spiegare all'utente lo scopo dell'accesso alla posizione

# 2.6. Gestione casi particolari

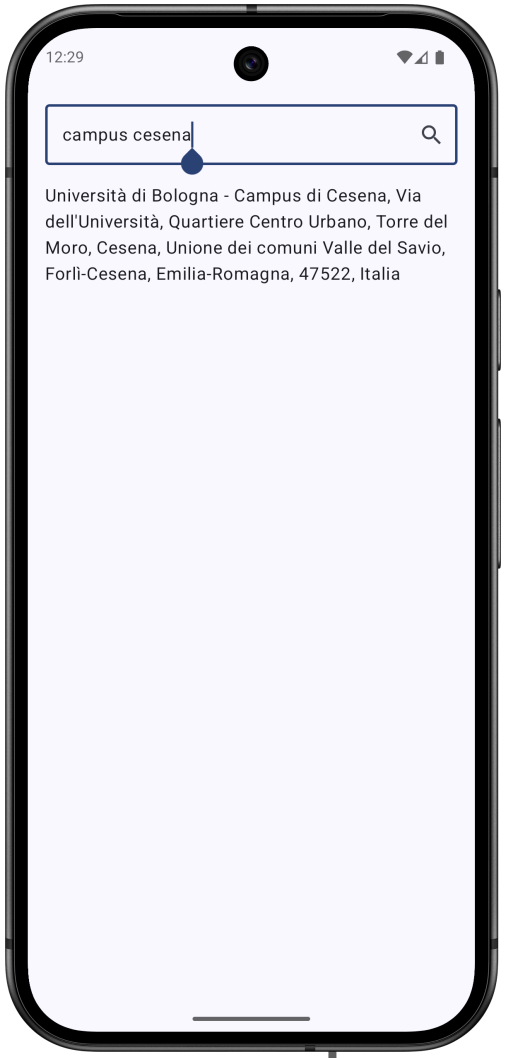
- Se l'**accesso** alla posizione è stato **permanentemente negato**, vogliamo mostare una Snackbar per concedere il permesso tramite le impostazioni di sistema

```
val ctx = LocalContext.current
if (showPermissionPermanentlyDeniedSnackbar) {
    LaunchedEffect(snackbarHostState) {
        val res = snackbarHostState.showSnackbar(
            "Location permission is required.",
            "Go to Settings",
            duration = SnackbarDuration.Long
        )
        if (res == SnackbarResult.ActionPerformed) {
            val intent = Intent(Settings.ACTION_APPLICATION_DETAILS_SETTINGS).apply {
                data = Uri.fromParts("package", ctx.packageName, null)
                flags = Intent.FLAG_ACTIVITY_NEW_TASK
            }
            if (intent.resolveActivity(ctx.packageManager) != null) {
                ctx.startActivity(intent)
            }
        }
        showPermissionPermanentlyDeniedSnackbar = false
    }
}
```

# 3. HTTP

# 3. HTTP

- Creare un’applicazione che, data una stringa di ricerca, vada a trovare un luogo associato
- Per farlo, l’app deve effettuare una richiesta HTTP alle API di OpenStreetMap
- L’app dovrà controllare la disponibilità di connessione internet prima di accedere alla rete



# 3. HTTP

- Punto di partenza: progetto Android vuoto con dependency injection già configurata
- Passaggi:
  1. Installazione dipendenze
  2. Aggiornamento manifest
  3. Utilizzo delle API di OpenStreetMap
  4. Dependency injection
  5. Interfaccia utente

# 3.1. Installazione dipendenze

- Utilizzeremo la libreria Ktor per l'invio di richieste HTTP
  - Ktor è un framework per creare applicazioni web, sia server che client, in linguaggio Kotlin
  - Nel nostro caso siamo interessati alla parte client, per cui abbiamo bisogno delle seguenti dipendenze (file **build.gradle.kts (modulo :app)**):

```
implementation("io.ktor:ktor-client-core:2.3.8")
implementation("io.ktor:ktor-client-okhttp:2.3.8")
implementation("io.ktor:ktor-client-content-negotiation:2.3.8")
implementation("io.ktor:ktor-serialization-kotlinx-json:2.3.8")
```

Replace with new library catalog declaration

# 3.1. Installazione dipendenze

- Le API di OpenStreetMap rispondono alle richieste con dati in formato JSON
- Per convertirli automaticamente in istanze di classi Kotlin, abbiamo bisogno di un plugin per la serializzazione dei dati:

libs.versions.toml

```
[plugins]
# ...
serialization = { id = "org.jetbrains.kotlin.plugin.serialization", version.ref = "kotlin" }
```

build.gradle.kts (Module :app)

```
plugins {
    // ...
    alias(libs.plugins.serialization)
}
```

# 3.2. Aggiornamento manifest

Dobbiamo aggiungere tre dichiarazioni al manifest:

1. Permesso per l'accesso allo stato della rete
   - Necessario per controllare se il dispositivo è online prima di effettuare richieste HTTP

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

# 3.2. Aggiornamento manifest

Dobbiamo aggiungere tre dichiarazioni al manifest:

1. Permesso per l'accesso allo stato della rete
2. Permesso per l'utilizzo della rete
   - Per inviare le richieste HTTP ad OpenStreetMap

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

# 3.2. Aggiornamento manifest

Dobbiamo aggiungere tre dichiarazioni al manifest:

1. Permesso per l'accesso allo stato della rete
2. Permesso per l'utilizzo della rete
3. Intent per aprire le impostazioni di rete
   - Per permettere all'utente di connettersi ad internet, se necessario

```
<queries>
    <intent>
        <action android:name="android.settings.WIRELESS_SETTINGS"/>
    </intent>
</queries>
```

# 3.2. Aggiornamento manifest

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

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

    <queries>
        <intent>
            <action android:name="android.settings.WIRELESS_SETTINGS"/>
        </intent>
    </queries>

    <!-- ... -->

</manifest>
```

# 3.3. Utilizzo delle API di OpenStreetMap

- In un package data.remote, creiamo un file OSMDataSource.kt in cui effettuare la richiesta alle API
- Partiamo definendo la struttura dei dati che ci aspettiamo di ricevere in risposta

```
@Serializable
data class OSMPlace(
    @SerialName("place_id")
    val id: Int,
    @SerialName("lat")
    val latitude: Double,
    @SerialName("lon")
    val longitude: Double,
    @SerialName("display_name")
    val displayName: String
)
```

# 3.3. Utilizzo delle API di OpenStreetMap

- Grazie alle annotazioni **`@Serializable`** e **`@SerialName`**, Ktor è in grado di estrarre i campi richiesti dalla risposta in formato JSON e convertirli in un oggetto **`OSMPlace`**

data.remote.OSMDataSource

```kotlin
@Serializable
data class OSMPlace(
    @SerialName("place_id")
    val id: Int,
    @SerialName("lat")
    val latitude: Double,
    @SerialName("lon")
    val longitude: Double,
    @SerialName("display_name")
    val displayName: String
)
```

https://nominatim.openstreetmap.org/?q=campus%20cesena&format=json&limit=1


```json
[
  {
    "place_id": 64893145,
    "licence": "Data © OpenStreetMap contributors, ODbL 1.0. http://osm.org/copyright",
    "osm_type": "way",
    "osm_id": 699360636,
    "lat": "44.1480776",
    "lon": "12.2351860386853 99",
    "class": "amenity",
    "type": "university",
    "place_rank": 30,
    "importance": 0.00000999999999995449,
    "addresstype": "amenity",
    "name": "Università di Bologna - Campus di Cesena",
    "display_name": "Università di Bologna - Campus di Cesena, Via dell'Università,
Quartiere Centro Urbano, Torre del Moro, Cesena, Unione dei comuni Valle del Savio,
Forlì-Cesena, Emilia-Romagna, 47522, Italy",
    "boundingbox": [
      "44.1474739",
      "44.1487452",
      "12.2348849",
      "12.2362448"
    ]
  }
]
```

# 3.3. Utilizzo delle API di OpenStreetMap

- A questo punto, definiamo una classe che invii la richiesta alle API
  - La classe accetta un **`HttpClient`** come parametro, che verrà fornito tramite dependency injection
  - La funzione **`searchPlaces`** è marcata come **`suspend`**, così da permettere l'esecuzione di richieste HTTP solo in una coroutine

```
class OSMDataSource(
    private val httpClient: HttpClient
) {
    companion object {
        private const val BASE_URL = "https://nominatim.openstreetmap.org"
    }

    suspend fun searchPlaces(query: String): List<OSMPlace> {
        val url = "$BASE_URL/?q=$query&format=json&limit=1"
        return httpClient.get(url).body()
    }
}
```

# 3.4. Dependency injection

- Configuriamo la dependency injection
- Abbiamo bisogno di
  - Un singleton di **`HttpClient`** da fornire a **`OSMDataSource`**
  - Un singleton di **`OSMDataSource`** che andremo a reperire lato UI

```
val appModule = module {
    single {
        HttpClient {
            install(ContentNegotiation) {
                json(Json {
                    ignoreUnknownKeys = true
                })
            }
        }
    }

    single { OSMDataSource(get()) }
}
```

# 3.5. Interfaccia utente

- Nella **`MainActivity`**, creiamo due variabili:
  - **`query`**: il testo nel TextField
  - **`result`**: la stringa contenente il risultato della ricerca

```
var query by remember { mutableStateOf("") }
var result by remember { mutableStateOf("-") }
```

# 3.5. Interfaccia utente

- Creiamo una funzione per controllare se il dispositivo è online

```
fun isOnline(): Boolean {
    val connectivityManager = ctx
        .getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
    val capabilities = connectivityManager
        .getNetworkCapabilities(connectivityManager.activeNetwork)
    return capabilities?.hasTransport(NetworkCapabilities.TRANSPORT_CELLULAR) == true ||
            capabilities?.hasTransport(NetworkCapabilities.TRANSPORT_WIFI) == true
}
```

# 3.5. Interfaccia utente

- Creiamo una funzione per controllare se il dispositivo è online

```
fun isOnline(): Boolean {
    val connectivityManager = ctx
        .getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
    val capabilities = connectivityManager
        .getNetworkCapabilities(connectivityManager.activeNetwork)
    return capabilities?.hasTransport(NetworkCapabilities.TRANSPORT_CELLULAR) == true ||
           capabilities?.hasTransport(NetworkCapabilities.TRANSPORT_WIFI) == true
}
```

Controlla la presenza di connessione cellulare

Controlla la presenza di connessione WiFi

# 3.5. Interfaccia utente

- Creiamo una funzione per aprire le impostazioni di rete tramite un Intent implicito

```
fun openWirelessSettings() {
    val intent = Intent(Settings.ACTION_WIRELESS_SETTINGS).apply {
        flags = Intent.FLAG_ACTIVITY_NEW_TASK
    }
    if (intent.resolveActivity(ctx.packageManager) != null) {
        ctx.startActivity(intent)
    }
}
```

# 3.5. Interfaccia utente

- Reperiamo il singleton di OSMDataSource
  - Siccome non è un ViewModel, dobbiamo usare l'elper **`koinInject`** invece di **`koinViewModel`**

```
val osmDataSource = koinInject<OSMDataSource>()
```

# 3.5. Interfaccia utente

- Creiamo una funzione che invia la richiesta HTTP da una coroutine

```
val scope = rememberCoroutineScope()
fun searchPlaces() = scope.launch {
    if (isOnline()) {
        result = "Loading..."
        val res = osmDataSource.searchPlaces(query)
        result = res.getOrNull(0)?.displayName ?: "Place not found"
    } else {
        val res = snackbarHostState.showSnackbar(
            message = "No Internet connectivity",
            actionLabel = "Go to Settings",
            duration = SnackbarDuration.Long
        )
        if (res == SnackbarResult.ActionPerformed) {
            openWirelessSettings()
        }
    }
}
```

# 3.5. Interfaccia utente

- Creiamo una funzione che invia la richiesta HTTP da una coroutine

```
val scope = rememberCoroutineScope()
fun searchPlaces() = scope.launch {
    if (isOnline()) {
        result = "Loading..."
        val res = osmDataSource.searchPlaces(query)
        result = res.getOrNull(0)?.displayName ?: "Place not found"
    } else {
        val res = snackbarHostState.showSnackbar(
            message = "No Internet connectivity",
            actionLabel = "Go to Settings",
            duration = SnackbarDuration.Long
        )
        if (res == SnackbarResult.ActionPerformed) {
            openWirelessSettings()
        }
    }
}
```

Se il dispositivo è online, invia la richiesta

Se il dispositivo è offline, mostra una snackbar per aprire le impostazioni di rete

# 3.5. Interfaccia utente

- Creiamo l'interfaccia utente

```
OutlinedTextField(
    value = query,
    onValueChange = { query = it },
    trailingIcon = {
        IconButton(onClick = ::searchPlaces) {
            Icon(Icons.Outlined.Search, "Search")
        }
    },
    modifier = Modifier.fillMaxWidth()
)
Spacer(Modifier.size(16.dp))
Text(result)
```

# 4. TravelDiary – finale

# 4. TravelDiary – finale

- Ora è possibile aggiungere le ultime funzionalità all'app TravelDiary:
  - Nella schermata di aggiunta di un nuovo viaggio, recuperare la posizione dell'utente al click sul  tasto del GPS
  - Inviare la posizione alle API di OpenStreetMap per reperire il nome del luogo e inserirlo nel campo "name"
  - Scattare la foto del luogo, salvarla nella galleria e associarla al Trip tramite un riferimento nel database Room

# Hint: riutilizzo del codice degli esercizi

- La maggior parte del codice necessario è già presente negli esercizi precedenti:
  - **`Permission.kt`** per la gestione dei permessi
  - **`Image.kt`** per il salvataggio delle immagini
  - **`Camera.kt`** per l'utilizzo della fotocamera
  - **`Location.kt`** per la posizione
  - **`OSMDataSource.kt`** per l'invio di richieste a OSM
    - Va aggiunta una nuova funzione **`reverse`** che riceve in input le coordinate ed effettua la richiesta all'indirizzo corretto delle API (vedi prossima slide)
  - E altro!

# Hint: API di OpenStreetMap

- Per reperire il nome di un luogo a partire dalle coordinate è necessario effettuare la seguente richiesta alle API di OpenStreetMap:
  https://nominatim.openstreetmap.org/reverse?lat=LATITUDE&lon=LONGITUDE&format=json&limit=1
- Esempio:
  https://nominatim.openstreetmap.org/reverse?lat=52.5487429714954&lon=-1.81602098644987&format=json&limit=1
- Documentazione API:
  https://nominatim.org/release-docs/latest/api/Reverse/

# Hint: riferimento all'immagine nel database

- In base all'implementazione, potrebbe essere necessario aggiungere una proprietà all'entità **`Trip`** in cui memorizzare, ad esempio, l'URI dell'immagine del luogo
- Il metodo più corretto sarebbe quello di aumentare la versione del database (**`TravelDiaryDatabase.kt`**) e creare una migrazione
- Per semplicità (ma sconsigliato per applicazioni serie), è possibile invece:
  a. Cancellare i dati dell'applicazione dal dispositivo, aggiungere la proprietà e riavviare l'app
  b. Oppure utilizzare **`.fallbackToDestructiveMigration()`** nella creazione del database, in modo da cancellare automaticamente i dati ogni volta che la struttura del database cambia

AppModule.kt

```
single {
    Room.databaseBuilder(
        get(),
        TravelDiaryDatabase::class.java,
        "travel-diary"
    )
        .fallbackToDestructiveMigration()
        .build()
}
```

https://developer.android.com/training/data-storage/room/migrating-db-versions

# Tocca a voi!

# Riferimenti

- Fotocamera
  https://developer.android.com/reference/androidx/activity/result/contract/ActivityResultContracts.TakePicture
- Ottenere un risultato di un'activity in un composable
  https://developer.android.com/reference/kotlin/androidx/activity/compose/package-summary#rememberLauncherForActivityResult(androidx.activity.result.contract.ActivityResultContract,kotlin.Function1)
- Posizione
  https://developer.android.com/develop/sensors-and-location/location
- Ktor (client)
  https://ktor.io/docs/client-create-and-configure.html