Fotocamera, GPS, HTTP

Esempi di utilizzo

Laboratorio di oggi

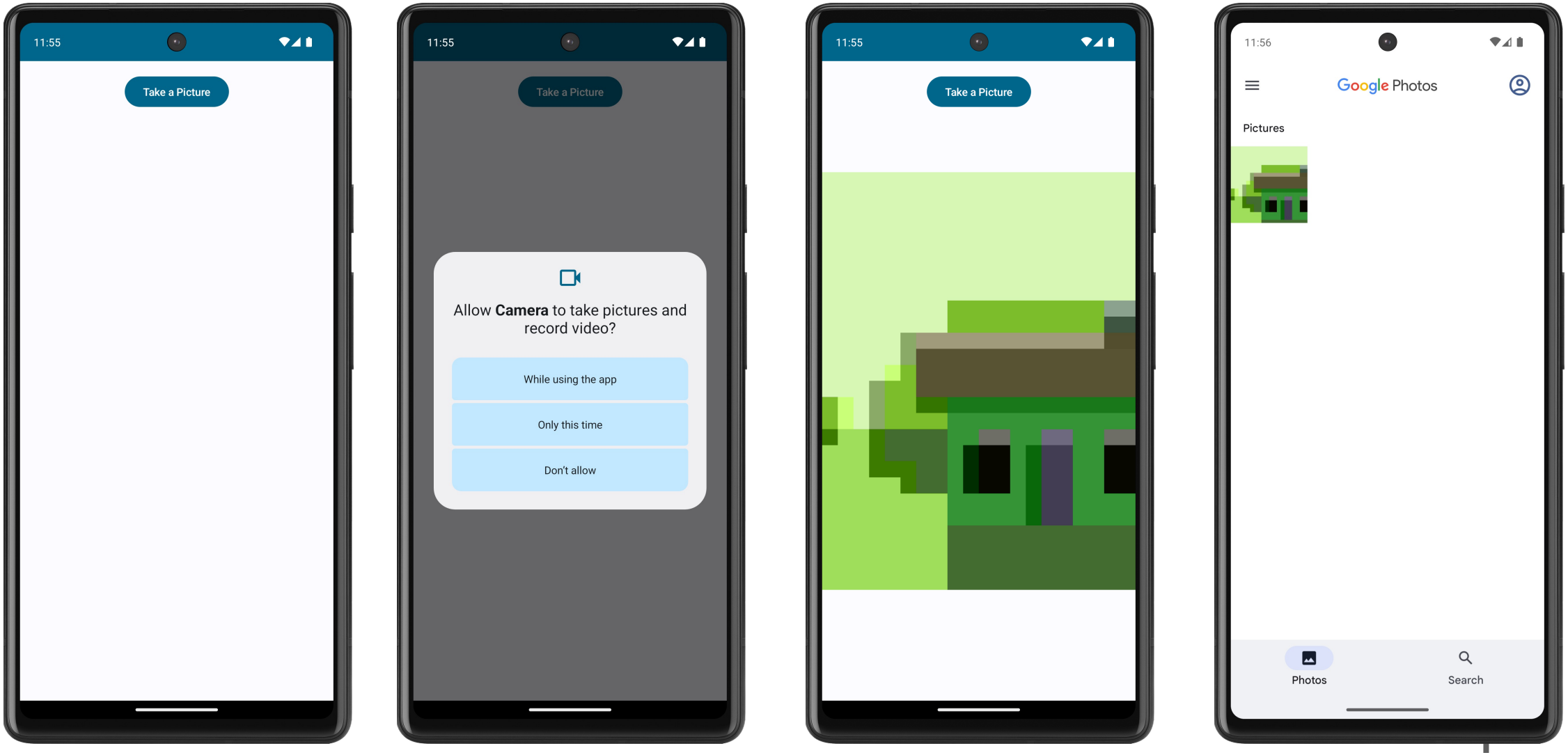
1. Fotocamera (esercitazione guidata)
   - Scattare foto e salvarle nella memoria del dispositivo
   - Progetto di partenza su Virtuale: **CameraBase.zip**
2. GPS (esercitazione guidata)
   - Ottenere la posizione del dispositivo
   - Progetto di partenza su Virtuale: **GPSBase.zip**
3. HTTP (esercitazione 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



Passaggi

1. Installazione dipendenze
2. Aggiornamento manifest
3. Creazione path provider
4. Gestione dei permessi
5. Scatto della foto
6. 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")
```

1.2. Aggiornamento manifest

- Dobbiamo aggiornare il manifest per:
  - Accedere alla fotocamera
  - Salvare immagini nello storage del dispositivo
  - Salvare immagini nella cache dell'app

1.2. Aggiornamento manifest

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-feature android:name="android.hardware.camera.any"/>

    <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.CAMERA"/>
    <uses-feature android:name="android.hardware.camera.any"/>

    <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 alla fotocamera

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. Gestione dei permessi

- In **`utils/Permission.kt`**, creiamo un helper per la gestione dei permessi
- Lo sfrutteremo in questo esercizio per autorizzare l'accesso alla fotocamera e nel successivo per il GPS
- Può essere utilizzato per richiedere qualsiasi tipo di permesso
- 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

1.4. 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
}
```

1.4. Gestione dei permessi

- Creiamo un'interfaccia **`PermissionHandler`** che contiene:
  - La stringa del permesso da richiedere
  - Lo stato del permesso
  - Una funzione per richiedere il permesso

utils/permission.kt

```
interface PermissionHandler {
    val permission: String
    val status: PermissionStatus
    fun launchPermissionRequest()
}
```

1.4. Gestione dei permessi

- Creiamo anche un composable **`rememberPermission`**
  - Prende come parametro il permesso da richiedere e una funzione da eseguire al cambiamento di stato del permesso
  - Restituisce un'istanza di **`PermissionHandler`**
  - Non genera interfaccia utente → nome con l'iniziale minuscola

utils/permission.kt

```
@Composable
fun rememberPermission(
    permission: String,
    onResult: (status: PermissionStatus) -> Unit = {}
): PermissionHandler {
    // ...
}
```

1.4. Gestione dei permessi

- Implementazione del composable:

```
var status by remember { mutableStateOf(PermissionStatus.Unknown) }
val activity = (LocalContext.current as ComponentActivity)

val permissionLauncher = rememberLauncherForActivityResult(
    ActivityResultContracts.RequestPermission()
) { isGranted ->
    status = when {
        isGranted -> PermissionStatus.Granted
        activity.shouldShowRequestPermissionRationale(permission) -> PermissionStatus.Denied
        else -> PermissionStatus.PermanentlyDenied
    }
    onResult(status)
}

val permissionHandler by remember {
    derivedStateOf {
        object : PermissionHandler {
            override val permission = permission
            override val status = status
            override fun launchPermissionRequest() = permissionLauncher.launch(permission)
        }
    }
}
return permissionHandler
```

1.4. Gestione dei permessi

- Implementazione del composable:

```
var status by remember { mutableStateOf(PermissionStatus.Unknown) }
val activity = (LocalContext.current as ComponentActivity)

val permissionLauncher = rememberLauncherForActivityResult(
    ActivityResultContracts.RequestPermission()
) { isGranted ->
    status = when {
        isGranted -> PermissionStatus.Granted
        activity.shouldShowRequestPermissionRationale(permission) -> PermissionStatus.Denied
        else -> PermissionStatus.PermanentlyDenied
    }
    onResult(status)
}

val permissionHandler by remember {
    derivedStateOf {
        object : PermissionHandler {
            override val permission = permission
            override val status = status
            override fun launchPermissionRequest() = permissionLauncher.launch(permission)
        }
    }
}
return permissionHandler
```

Stato del permesso

Activity launcher per la richiesta del permesso

Esecuzione della callback passata come parametro dopo ogni richiesta di permesso

1.4. Gestione dei permessi

- Implementazione del composable:

```
var status by remember { mutableStateOf(PermissionStatus.Unknown) }
val activity = (LocalContext.current as ComponentActivity)

val permissionLauncher = rememberLauncherForActivityResult(
    ActivityResultContracts.RequestPermission()
) { isGranted ->
    status = when {
        isGranted -> PermissionStatus.Granted
        activity.shouldShowRequestPermissionRationale(permission) -> PermissionStatus.Denied
        else -> PermissionStatus.PermanentlyDenied
    }
    onResult(status)
}

val permissionHandler by remember {
    derivedStateOf {
        object : PermissionHandler {
            override val permission = permission
            override val status = status
            override fun launchPermissionRequest() = permissionLauncher.launch(permission)
        }
    }
}
return permissionHandler
```

Con remember + derivedStateOf, PermissionHandler viene ricreato solo se le sue dipendenze cambiano (permission, status, permissionLauncher)

1.4. 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 nel prossimo esercizio

1.4. Gestione dei permessi

- Esempio di utilizzo dell'helper:

```
val cameraPermission = rememberPermission(
    Manifest.permission.CAMERA
) { status ->
    if (status.isGranted) {
        cameraLauncher.launch(imageUri)
    } else {
        Toast.makeText(ctx, "Permission denied", Toast.LENGTH_SHORT).show()
    }
}
```

1.5. 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 {
    // ...
}
```

1.5. Scatto della foto

1. Creiamo l’immagine in cache

```
val ctx = LocalContext.current
val imageUri = remember {
    val imageFile = File.createTempFile("tmp_image", ".jpg", ctx.externalCacheDir)
    FileProvider.getUriForFile(ctx, ctx.packageName + ".provider", imageFile)
}
```

1.5. Scatto della foto

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

```
var capturedImageUri by remember { mutableStateOf(Uri.EMPTY) }
val cameraActivityLauncher =
    rememberLauncherForActivityResult(ActivityResultContracts.TakePicture()) { pictureTaken ->
        if (pictureTaken) {
            capturedImageUri = imageUri
            onPictureTaken(capturedImageUri)
        }
    }
```

1.5. Scatto della foto

1. Creiamo l'immagine in cache
2. Creiamo il launcher per l'activity della fotocamera
3. Creiamo l'oggetto **`CameraLauncher`** con **`remember`** + **`derivedStateOf`**, in modo che non venga re-istanziato ad ogni recomposition

```
val cameraLauncher by remember {
    derivedStateOf {
        object : CameraLauncher {
            override val capturedImageUri = capturedImageUri
            override fun captureImage() = cameraActivityLauncher.launch((imageUri))
        }
    }
}
```

1.5. Scatto della foto

utils/camera.kt

```
@Composable
fun rememberCameraLauncher(onPictureTaken: (imageUri: Uri) -> Unit = {}): CameraLauncher {
    val ctx = LocalContext.current
    val imageUri = remember {
        val imageFile = File.createTempFile("tmp_image", ".jpg", ctx.externalCacheDir)
        FileProvider.getUriForFile(ctx, ctx.packageName + ".provider", imageFile)
    }
    var capturedImageUri by remember { mutableStateOf(Uri.EMPTY) }
    val cameraActivityLauncher =
        rememberLauncherForActivityResult(ActivityResultContracts.TakePicture()) { pictureTaken ->
            if (pictureTaken) {
                capturedImageUri = imageUri
                onPictureTaken(capturedImageUri)
            }
        }

    val cameraLauncher by remember {
        derivedStateOf {
            object : CameraLauncher {
                override val capturedImageUri = capturedImageUri
                override fun captureImage() = cameraActivityLauncher.launch((imageUri))
            }
        }
    }
    return cameraLauncher
}
```

Risultato

1.5. Scatto della foto

- Nella MainActivity, utilizziamo i due helper appena creati per gestire i permessi della fotocamera e scattare la foto:

```
val cameraLauncher = rememberCameraLauncher { imageUri ->
    saveImageToStorage(imageUri, ctx.applicationContext.contentResolver)
}

val cameraPermission = rememberPermission(Manifest.permission.CAMERA) { status ->
    if (status.isGranted) {
        cameraLauncher.captureImage()
    } else {
        Toast.makeText(ctx, "Permission denied", Toast.LENGTH_SHORT).show()
    }
}

fun takePicture() =
    if (cameraPermission.status.isGranted) {
        cameraLauncher.captureImage()
    } else {
        cameraPermission.launchPermissionRequest()
    }
```

1.5. Scatto della foto

- Nella MainActivity, utilizziamo i due helper appena creati per gestire i permessi della fotocamera e scattare la foto:

```
val cameraLauncher = rememberCameraLauncher { imageUri ->
    saveImageToStorage(imageUri, ctx.applicationContext.contentResolver)
}

val cameraPermission = rememberPermission(Manifest.permission.CAMERA) { status ->
    if (status.isGranted) {
        cameraLauncher.captureImage()
    } else {
        Toast.makeText(ctx, "Permission denied", Toast.LENGTH_SHORT).show()
    }
}

fun takePicture() =
    if (cameraPermission.status.isGranted) {
        cameraLauncher.captureImage()
    } else {
        cameraPermission.launchPermissionRequest()
    }
```

Salviamo ogni foto scattata nel dispositivo. Funzione già fornita nel codice di partenza

1.5. Scatto della foto

- Nella MainActivity, utilizziamo i due helper appena creati per gestire i permessi della fotocamera e scattare la foto:

```
val cameraLauncher = rememberCameraLauncher { imageUri ->
    saveImageToStorage(imageUri, ctx.applicationContext.contentResolver)
}

val cameraPermission = rememberPermission(Manifest.permission.CAMERA) { status ->
    if (status.isGranted) {
        cameraLauncher.captureImage()
    } else {
        Toast.makeText(ctx, "Permission denied", Toast.LENGTH_SHORT).show()
    }
}

fun takePicture() =
    if (cameraPermission.status.isGranted) {
        cameraLauncher.captureImage()
    } else {
        cameraPermission.launchPermissionRequest()
    }
```

1. Se il permesso è già stato concesso, scattiamo la foto. In caso contrario, richiediamo il permesso

1.5. Scatto della foto

- Nella MainActivity, utilizziamo i due helper appena creati per gestire i permessi della fotocamera e scattare la foto:

```
val cameraLauncher = rememberCameraLauncher { imageUri ->
    saveImageToStorage(imageUri, ctx.applicationContext.contentResolver)
}

val cameraPermission = rememberPermission(Manifest.permission.CAMERA) { status ->
    if (status.isGranted) {
        cameraLauncher.captureImage()
    } else {
        Toast.makeText(ctx, "Permission denied", Toast.LENGTH_SHORT).show()
    }
}

fun takePicture() =
    if (cameraPermission.status.isGranted) {
        cameraLauncher.captureImage()
    } else {
        cameraPermission.launchPermissionRequest()
    }
```

2. Quando il permesso viene concesso, avviamo immediatamente la fotocamera

1.6. UI per la foto scattata

- Utilizziamo Async image 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"
    )
}
```

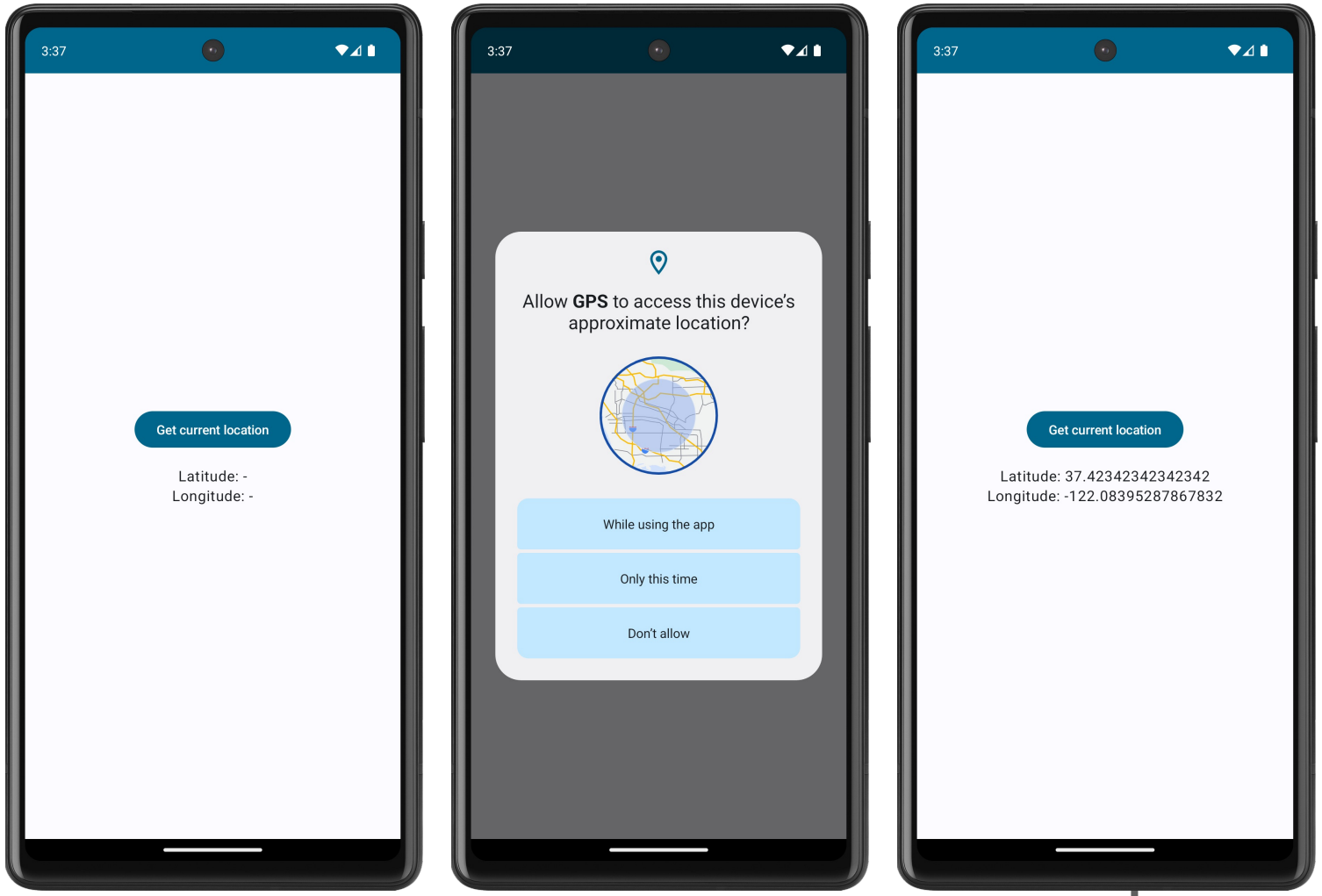
Nota finale: salvataggio file con API ≤ 28

- Per API version ≤ 28 (Android ≤ 9), è necessario richiedere un permesso aggiuntivo per il salvataggio di file sullo storage del dispositivo
- Il permesso è **`android.permission.WRITE_EXTERNAL_STORAGE`**, ed è ottenibile tramite l'helper rememberPermission già usato per la fotocamera

2. GPS

2. GPS

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



Passaggi

1. Installazione dipendenze
2. Aggiornamento manifest
3. Classe per la gestione della posizione
4. Reperimento della posizione nell'interfaccia utente
5. Gestione casi particolari: permessi negati, GPS disattivato

2.1. Installazione dipendenze

- Per l'accesso alla posizione è necessaria la seguente dipendenza (file **`build.gradle.kts`** (modulo :app)):

```
implementation("com.google.android.gms:play-services-location:21.2.0")
```

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 $50m^2$

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

    <uses-permission android:name="android.permission.ACCESS_COARSE_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. 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:
  - Apertura delle impostazioni del GPS
  - Richiesta di monitoraggio della posizione
  - Sospensione, ripresa e terminazione del monitoraggio

2.3. Classe per la gestione della posizione

- Classi necessarie a **`LocationService`**:
  - Enum per rappresentare lo stato del monitoraggio

```
enum class MonitoringStatus { Monitoring, Paused, NotMonitoring }
```

  - Data class per memorizzare una coppia di coordinate GPS

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

2.3. Classe per la gestione della posizione

- L'implementazione delle parti importanti è spiegata di seguito
- Per il resto del codice, fare riferimento alle soluzioni su Virtuale

```
class LocationService(private val ctx: Context) {
    var isLocationEnabled: Boolean
    var monitoringStatus: MonitoringStatus
    var coordinates: Coordinates?

    private val fusedLocationProviderClient
        : FusedLocationProviderClient
    private val locationRequest: LocationRequest
    private val locationCallback: LocationCallback

    fun openLocationSettings() { /* ... */ }

    fun requestCurrentLocation() { /* ... */ }

    fun endLocationRequest() { /* ... */ }

    fun pauseLocationRequest() { /* ... */ }

    fun resumeLocationRequest() { /* ... */ }
}
```

2.3. Classe per la gestione della posizione

- Abbiamo bisogno di tre oggetti principali per la gestione della posizione
  - **`fusedLocationProviderClient`** gestisce l'avvio e l'interruzione del monitoraggio
  - **`locationRequest`** definisce i parametri del monitoraggio (es. livello di precisione, intervallo aggiornamento)
  - **`locationCallback`** definisce le azioni da effettuare all'ottenimento di una nuova posizione

```
private val fusedLocationProviderClient =
    LocationServices.getFusedLocationProviderClient(ctx)

private val locationRequest =
    LocationRequest.Builder(Priority.PRIORITY_HIGH_ACCURACY, 10000)
        .apply {
            setGranularity(Granularity.GRANULARITY_PERMISSION_LEVEL)
        }
        .build()

private val locationCallback = object : LocationCallback() {
    override fun onLocationResult(p0: LocationResult) {
        super.onLocationResult(p0)
        with(p0.locations.last()) {
            coordinates = Coordinates(latitude, longitude)
        }
        endLocationRequest()
    }
}
```

2.3. Classe per la gestione della posizione

- Avvio del monitoraggio:

```
fun requestCurrentLocation() {
    val locationManager = ctx.getSystemService(Context.LOCATION_SERVICE) as LocationManager
    isLocationEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)
    if (isLocationEnabled != true) return

    val permissionGranted = ContextCompat.checkSelfPermission(
        ctx,
        Manifest.permission.ACCESS_COARSE_LOCATION
    ) == PackageManager.PERMISSION_GRANTED
    if (!permissionGranted) return

    fusedLocationProviderClient.requestLocationUpdates(
        locationRequest,
        locationCallback,
        Looper.getMainLooper()
    )
    monitoringStatus = MonitoringStatus.Monitoring
}
```

2.3. Classe per la gestione della posizione

- Avvio del monitoraggio:

```
fun requestCurrentLocation() {
    val locationManager = ctx.getSystemService(Context.LOCATION_SERVICE) as LocationManager
    isLocationEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)
    if (isLocationEnabled != true) return

    val permissionGranted = ContextCompat.checkSelfPermission(
        ctx,
        Manifest.permission.ACCESS_COARSE_LOCATION
    ) == PackageManager.PERMISSION_GRANTED
    if (!permissionGranted) return

    fusedLocationProviderClient.requestLocationUpdates(
        locationRequest,
        locationCallback,
        Looper.getMainLooper()
    )
    monitoringStatus = MonitoringStatus.Monitoring
}
```

Esce anticipatamente se la posizione è disabilitata o mancano i permessi

2.3. Classe per la gestione della posizione

- Interruzione del monitoraggio:

```
fun endLocationRequest() {
    if (monitoringStatus == MonitoringStatus.NotMonitoring) return
    fusedLocationProviderClient.removeLocationUpdates(locationCallback)
    monitoringStatus = MonitoringStatus.NotMonitoring
}
```

2.3. 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)
    }
}
```

2.4. Reperimento della posizione nell'interfaccia utente

- Creiamo l'istanza di **`LocationService`**
  - Avendo bisogno dell'istanza completa dell'activity, è necessario marcarlo come **`lateinit`** e inizializzarlo nell'**`onCreate`**
- Sospendiamo il monitoraggio quanto l'activity non è in foreground (**`onPause`**) e lo riprendiamo in **`onResume`**

```
class MainActivity : ComponentActivity() {
    private lateinit var locationService: LocationService

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        locationService = LocationService(this)

        // ...
}

    override fun onPause() {
        super.onPause()
        locationService.pauseLocationRequest()
    }

    override fun onResume() {
        super.onResume()
        locationService.resumeLocationRequest()
    }
}
```

2.4. Reperimento della posizione nell'interfaccia utente

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

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

2.4. Reperimento della posizione nell'interfaccia utente

- Nella MainActivity:
  - Creiamo delle variabili per i messaggi di warning che andremo a mostare in caso di permessi negati o posizione disabilitata
  - Gestiamo il permesso per l'accesso alla posizione tramite l'helper dell'esercizio precedente

```
val locationPermission = rememberPermission(
    Manifest.permission.ACCESS_COARSE_LOCATION
) { status ->
    when (status) {
        PermissionStatus.Granted -> locationService.requestCurrentLocation()
        PermissionStatus.Denied -> showPermissionDeniedAlert = true
        PermissionStatus.PermanentlyDenied -> showPermissionPermanentlyDeniedSnackbar = true
        PermissionStatus.Unknown -> {}
    }
}
```

2.4. Reperimento della posizione nell'interfaccia utente

- Nella MainActivity:
  - Creiamo delle variabili per i messaggi di warning che andremo a mostare in caso di permessi negati o posizione disabilitata
  - Gestiamo il permesso per l'accesso alla posizione tramite l'helper dell'esercizio precedente
  - Accediamo alla posizione tramite **`LocationService`**

```
fun requestLocation() =
    if (locationPermission.status.isGranted) {
        locationService.requestCurrentLocation()
    } else {
        locationPermission.launchPermissionRequest()
    }
```

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

2.4. Reperimento della posizione nell'interfaccia utente

- Nella MainActivity:
  - Creiamo delle variabili per i messaggi di warning che andremo a mostare in caso di permessi negati o posizione disabilitata
  - Gestiamo il permesso per l'accesso alla posizione tramite l'helper dell'esercizio precedente
  - Accediamo alla posizione tramite **`LocationService`**
  - Mostriamo su schermo le coordinate ottenute

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

2.5. 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
current location in the app.") },
        confirmButton = {
            TextButton(onClick = {
                locationService.openLocationSettings()
                showLocationDisabledAlert = false
            }) {
                Text("Enable")
            }
        },
        dismissButton = {
            TextButton(onClick = {
                showLocationDisabledAlert = false
            }) {
                Text("Dismiss")
            }
        },
        onDismissRequest = {
            showLocationDisabledAlert = false
        }
    )
}
```

2.5. Gestione casi particolari

```
if (showPermissionDeniedAlert) {
    AlertDialog(
        title = { Text("Location permission denied") },
        text = { Text("Location permission is required to get
your current location in the app.") },
        confirmButton = {
            TextButton(onClick = {
                locationPermission.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

2.5. Gestione casi particolari

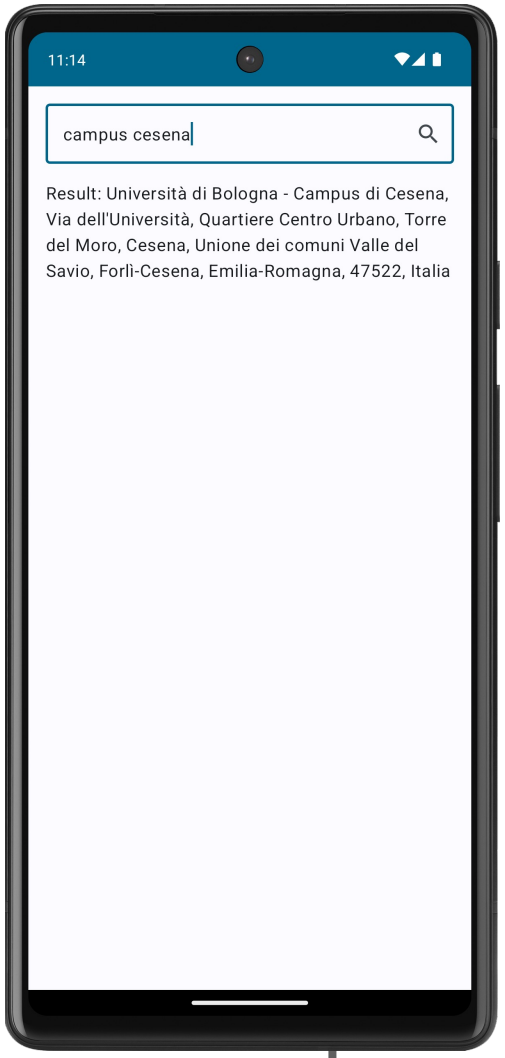
- Se l'**accesso** alla posizione è stato **permanentemente negato**, vogliamo mostare una Snackbar per concedere il permesso tramite le impostazioni di sistema

```
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)):

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

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 (file **`build.gradle.kts`** (modulo :app)):

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

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
) {
    private val baseUrl = "https://nominatim.openstreetmap.org"

    suspend fun searchPlaces(query: String): List<OSMPlace> {
        val url = "$baseUrl/?q=$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 tre variabili:
  - **`text`**: il testo nel TextField
  - **`place`**: il risultato della ricerca
  - **`placeNotFound`**: booleana che indica se la ricerca non ha dato risultati

```
var text by remember { mutableStateOf("") }
var place by remember { mutableStateOf<OSMPlace?>(null) }
var placeNotFound by remember { mutableStateOf(false) }
```

3.5. Interfaccia utente

- Creiamo una funzione per controllare se il dispositivo è online

```
fun isOnline(): Boolean {
    val connectivityManager = ctx
        .applicationContext
        .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
        .applicationContext
        .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(applicationContext.packageManager) != null) {
        applicationContext.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 coroutineScope = rememberCoroutineScope()
fun searchPlaces() = coroutineScope.launch {
    if (isOnline()) {
        val res = osmDataSource.searchPlaces(text)
        place = res.getOrNull(0)
        placeNotFound = res.isEmpty()
    } 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 coroutineScope = rememberCoroutineScope()
fun searchPlaces() = coroutineScope.launch {
    if (isOnline()) {
        val res = osmDataSource.searchPlaces(text)
        place = res.getOrNull(0)
        placeNotFound = res.isEmpty()
    } 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 = text,
    onValueChange = { text = it },
    trailingIcon = {
        IconButton(onClick = ::searchPlaces) {
            Icon(Icons.Outlined.Search, "Search")
        }
    },
    modifier = Modifier.fillMaxWidth()
)
Spacer(Modifier.size(16.dp))
Text("Result: ${when {
    place != null -> place?.displayName
    placeNotFound -> "Place not found"
    else -> "-"
}}")
```

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 e salvarla sia nella galleria che 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
  - **`LocationService.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 **`Place`** 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