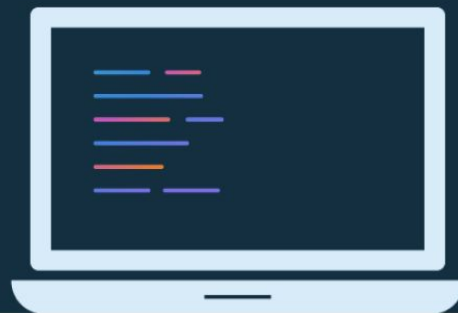




# Lesson 11: Connect to the internet



# About this lesson

## Lesson 11: Connect to the internet

- [Android permissions](#)
- [Connect to, and use, network resources](#)
- [Connect to a web service](#)
- [Display images](#)
- [Summary](#)

# Android permissions

# Permissions

- Protect the privacy of an Android user
- Declared with the `<uses-permission>` tag in the `AndroidManifest.xml`

# Permissions granted to your app

- Permissions can be granted during installation or runtime, depending on protection level.
- Each permission has a protection level: normal, signature, or dangerous.
- For permissions granted during runtime, prompt users to explicitly grant or deny access to your app.

# Permission protection levels

Protection Level	Granted when?	Must prompt before use?	Examples
Normal	Install time	No	ACCESS_WIFI_STATE, BLUETOOTH, VIBRATE, INTERNET
Signature	Install time	No	N/A
Dangerous	Runtime	Yes	GET_ACCOUNTS, CAMERA, CALL_PHONE

# Add permissions to the manifest

In `AndroidManifest.xml`:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sampleapp">

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

    <application>
        <activity
            android:name=".MainActivity" ... >
            ...
        </activity>
    </application>
</manifest>
```

# Internet access permissions

In `AndroidManifest.xml`:

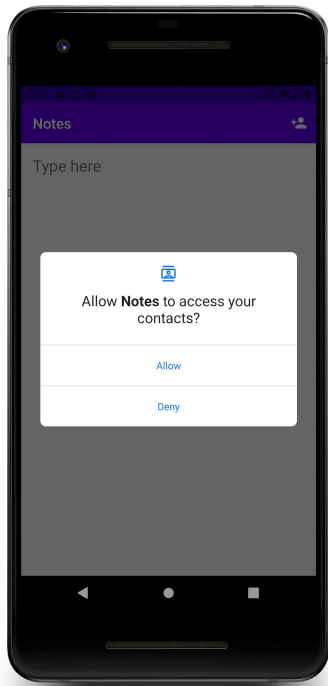
```
<uses-permission android:name="android.permission.INTERNET" />  
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```



# Request dangerous permissions

- Prompt the user to grant the permission when they try to access functionality that requires a dangerous permission.
- Explain to the user why the permission is needed.
- Fall back gracefully if the user denies the permission (app should still function).

# Prompt for dangerous permission



# App permissions best practices

- Only use the permissions necessary for your app to work.
- Pay attention to permissions required by libraries.
- Be transparent.
- Make system accesses explicit.

# Connect to, and use, network resources

# Retrofit

- Networking library that turns your HTTP API into a Kotlin and Java interface
- Enables processing of requests and responses into objects for use by your apps
  - Provides base support for parsing common response types, such as XML and JSON
  - Can be extended to support other response types

# Why use Retrofit?

- Builds on industry standard libraries, like OkHttp, that provide:
  - HTTP/2 support
  - Connection pooling
  - Response caching and enhanced security
- Frees the developer from the scaffolding setup needed to run a request

# Add Gradle dependencies

```
implementation "com.squareup.retrofit2:retrofit:2.9.0"  
implementation "com.squareup.retrofit2:converter-moshi:2.9.0"  
  
implementation "com.squareup.moshi:moshi:$moshi_version"  
implementation "com.squareup.moshi:moshi-kotlin:$moshi_version"  
kapt "com.squareup.moshi:moshi-kotlin-codegen:$moshi_version"
```

# Connect to a web service



# HTTP methods

- GET
- POST
- PUT
- DELETE

# Example web service API

URL	DESCRIPTION	METHOD
<code>example.com/posts</code>	Get a list of all posts	GET
<code>example.com/posts/username</code>	Get a list of posts by user	GET
<code>example.com/posts/search?filter=queryterm</code>	Search posts using a filter	GET
<code>example.com/posts/new</code>	Create a new post	POST

# Define a Retrofit service

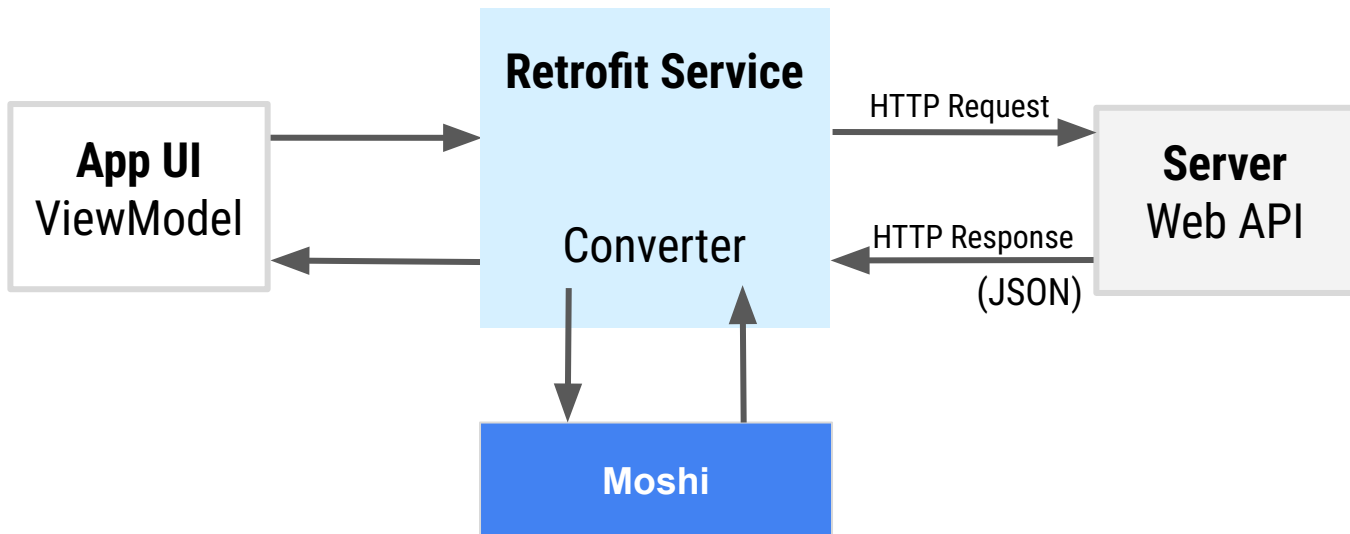
```
interface SimpleService {  
    @GET("posts")  
    suspend fun listPosts(): List<Post>  
  
    @GET("posts/{userId}")  
    suspend fun listByUser(@Path("userId") userId:String): List<Post>  
  
    @GET("posts/search") // becomes post/search?filter=query  
    suspend fun search(@Query("filter") search: String): List<Post>  
  
    @POST("posts/new")  
    suspend fun create(@Body post : Post): Post  
}
```

# Create a Retrofit object for network access

```
val retrofit = Retrofit.Builder()
    .baseUrl("https://example.com")
    .addConverterFactory(...)
    .build()

val service = retrofit.create(SimpleService::class.java)
```

# End-to-end diagram



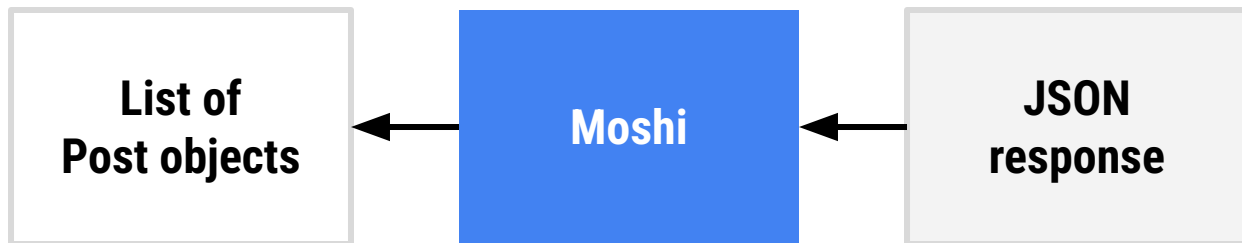
# Converter.Factory

Helps convert from a response type into class objects

- JSON (Gson or Moshi)
- XML (Jackson, SimpleXML, JAXB)
- Protocol buffers
- Scalars (primitives, boxed, and Strings)

# Moshi

- JSON library for parsing JSON into objects and back
- Add Moshi library dependencies to your app's Gradle file.
- Configure your Moshi builder to use with Retrofit.



# Moshi JSON encoding

```
@JsonClass(generateAdapter = true)
data class Post (
    val title: String,
    val description: String,
    val url: String,
    val updated: String,
    val thumbnail: String,
    val closedCaptions: String?)
```



# JSON code

```
{  
  "title": "Android Jetpack: EmojiCompat",  
  "description": "Android Jetpack: EmojiCompat",  
  "url": "https://www.youtube.com/watch?v=sYGKUtm2ga8",  
  "updated": "2018-06-07T17:09:43+00:00",  
  "thumbnail": "https://i4.ytimg.com/vi/sYGKUtm2ga8/hqdefault.jpg"  
}
```

# Set up Retrofit and Moshi

```
private val moshi = Moshi.Builder()
    .add(KotlinJsonAdapterFactory())
    .build()

val retrofit = Retrofit.Builder()
    .addConverterFactory(MoshiConverterFactory.create(moshi))
    .baseUrl(BASE_URL)
    .build()

object API {
    val retrofitService : SimpleService by lazy {
        retrofit.create(SimpleService::class.java)
    }
}
```

# Use Retrofit with coroutines

Launch a new coroutine in the view model:

```
viewModelScope.launch {  
    Log.d("posts", API.retrofitService.searchPosts("query"))  
}
```

# Display images

# Glide

- Third-party image-loading library in Android
- Focused on performance for smoother scrolling
- Supports images, video stills, and animated GIFs

# Add Gradle dependency

```
implementation "com.github.bumptech.glide:glide:$glide_version"
```

# Load an image

```
Glide.with(fragment)
    .load(url)
    .into(imageView);
```

# Customize a request with RequestOptions

- Apply a crop to an image
- Apply transitions
- Set options for placeholder image or error image
- Set caching policies



# RequestOptions example

```
@BindingAdapter("imageUrl")
fun bindImage(imgView: ImageView, imgUrl: String?) {
    imgUrl?.let {
        val imgUri = imgUrl.toUri().buildUpon().scheme("https").build()

        Glide.with(imgView)
            .load(imgUri)
            .apply(RequestOptions()
                .placeholder(R.drawable.loading_animation)
                .error(R.drawable.ic_broken_image))
            .into(imgView)
    }
}
```

# Summary

# Summary

In Lesson 11, you learned how to:

- Declare permissions your app needs in `AndroidManifest.xml`
- Use the three protection levels for permissions: normal, signature, and dangerous (prompt the user at runtime for dangerous permissions)
- Use the Retrofit library to make web service API calls from your app
- Use the Moshi library to parse JSON response into class objects
- Load and display images from the internet using the `Glide` library

# Learn More

- [App permissions best practices](#)
- [Retrofit](#)
- [Moshi](#)
- [Glide](#)

# Pathway

Practice what you've learned by completing the pathway:

[Lesson 11: Connect to the internet](#)

