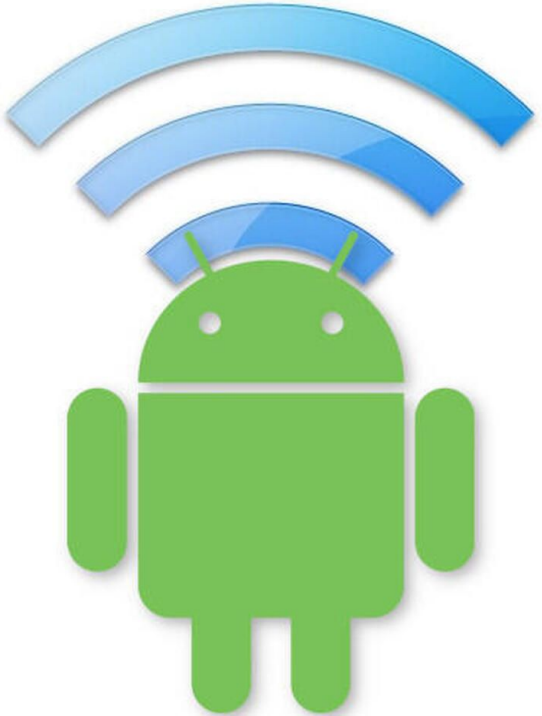

Networks

# Overview

- Permissions
- Retrofit
- GSON Converter Factory
- Display Web Hosted Images



# Permissions

# Permissions

- 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.

## Android Manifest

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

## Internet

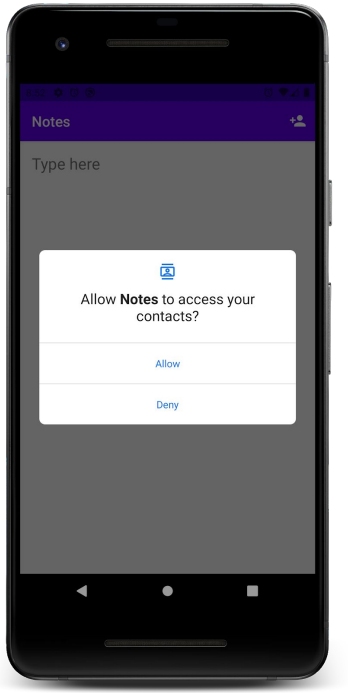
```
<uses-permission android:name="android.permission.INTERNET"/>
```

## Check Network State

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

# Permissions - Best Practices

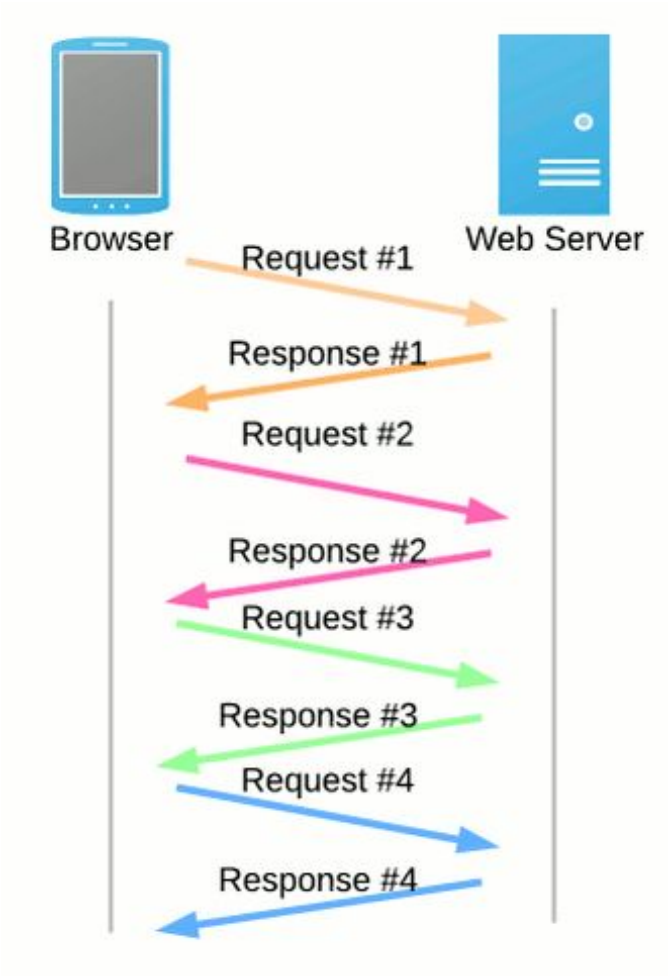
- Only use the permissions necessary for your app to work.
- Pay attention to permissions required by libraries.
- Be transparent.
- Make system accesses explicit.



# Permissions - Protection Levels

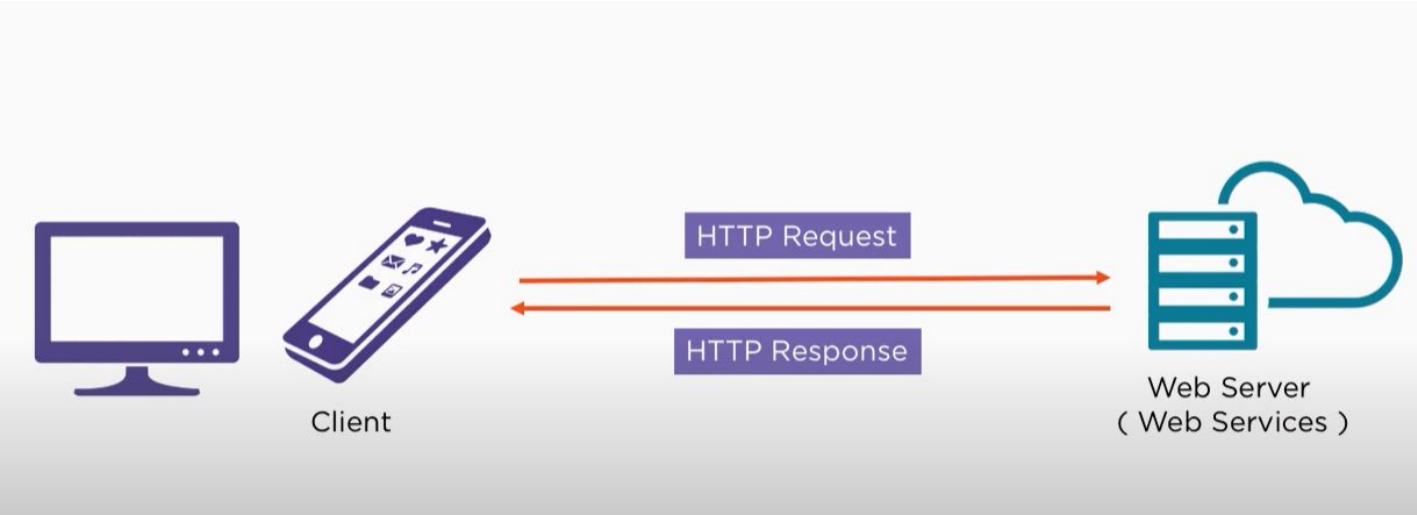
| Protection Level | Granted when? | Must prompt before use? | Examples |
| --- | --- | --- | --- |
| Normal | Install time | No | `ACCESS_WIFI_STATE, BLUETOOTH, VIBRATE, INTERNET` |
| Signature | Install time | No | N/A |
| Dangerous | Runtime | Yes | `GET_ACCOUNTS, CAMERA, CALL_PHONE` |

# HTTP


Browser
Web Server
Request #1
Response #1
Request #2
Response #2
Request #3
Response #3
Request #4
Response #4

# How does HTTP work?

- Okhttp is a library that handles http at a much lower level
- Client makes an http request, while server sends and http response



# HTTP Request Structure

Request Line: Method, URL, HTTP version

Request Headers: Meta Data

Request Body: (Optional) Send data to Server

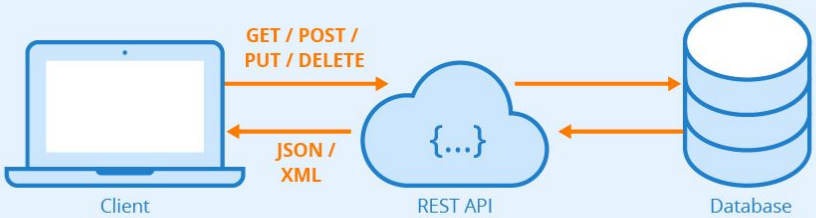
Eg. JSON data for Student objects

```
{
    “name” = “Joseph”,
    “age” = 17
}
```

# Request Line

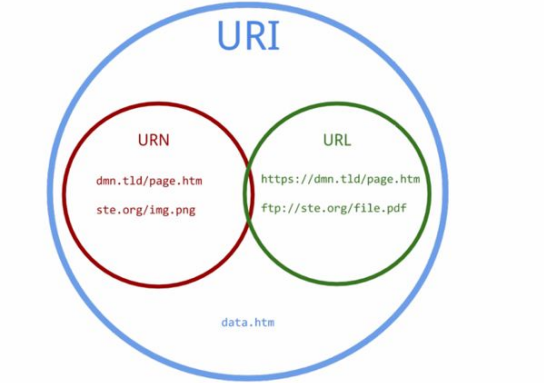
## Methods

- POST - add a resource
- GET - retrieve a resource
- PUT - replace a resource
- PATCH - update part of  a resource
- DELETE - remove a resource




## URI - point of connection to the API

- file://
- http:// and https://
- content://



## Request Headers

- This is where you can specify the data format which information is packaged and transferred as

## Request Body

- This is where you can specify the data you want to send to the server

# Common HTTP Request Methods

| HTTP verb | CRUD | Description |
| --- | --- | --- |
| POST | Create | The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server. |
| GET | Read | The GET method requests a representation of the specified resource. Requests using GET should only retrieve data. |
| PUT | Update | The PUT method replaces all current representations of the target resource with the request payload. |
| DELETE | Delete | The DELETE method deletes the specified resource. |

This allows for RESTful API services

# Retrofit

# What is Retrofit

- Type-safe HTTP client for Android and Java
- Networking library that turns your HTTP API into a Kotlin and Java interface
- 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

## Why Retrofit

- Provides base support for parsing common response types, such as XML and JSON
- Abstracts away a lot of extra code
- Asynchronously runs in a background
- Can convert JSON into predefined Kotlin objects

Some other common alternatives include HTTPUrlConnection, Volley, and more

# Adding Dependencies for Retrofit

## Add Internet permissions to Android Manifest

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

## Add Retrofit and GSON libraries to app level gradle

```
Implementation ‘com.squareup.retrofit2:retrofit:2.5.0’          // Retrofit
Implementation ‘com.squareup.retrofit2:convertor-gson:2.5.0’    // GSON
```

## Alternative project-Gradle Dependencies for Moshi

```
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"
```

# Creating an Interface

## Format

```
interface NameOfInterface{
      @REQUESTMETHOD(“api_endpoint”)
      fun FunctionThatDoesAPIRequest(): Call<returnType>
}
```

## Example

```
interface DestinationService {
      @GET(“destination”)
      fun getDestinationList(): Call<List<Destination>>
}
```

# Retrofit Object

```
val retrofit = Retrofit.Builder()
    .baseUrl("https://example.com/")    // Trailing slash is needed
    .addConverterFactory(GsonConverterFactory.create())
    .build()

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

## Example Interface

```
interface DestinationService {
      @GET(“destination”)
      fun getDestinationList(): Call<List<Destination>>
}
```

# Use Retrofit to Make a Request

```
service.getDestinationList().enqueue(object : Callback<List<Destination>> {
    override fun onResponse(call: Call<List<Destination>>, response: Response<List<Destination>>) {
        Log.d("SERVER_CONNECTION_SUCCESS", response.toString())
    }

    override fun onFailure(call: Call<List<Destination>>, t: Throwable) {
        Log.d("SERVER_CONNECTION_ERROR", t.toString())
    }
})
```

In onResponse, you can access the data using `response.body`
This will give you a List<Destination>

# More About Service Interfaces

# Retrofit Service Example 1

```
interface SimpleService {

    @GET("posts")
    suspend fun listAll(): 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
}
```

# Retrofit Service Example 2

```
interface MyApiEndpointInterface {
    // Request method and URL specified in the annotation
    @GET("users/{username}")
    fun getUser(@Path("username") username: String?): Call<User?>?

    @GET("group/{id}/users")
    fun groupList(@Path("id") groupId: Int, @Query("sort") sort: String?): Call<List<User?>?>?

    @POST("users/new")
    fun createUser(@Body user: User?): Call<User?>?
}
```

# Retrofit Service Example 3

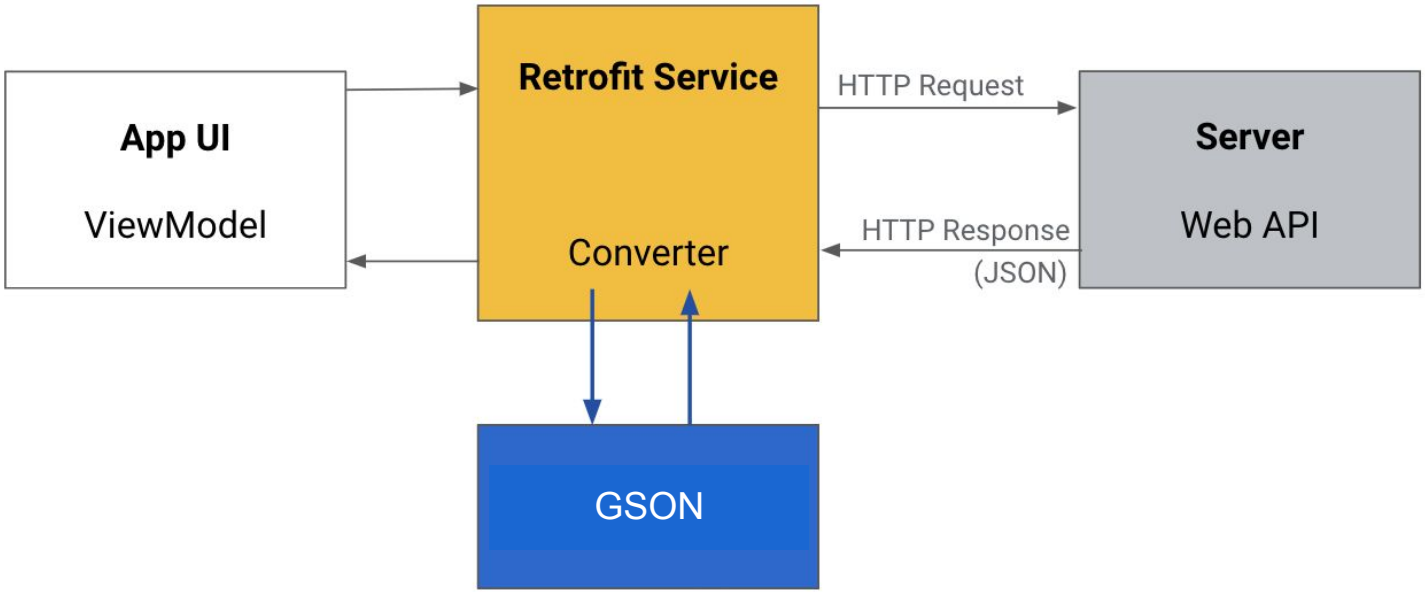
```
interface ShoppingService {
    @GET("shopping/all")
    fun getAll(): Call<List<Post>>

    @POST("shopping")
    fun create(@Body post: Post): Call<Message>

    @Multipart
    @POST("images/upload")
    fun uploadImage(@Part image: MultipartBody.Part): Call<ImageUploadResponse>
}
```

# Retrofit Service Annotation

| Annotation | Description |
| --- | --- |
| @Path | variable substitution for the API endpoint (i.e. username will be swapped for {username} in the URL endpoint). |
| @Query | specifies the query key name with the value of the annotated parameter. |
| @Body | payload for the POST call (serialized from a Java object to a JSON string) |
| @Header | specifies the header with the value of the annotated parameter |

# GSON Converter Factory

# Our Retrofit Architecture Overview



# Retrofit Class - Converter.Factory

## Purpose

Helps convert from a response type into class objects

## Example of Converters

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

## GSON / 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

## Kotlin Class Definition

```
@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 Sample

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

- Make sure member variables in the Kotlin class match the key's inf the JSON object
- JSON objects are saved with comma separated "key": value pairs

# Set up Retrofit and GSON

```
// Trailing slash is needed
public static final String BASE_URL = "http://api.myservice.com/";
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl(BASE_URL)
    .addConverterFactory(GsonConverterFactory.create())
    .build();
```

# Display Images

# Android Library - Glide

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

## Add gradle dependencies

```
dependencies {
  implementation 'com.github.bumptech.glide:glide:4.11.0'
}
```

# Using Glide to Display Images from the web

## Simple

```
GlideApp.with(context)
    .load("http://via.placeholder.com/300.png")
    .into(myImageView);
```

## Placeholder and error images

```
GlideApp.with(context)
    .load("http://via.placeholder.com/300.png")
    .placeholder(R.drawable.placeholder)
    .error(R.drawable.imagenotfound)
    .into(myImageView);
```

## Resizing images

```
GlideApp.with(context)
    .load("http://via.placeholder.com/300.png")
    .override(300, 200)
    .into(myImageView);
```

## Cropping images

```
GlideApp.with(context)
    .load("http://via.placeholder.com/300.png")
    .centerCrop()
    .into(myImageView);
```