Third Party Libraries

Using 3rd party libraries

- Add the name of the libraries as a new dependency in your app `build.gradle.kts`

```
dependencies {
  implementation("groupId:artifactId:version")
}
```

- Read the associated documentation

Sample Android Libraries

- Retrofit2: HTTP client built on-top of OkHttp
- OkHttp: “low-level” HTTP client
- Picasso (successor of Glide): Image Loading
- Room: SQLite object mapping
- Lottie: Animation
- Moshi: JSON
- ZXing: barcode image processing
- Espresso: testing
- Dagger2: Dependency Injection
- Hilt: Dependency Injection

Kotlin Specific Libraries

- Ktor: a framework for building async servers and clients
- Kotest: testing
- Exposed: lightweight SQL
- Kotlinx Serialization
- Koin: Dependency Injection
- KMongo: MongoDB
- Vaadin: UI libraries
- MockK: mocking/testing (*the name is **not** a typo it is Mock K with no space*)

OkHttp

- Simplify integration with web APIs
- Created by Square
- Repo: https://square.github.io/okhttp/

GSON

- Simplify working with JSON data representations
- Created by Google
- Repo: https://github.com/google/gson/

Lottie Animation

- Created by AirBnB Team
- JSON-based Animation
- Multi Platform Support
  - Android `<LottieAnimationView >`
  - iOS: `LottiView`

Lottie Animation: <LottieAnimationView>

Retrofit

- Declarative way to define HTTP requests and responses
  - Turn HTTP API into a Java Object
- Use Retrofit2 for Kotlin
- Created by Square
- Based on OkHttp
- Require Android API 21+
- Repo & Online Documentation: https://square.github.io/retrofit/

Accessing Data From Web Services

Prerequsites

- HTTP Protocol
  - GET requests (read/download data from server)
  - POST requests (write/upload data to server)
- URL Syntax
- JSON

Components of a URL

protocol

path

```
https://randomuser.me/api?results=2&inc=name,email
https://api.quotable.io/quotes/random?limit=2
https://swapi.dev/api/people/1
```

hostname

query parameters

key=value pairs

Quotable API

- Project Home: https://github.com/lukePeavey/quotable
- API Base URL: https://api.quotable.io
- Sample API endpoints

Description	URL
Random quote (≤ 100 character)	`http://api.quotable.io/random?maxLength=100`
Random quotes (2 quotes)	`http://api.quotable.io/quotes/random?limit=2`
Quotes of specific topics	`http://api.quotable.io/quotes?tags=science&limit=2`

Sample Web Service

`https://randomuser.me/api?results=2&inc=name,email`


```
{
    "results":[
      {
        "name":{
            "title":"Mrs",
            "first":"Laura",
            "last":"Candelaria"
        },
        "email":"laura.candelaria@example.com"
      },
      {
        "name":{
            "title":"Mrs",
            "first":"آدرینا",
            "Last":"جعفری"
        },
        "email":"adryn.jaafry@example.com"
      }
    ],
    "info": {
        "seed":"07898fb90c25ca1c",
        "results":2,
        "page":1,
        "Version":"1.4"
    }
}
```

JSON Basic Syntax

```
{ /* I'm an object */ }
```

```
{
  symbol: "C",
  name: "Carbon",
  weight: 12
}
```

```
[ /* I'm an array */ ]
```

```
["Carbon", "Fluor", "Helium"]
```


JSON Compound Syntax

```
/* Array within an object */
{
  "name": "Oxygen,
  "symbol": "O",
  "weight": 16,
  "orbitals": ["1s2", "2s2", "2p4"]
}
```

```
/* Array of objects */
[
  { "element": "Oxygen, "symbol": "O", "weight": 16},
  { "element": "Carbon, "symbol": "C", "weight": 12},
]
```

More Public APIs

https://github.com/public-apis/public-apis

Using Retrofit

Using Retrofit: Overview

1. Understand the data format of the response from the server
   a. Is it a (JSON) object, a (JSON) array
   b. Is it unstructured text?
2. Inspect the URL details
   a. Server Domain Name
   b. Which Port number?
   c. Which paths?
   d. Any query parameters?

Step 0: Add Library Dependencies

```
dependencies {
  // These two are mandatory
  implementation (“com.squareup.retrofit2:retrofit:x.y.z”)
  implementation (“com.squareup.retrofit2:converter-gson:x.y.z”)

  // Only when you need to debug the HTTP traffic
  implementation (“com.squareup.retrofit2:logging-interceptor:x.y.z”)

  // ViewModel
  implementation("androidx.activity:activity-ktx:1.8.2")
  implementation("androidx.lifecycle:lifecycle-livedata-ktx:2.7.0")
  implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:2.7.0")

  // Corroutines
  implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:1.7.1")
  implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.7.1")
}
```

Step 1: Add INTERNET permission

- Add INTERNET permission to AndroidManifest.xml

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

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

   <application>
     <!-- more contents here not shown -->
   </application>

</manifest>
```

- Make sure your Android Emulator is connected to Wifi!

Step 2: Map API Response To Data Class(es)

```
/* API Response */
{
  "results":[
    {
      "name":{
         "title":"___",
         "first":"___",
         "last":"___"
       },
       "email":"___"
      },
      “name”: {
         "title":"___",
         "first":"___",
         "last":"___"
      }
      “email”: “___”,
  ],
  "info": {
     "seed":"___",
     "results":2,
     "page":1,
     "Version":"___"
  }
}
```

results:
array of objects

info:
an ***object with 4 fields***

```
data class RandomNameResponse(
  val results: Array<Person>,
  val info: Any // We don't care
)
```

```
data class Person(
  val name:Name,
  val email:String)

data class Name(
  // title will be ignored
  val first: String,
  val last: String)
```

Step 3: Define Interface (@Query Example)

https://randomuser.me/api?inc=name,email&results=2

query params

```
interface RandomUserApi {
  @GET("api/?inc=name,email")
  suspend fun getNames(@Query("results") N: Int): Response<RandomNameResponse>

  @GET("api/")
  suspend fun getData(@Query(“inc”) filter: String,
                      @Query("results") N: Int):  Response<RandomNameResponse>
}
// the name in @Query() must match the key name in the URL query string
```

Live Server Response

```
// Defined in Step (2)
data class RandomNameResponse(
   val results: List<Person>,
   val info: Any // We don't care
)
```


Step 3: Define Interface (@Path Example)

```
https://swapi.dev/api/people/13
https://swapi.dev/api/starships/7
```

```
interface StarWarsApi {
  @GET("api/people/{who}")
  suspend fun getPeople(@Path("who") pid: Int): Response<PeopleResponse>

  @GET("api/starships/{vehicleId}")
  suspend fun getVehicle(@Path(“vehicleId”) vid: Int):  Response<VehicleResponse>
}
```

Live Server Response

Step 4a: Build the Client Object

```
object RandomUserClient {
   val BASE_URL = "https://randomuser.me/"

   fun getInstance(): Retrofit {
       return Retrofit.Builder().baseUrl(BASE_URL)
           .addConverterFactory(GsonConverterFactory.create())
           .build()
   }
}
```


Step 4b (Optional): Add Logging

```
object RandomUserClient {
   val BASE_URL = "https://randomuser.me/"
   val okHttpClientBuilder  = OkHttpClient.Builder()
   val logInterceptor = HttpLoggingInterceptor()

   fun getInstance(): Retrofit {
       logInterceptor.level = HttpLoggingInterceptor.Level.BASIC
       okHttpClientBuilder.addInterceptor(logInterceptor)

       return Retrofit.Builder().baseUrl(BASE_URL)
           .addConverterFactory(GsonConverterFactory.create())
           .client(okHttpClientBuilder.build())
           .build()
   }
}
```

Step 5a: Use the Client (in ViewModel)

```
class YourViewModel: ViewModel() {
   var apiEndpoint: RandomUserApi // interface declared in step 3

   init {
       // RandomUserClient is the object defined in Step 4
       apiEndpoint = RandomUserClient.getInstance().create(RandomUserApi::class.java)
   }

   fun getDataFromTheService(/* arg here */) {
       vieModelScope.launch(Dispatchers.IO) {
           // getRandomNames() is a function defined in Step 3
           val serverResponse = apiEndPoint.getRandomNames(27);
           serverResponse.body()?.let { data: RandomName
               println("Server returns random names ${data.results}")
           }
       }
   }
}
```


Step 5b: Handling HTTP Errors

```
class YourViewModel: ViewModel() {
   var apiEndpoint: RandomUserApi // interface declared in step 3

   init {
       apiEndpoint = RandomUserClient.getInstance().create(RandomUserApi::class.java)
   }

   fun getDataFromTheService(/* arg here */) {
       vieModelScope.launch(Dispatchers.IO) {
           val serverResponse = apiEndPoint.getRandomNames(27);
           if (serverResponse.isSuccessful()) {
               serverResponse.body()?.let { data: RandomName
               }
           } else {
               println("HTTP request failed with code: ${serverResponse.code()}")
           }
       }
   }
}
```

Typical HTTP Errors

Code	Description
1xx	Informational
	100: Continue 101: Switching Protocol
2xx	Success
	200: OK 201: Created 202: Accepted
3xx	Redirection
4xx	**Client Error (Mistakes in your Android App)**
	400: Bad Request 401: Unauthorized 404: Not Found
5xx	Server Error

Complete List

Variations of Retrofit Use Cases

- Field names in data class different from field names in server response
- HTTP Request methods
  - GET
  - POST & Message Body
- Data format from server
  - JSON object
  - JSON array
  - XML

Rename field names

```
/* API Response */
{
  "name":"Luke Skywalker",
  "height": 172,
  "hair_color": "blond",
  "films":[
    "https://swapi.dev/api/films/1/",
    "https://swapi.dev/api/films/2/",
  ],
  "vehicles":[
    "https://swapi.dev/api/vehicles/14/",
    "https://swapi.dev/api/vehicles/30/"
  ],
  "created":"2014-12-09T13:50:51.644000Z",
}
```

```
data class StarWarsCharacter(
  val name: String,
  val height: Int,
  @SerializedName(“hair_color”)
  val hairColor: String,
  val films: List<String>,
  val vehicles: List<String>,
  val created: String
)
```

```
// StarWars API: https://swapi.dev/api/people/1
interface StarwarsApi {
  @GET("api/people/{id}")
  suspend fun getRandomQuotes(@Path("id") id:Int): Response<StarwarsCharacter>>
}
```


Server response is an Array<T>

```
[
  {
    "_id": "sXDz-2N4-nnJ",
    "content": "An invasion of armies . . .",
    "author": "Victor Hugo",
    "tags": ["Famous Quotes"],
    "length": 75,
  },
  {
    "_id": "KmBzNBNUV-Tp",
    "content": "If you don't like something. . .",
    "author": "Maya Angelou",
    "tags": ["Change","Wisdom"],
    "length": 85,
  }
]
```

https://api.quotable.io/quotes/random?limit=2

```
data class Quote(
   @SerializedName(“_id”)
   val id: String,
   val content: String,
   val author: String,
   val tags: List<String>,
   val length: Int
)
```

```
interface QuotesApi {
  @GET("quotes/random")                    // You can use either List<> or Array<>
  suspend fun getRandomQuotes(@Query("limit") N:Int): Response<List<Quote>>
  //suspend fun getRandomQuotes(@Query("limit") N:Int): Response<Array<Quote>>
}
```

Adding HTTP Headers

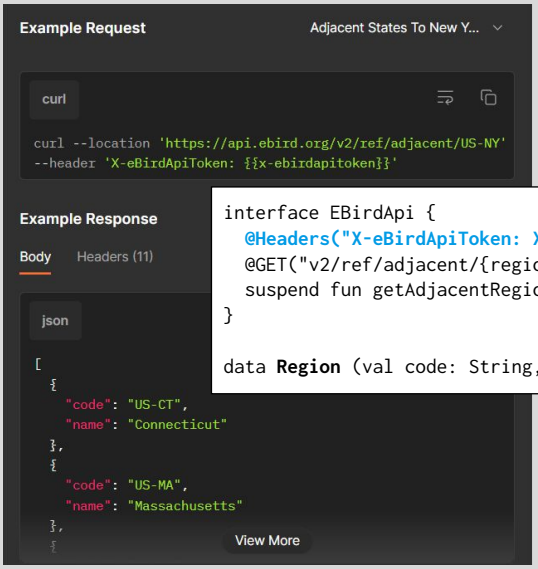
- Some Web services require you supply additional headers included in every HTTP request
- Common use cases of these headers
  - API key (specific code that verifies that the HTTP connections originate from a registered user, not from any random users)
  - Login Token (similar purpose as API key), may be used in conjunction with API key
  - Accounting Info how to charge the request

Adding HTTP Headers

```
interface NameOfYourInterface {
  @Headers(“ApiKey: AgdfduYYFdf23”)
  @GET("api/xxx/yyy/zzz/{id}")
  suspend fun getSomething(@Path("id") id:Int): Response<NameOfYourClass>>
}
```

```
interface NameOfYourInterface {
  @Headers({
     “ApiKey: AgdfduYYFdf23”,
     “X-Account: 7263GZ5”
  })
  @GET("api/xxx/yyy/zzz/{id}")
  suspend fun getSomething(@Path("id") id:Int): Response<NameOfYourClass>>
}
```

Header Example: http://api.ebird.org




```
interface EBirdApi {
  @Headers("X-eBirdApiToken: XXXXXXXXXXX")
  @GET("v2/ref/adjacent/{region}")
  suspend fun getAdjacentRegionsTo(@Path("region") regCode:String): Response<List<Region>>
}

data Region (val code: String, val name: String)
```


Using Ktor

Step 1a: Setup project build.gradle.kts

```
// Project's build.gradle.kts
buildscript {
   dependencies {
       classpath ("org.jetbrains.kotlin:kotlin-serialization:1.9.22")
   }
}
```


Step 1b: Setup app build.gradle.kts

```
// Module/App build.gradle.kts
plugin {
  id("kotlinx-serialization")
}

dependencies {
  // HTTP client engine that handle network requests
  implementation("io.ktor:ktor-client-android:1.5.0")
  // Handle data (de)serialization
  implementation("io.ktor:ktor-client-serialization:1.5.0")
  implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.0.1")
  // Handle HTTP request logging (useful for debugging)
  implementation("io.ktor:ktor-client-logging-jvm:1.5.0")
}
```

Step 2: Define Data Classes

```
/* API Response */
{
  "results":[
    {
      "name":{
         "title":"___",
         "first":"___",
         "last":"___"
       },
       "email":"___"
      },
      /* more data */
  ],
  "info": {
     "seed":"___",
     "results":2,
     "page":1,
     "version":"___"
  }
}
```

```
import kotlinx.serialization.Serializable

@Serializable
data class RandomNameResponse(
   val results: List<Person>,
   // info will be ignored
)

@Serializable
data class Name(
  // title will be ignored
  val first: String,
  val last: String)

@Serializable
data class Person(val name  val email:String)
```


Step 3a: Create HTTP Client (Minimal Setup)

```
import io.ktor.client.HttpClient
import io.ktor.client.engine.android.Android
import io.ktor.client.features.DefaultRequest
import io.ktor.client.features.json.JsonFeature
import io.ktor.client.features.json.serializer.KotlinxSerializer
import kotlinx.serialization.json.Json
import io.ktor.http.ContentType
import io.ktor.http.HttpHeaders

val userClient = HttpClient(Android) {
   install(JsonFeature) {
       serializer = KotlinxSerializer(Json {
           ignoreUnknownKeys = true
       })
   }

   install(DefaultRequest) {
       header(HttpHeaders.ContentType, ContentType.Application.Json)
   }
}
```

Step 3b: Create HTTP Client (With Logging)

```
import io.ktor.client.*
import io.ktor.client.features.logging.*
import kotlinx.serialization.json.Json
import io.ktor.http.*

val userClient = HttpClient(Android) {
   install(JsonFeature) {
       serializer = KotlinxSerializer(Json {
           ignoreUnknownKeys = true
       })
   }
   install(Logging) {
       logger = object: Logger {
           override fun log(message: String) {
               println("Logger Ktor => ", message)
           }
       }
   }
   install(DefaultRequest) {
       header(HttpHeaders.ContentType, ContentType.Application.Json)
   }
}
```


Step 4: Create & Use Client Object

```
// MyClient.kt
object MyApiClient {
  suspend fun getRandomNames(total: Int): RandomNameResponse =
    userClient.get("https://randomuser.me/api/?inc=name,email&results=${total}")
    // userClient is defined in Step 3
}
```

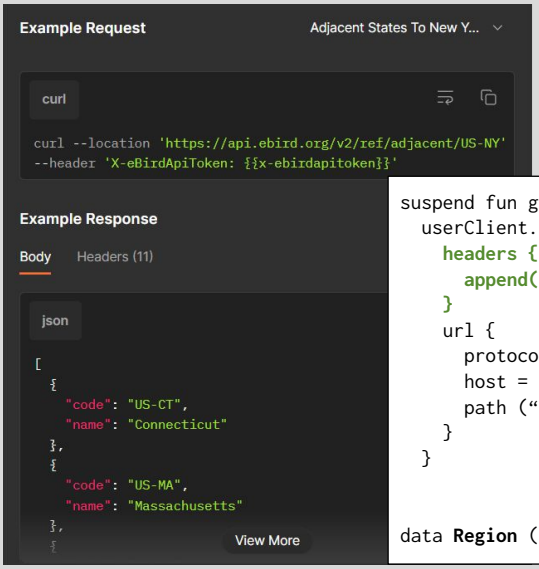
```
class MyViewModel: ViewModel() {
  fun fetchNames(N: Int) {
    viewModelScope.launch(Dispatchers.IO) {
       val response = MyApiClient.getRandomNames(N);
       for (n in response.results) {
         println("Random name $n")
       }
    }
  }
}
```

Alternate Syntax for Sending HTTP Request

```
// Option 1: Pass everything as a LONG string
suspend fun getRandomNames(total: Int): RandomNameResponse =
  userClient.get("https://randomuser.me/api/?inc=name,email&results=${total}")
```

```
// Option 2: Use lambda expressions
//           and break up the request details into various elements
suspend fun getRandomNames(total: Int): RandomNameResponse =
  userClient.request {
    url {
      protocol = URLProtocol.HTTPS
      host = "randomuser.me"
      path ("api")
      parameters.append("inc", "name,email")
      parameters.append("results", total.toString())
    }
  }
```


KTor Header: http://api.ebird.org




```
suspend fun getAdjacentRegionsTo(regionCode:String): List<Region> =
  userClient.request {
    headers {
      append("X-eBirdApiToken", "XXXXXXXXXX")
    }
    url {
      protocol = URLProtocol.HTTPS
      host = "api.ebird.org"
      path ("v2/ref/adjacent/${regionCode}")
    }
  }


data Region (val code: String, val name: String)
```

Reading Assignment

Engelsma/Dulimarta textbook

- Chapters 1.2, 1.5
- Chapters 2.2, 2.4