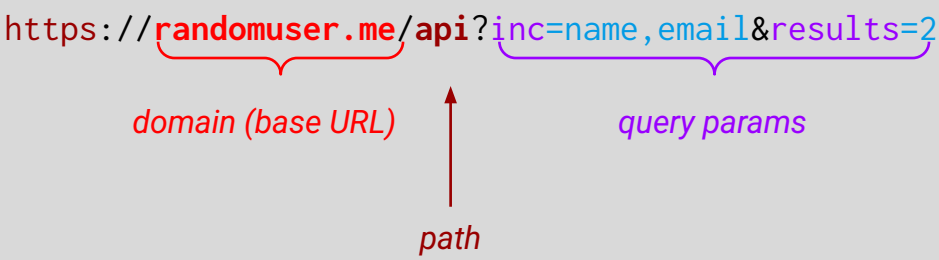
Accessing Web Services
KTor (Android) & URLSession (iOS)

Accessing Web Services: 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?
3. JSON Formatter Extension for your Web Browser

Parts of URL



Live Server Response

Step 1a: Setup Top-Level build.gradle.kts

```
// Project's build.gradle.kts
buildscript {
   dependencies {
       // The version number below should be compatible with Kotlin version
       id ("org.jetbrains.kotlin.plugin.serialization") version "2.3.0" apply false
   }
}
```

Step 1b: Setup in app build.gradle.kts

```
// Module/App build.gradle.kts
plugin {
  id("org.jetbrains.kotlin.plugin.serialization")
}

dependencies {
  implementation("io.ktor:ktor-client-core:3.4.1")
  // HTTP client engine that handle network requests
  implementation("io.ktor:ktor-client-android:3.4.1")
  // Handle data (de)serialization
  implementation("io.ktor:ktor-client-serialization:3.4.1")
  implementation("io.ktor:ktor-client-content-negotiation:3.4.1")
  implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.7.3")
  // Handle HTTP request logging (useful for debugging)
  implementation("io.ktor:ktor-client-logging:3.4.1")
}
```


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

```
/* API Response (Object) */
{
  "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***

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

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

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


Step 3: Create HTTP Client (Minimal Setup)

```
class MyViewModel: ViewModel() {
    private val userClient = HttpClient() {
        install(ContentNegotiation) {
            json(Json {
                ignoreUnknownKeys = true // Enable this option to ignore fields
                                         // which are not defined in your data class
            })
        }
        install(Logging) {
            level = LogLevel.ALL         // Or: .HEADERS, .BODY, .NONE
        }
    }
}
```

Step 4: Create & Use Client Object

```
class MyViewModel: ViewModel() {
    private val userClient = ______ // defined in Step 3

    fun fetchNames(N: Int) {
        viewModelScope.launch(Dispatchers.IO) {
            val response = userClient.get(“https://randomuser.me/api/?limits=${N}”);
        }
    }
    //***** Alternative GET call syntax
    //  val response = userClient.get {
    //      url {
    //          protocol = URLProtocol.HTTPS
    //          host = “randomuser.me”
    //          path(“/api”)
    //          parameters.append(“limits”, N.toString());
    //      }
    //  }
}
```


Step 5a: Parse the response

```
@Serializable
data class RandomNameResponse(
  val results: List<Person>
)
```

```
class MyViewModel: ViewModel() {
    private val userClient = ______ // defined in Step 3
    private val _users = MutableStateFlow<List<Person>>(emptyList())

    fun fetchNames(N: Int) {
        viewModelScope.launch(Dispachers.IO) {
            val response = userClient.get { /* details omitted here */ }

            // For debugging purpose, you can parse the entire response as a string
            val body_text = response.body<String>()
            println(body_text)
            // For actual production, parse the response as your data class
            val randUsr: RandomNameResponse = response.body<RandomNameResponse>()
            _user.update { randUsr.results }
        }
    }
}
```

Parse Array Response

```
/* API Response (Array of objects) */
[
    {
        "picture": {
            "large":"http://___",
            "thumbnail":"http://___",
         },
        "description":"___"
    },
    {
        "picture": {
            "large":"http://___",
            "thumbnail":"http://___",
         },
        "description":"___"
    }
]
```

```
@Serializable
data class Artwork (
  val picture: Picture,
  val description: String
)

@Serializable
data class Picture(
  // URL to the actual image
  val large: String,
  val thumbnail: String
)
```

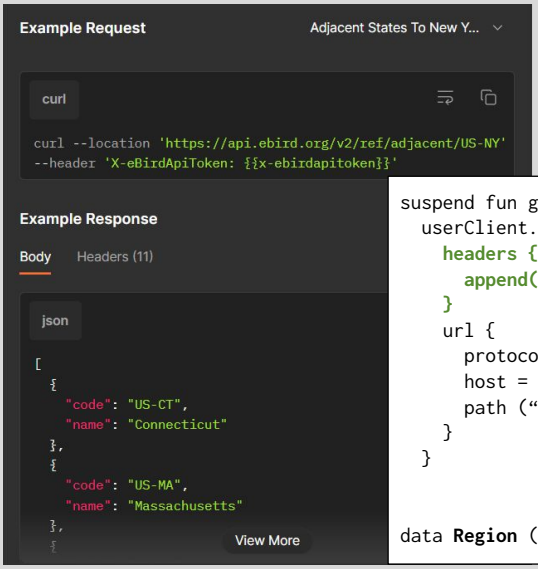

Step 5b: Parse Array Response

```
class MyViewModel: ViewModel() {
    private val userClient = ______ // defined in Step 3
    private val _artsy = MutableStateFlow<List<Artwork>>(emptyList())

    fun fetchNames(N: Int) {
        viewModelScope.launch(Dispachers.IO) {
            val response = userClient.get { /* details omitted here */ }

            // For actual production, parse the response as your data class
            val arr: List<Artwork> = response.body<List<Artwork>>()
            _artsy.update { arr }
        }
    }
}
```

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)
```


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

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>>
}
```


Web Client in iOS

Kotlin @Serializable ⇒ Swift Decodable

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

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

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

```swift
// Swift struct equivalent

struct RandomNameResponse: Decodable {
  let results: Array<Person>
}


struct Person: Decodable {
    let name: Name
    let email: String
}

struct Name: Decodable {
    let first: String
    let last: String
}
```


Parse JSON response (iOS)

```swift
struct RandomNameResponse: Decodable
{
  let results: Array<Person>
{
```

```kotlin
class MyViewModel: ViewModel() {
    fun fetchNames(N: Int) {
        viewModelScope.launch(Dispachers.IO) {
            val response = userClient.get { /* details omitted here */ }
            val randUsr: RandomNameResponse = response.body<RandomNameResponse>()
        }
    }
}
```

```swift
class MyViewModel: ObservableObject {
  @Published var users = Array<Person>()

  func fetchNames(N: Int) {
    let url = URL("https://randomuser.me/api?results=3)
    Task {
      let (rawData, rawResp) = try! await URLSession.shared.data(from: url!)
      guard let response = rawResp as? HTTPURLResponse else { return }
      debugPrint(response.statusCode)
      let decodedData = try! JSONDecoder().decode(RandomNameResponse.self, from: rawData)
      users = decodedData.results
    }
  }
}
```