Apunts

Descarregar aquests apunts en pdf y html

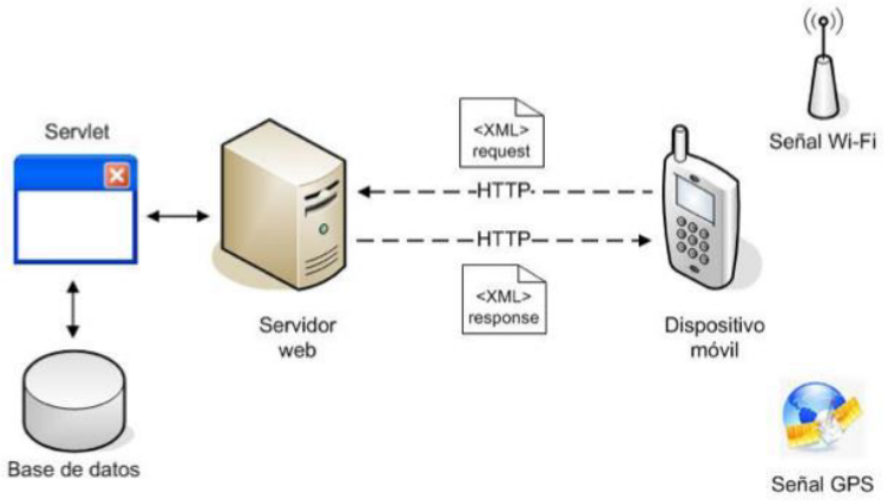
Tema 11.2. Persistència de Dades II Retrofit

Índex

Accés a bases de dades remotes

Introducció

Per a obtindre i guardar informació d'una base de dades remota, és necessari connectar-se a un servidor on es trobarà la BBDD. L'esquema de funcionament el podem veure en la següent imatge:



En el servidor funcionen en realitat tres components bàsics:

- Una base de dades, que emmagatzema tota la informació dels usuaris. Per a la BBDD es pot utilitzar MySQL, que és de llicència gratuïta.

- Un servlet, que atén la petició rebuda, la processa i envia la resposta corresponent. Un servlet no és més que un component Java, generalment xicotet i independent de la plataforma.
- Un servidor web , on resideix i s'executa el servlet , i que roman a l'espera de connexions HTTP entrants.

Els formats més utilitzats per a compartir informació mitjançant aquests serveis web són XML (i altres derivats) i JSON.

Què és JSON?

JSON (JavaScript Objec t Notation) és un format lleuger d'intercanvi de dades entre clients i servidors, basat en la sintaxi de JavaScript per a representar estructures en forma organitzada. És un format en text pla independent de tot llenguatge de programació, és més, suporta l'intercanvi de dades en gran varietat de llenguatges de programació com PHP Python C++ C# Java i Ruby.

XML també pot usar-se per a l'intercanvi, però pel fet que la seua definició genera un DOM , l'analitze es torna extens i pesat. A més d'això XML ha d'usar Xpath per a especificar rutes d'elements i atributs, per la qual cosa demora la reconstrucció de la petició. En canvi JSON no requereix restriccions addicionals, simplement s'obté el text pla i el engine de JavaScript en els navegadors fa el treball de parsing sense cap complicació.

Tipus de dades en JSON

Similar a l'estructuració de dades primitives i complexos en els llenguatges de programació, JSON estableix diversos tipus de dades: cadenes, números, booleans, arrays, objectes i valors null. El propòsit és crear objectes que continguen diversos atributs compostos com a parells clau valor. On la clau és un nom que identifique l'ús del valor que l'acompanya. Vegem un exemple:

```
{
   "id": 101,
   "Nombre": "Carlos",
   "EstaActivo": true,
   "Notas": [2.3, 4.3, 5.0]
}
```

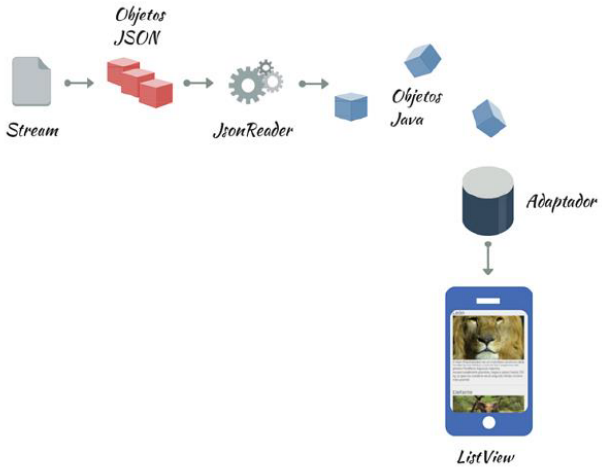
L'anterior estructura és un objecte JSON compost per les dades d'un estudiant. Els objectes JSON contenen els seus atributs entre claus `{}` , igual que un bloc de codi en JavaScript, on cada atribut ha d'anar separat per coma `,` per a diferenciar cada parell.

La sintaxi dels parells ha de contindre dos punts `:` per a dividir la clau del valor. El nom del parell ha de tractar-se com a cadena i afegir-li cometes dobles.
Si notes, aquest exemple porta un exemple de cada tipus de dada:

- El *id* és de tipus sencer, ja que conté un número que representa el codi de l'estudiant.
- El *Nom* és un string. Usa cometes dobles per a identificar-les.
- *EstaActivo* és un tipus booleà que representa si l'estudiant es troba en la institució educativa o no. Usa les paraules reservades true i false per a declarar el valor.
- *Notes* és un arranjament de nombres reals. El conjunt dels seus elements has d'incloure'ls dins de claudàtors `[ ]` i separar-los per coma.

La idea és crear un mecanisme que permeta rebre la informació que conté la base de dades externa en format JSON cap a l'aplicació. Amb això s'analitzarà cada element i serà interpretat en forma d'objecte Java per a integrar correctament l'aspecte en la interfície d'usuari.

La classe **JsonObject** de la llibreria **org.json.JSONObject**, pot interpretar dades amb format JSON i analitzar-los a objectes Java o al revés.
Vegem una il·lustració que mostra el procés d'analitze que serà estudiat:



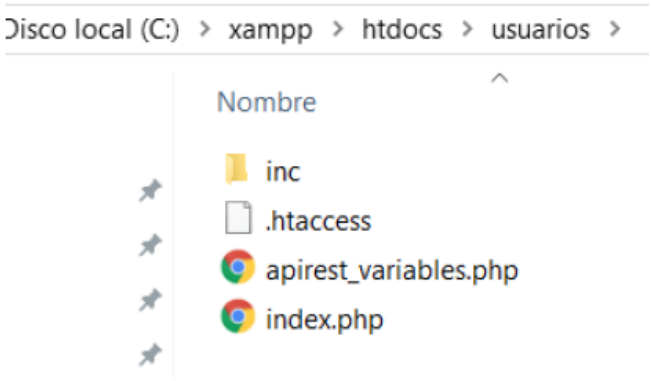
- Com pots observar l'origen de les dades és un servidor extern o hosting que hages contractat com a proveïdor per als teus serveis web. L'aplicació web que realitza la gestió d'encriptació de les dades a format `JSON` pot ser `PHP`, `JavaScript`, `ASP.NET`, etc..
- La teua aplicació Android a través d'un client realitza una petició a la direcció URL del recurs amb la finalitat d'obtindre les dades. Aqueix flux entrant ha d'interpretar-se amb ajuda d'un parser personalitzat que implementaren les classes que s'utilitzen per a treballar amb `JSON`.
- El resultat final és un conjunt de dades adaptable al `API` d'Android. Depenent de les teues necessitats, pots convertir-los en una llista d'objectes estructurats que alimenten un adaptador que poble un ListView o simplement actualitzar la base de dades local de la teua aplicació en `SQLite`.

Accés a bases de dades amb Apirest i Retrofit

Retrofit és un client *REST* per a Android i Java, desenvolupat per Square, molt simple i fàcil d'aprendre. Permet fer peticions `GET`, `POST`, `PUT`, `PATCH`, `DELETE` i `HEAD`; gestionar diferents tipus de paràmetres i analitzar automàticament la resposta a un POJO (Plain Old Java Object). Utilitzarem aquesta llibreria per la seua facilitat de maneig.

Definició del Servei Rest

Com és de suposar, per a poder accedir a un servei Rest des d'una de les nostres aplicacions, aquest ha d'haver sigut creat amb anterioritat i allotjat en un Hosting adequat.
La part de creació ja siga amb PHP, Java, o qualsevol dels altres llenguatges que ho permeten no correspon a la nostra assignatura, per la qual cosa la passarem per alt i usarem un ApiRest general que s'ha proporcionat en el mòdul d'Accés a Dades i que amb unes xicotetes modificacions és valgut per a la majoria de casos. La carpeta amb el ApiRest està composta dels següents arxius:



La configuració del nostre API consta de dos arxius:

1. `.htaccess` En aquest arxiu es configuren les regles d'accés, sobre una ruta que li indiquem en RewriteBase (ací és on haurem d'afegir la carpeta en la qual haurem allotjat nostra API, començant i acabant amb `/`).

```
RewriteEngine On
RewriteBase "/uaurios/"
RewriteRule ^([a-zA-Z0-9_-]*)$ index.php?action=$1&%{QUERY_STRING}
RewriteRule ^([a-zA-Z0-9_-]*)/([a-zA-Z0-9_-]*)$ index.php?action=$1&value=$2&%{QUERY_STRI
RewriteRule ^([a-zA-Z0-9_-]*)/([a-zA-Z0-9_-]*)/([a-zA-Z0-9_-]*) index.php?action=$1&field
```

2. `apirest_variables.php`, en aquest arxiu es defineixen les dades de connexió a la base de dades. s'indiquen les taules que té aquesta i el nom de l'identificador de cadascuna d'elles.

```php
<?php
    // CONFIGURACIÓN BASE DE DATOS MYSQL
    $servername = "127.0.0.1";
    $username = "root";
    $password = "";

    // BASE DE DATOS
    $dbname = "uruariosmensajes";

    // TABLAS Y SU CLAVE
    $tablas = array();
    $tablas["mensajes"]="_id";
    $tablas["usuarios"]="_id";
```

La resta d'arxius del ApiREst no hauran de modificar-se, ja que està construïda de manera genèrica amb les necessitats més comunes per a aquests casos. Haurem de crear la BD i allotjar el ApiRest de manera local o en el núvol, usant els coneixements que es tenen del mòdul d'Accés a Dades.

Suposarem un exemple molt senzill d'una base de dades Usuaris en el qual tindrem solament una taula Usuaris amb dos camps de tipus String (nick i nom). La BD l'haurem construïda en el servidor amb antelació (en aquest cas amb taula usuaris de tres camps, nick i nom de tipus cadena i _aneu de tipus numèric autoincrementable com a clau). També crearem una aplicació amb dos camps de text que ens permeta inserir les dades de l'usuari i un botó flotant d'afegir, com es veu en la imatge següent:

Consum d'un Servei Rest des d'Android

Existeixen diferents llibreries que ens permetrien consumir els serveis des de l'App d'Android, però donada la seua facilitat utilitzarem les llibreries: **Retrofit2** i **Gson**.

Retrofit la utilitzarem per a fer peticions i processar les respostes del APIRest, mentre que amb Gson transformarem les dades de JSON als propis que utilitze l'aplicació.

Per a això afegirem les següents línies en el build.gradle de l'app, i no oblides incloure permisos d'internet:

```
implementation "com.squareup.retrofit2:converter-gson:2.9.0"
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
```

Podem dir que els passos a seguir seran els següents:

1. Creació d'un builder de Retrofit. Per a poder utilitzar Retrofit necessites crear un builder que et servirà per a comunicar-te amb la API triada. En la imatge s'ha utilitzat un mètode per a encapsular el codi, encara que no és necessari.

```
private fun crearRetrofit(): ProveedorSersvicio {
    //val url = "http://10.0.2.2/usuarios/" //para el AVD de android
    val url="http://xusa.iesdoctorbalmis.info/usuarios/" //para servidor del
    val retrofit = Retrofit.Builder()
            .baseUrl(url)
            .addConverterFactory(GsonConverterFactory.create())
            .build()
    return retrofit.create(ProveedorSersvicio::class.java)
}
```

   Si et fixes en la segona línia veuràs que cal escriure la url base de la API a la qual faràs les peticions. L'obtens de la documentació proporcionada pel creador de la API.
   En la següent línia cal indicar la manera de convertir els objectes de la API als de la teua aplicació i viceversa, ací és on específiques que utilitzaràs Gson.
   Aquest builder ho necessitaràs per a fer les anomenades a la API, així que procura que siga accessible.
2. Creació de les classes Pojo que li servirà al Gson per a analitzar els resultats. Però primer necessites conéixer l'estructura del Json que et retornarà la API, ja que no hi ha un patró establit. Per a conéixer l'estructura prèviament, es pot utilitzar PostMan i realitzar les diferents peticions (GET, POST, PUT, etc) des d'aquest.
   Existeixen aplicacions o webs que faciliten la creació de la classe Pojo a partir d'un JSon, com per exemple: [http://www.jsonschema2pojo.org/]

```
class RespuestaJSon {
    var respuesta = 0
    var metodo: String? = null
    var tabla: String? = null
    var mensaje: String? = null
    var sqlQuery: String? = null
    var sqlError: String? = null
}
```

   Després has de crear l'estructura de classes per a emmagatzemar la informació que et resulte útil. Classe Usuari en aquest exemple.

```
class Usuarios(var nick: String, var nombre: String, var _id: Int =0)
```

3. Un altre element imprescindible, és la gestió dels serveis que es vulguen utilitzar. Per a cadascun d'ells s'haurà de fer una petició a la API. Necessitarem crear una interfície amb tots els serveis que vulgues utilitzar. Ací tens uns exemples:

```
interface ProveedorSersvicio {
    @GET("usuarios")
    @Headers("Accept: application/json", "Content-Type: application/json")
    suspend fun usuarios(): Response<List<Usuarios>>

    @GET("mensajes")
    @Headers("Accept: application/json", "Content-Type: application/json")
    suspend fun mensajes():Response<List<Mensaje>>

    @GET("mensajes/{nick}")
    @Headers("Accept: application/json", "Content-Type: application/json")
    suspend fun getUsuario(@Path("nick") nick: String): Response<List<Usuari

    @POST("usuarios")
    @Headers("Accept: application/json", "Content-Type: application/json")
    suspend fun insertarUsuario(@Body usuarios: Usuarios): Response<Respuest

    @POST("mensajes")
    @Headers("Accept: application/json", "Content-Type: application/json")
    suspend fun insertarMensaje(@Body mensaje: Mensaje): Response<RespuestaJ
}
```

En el codi de dalt hi ha dos serveis dos són de tipus `GET` sense paraula clau i que serviran per a obtindre tots els usuaris i tots els missatges. Després es té un altre servei `GET` amb paraula clau (Nick) que servirà per a seleccionar els missatges d'un determinat Nick. I després dos `POST` als quals se'ls passa en el Bodi les dades a afegir.
Usant `Corrutinas` es realitzarà l'anomenada a la API, per la qual cosa els mètodes de la interfície han de ser de tipus **suspend**. En cada mètode s'indica el tipus de dada que espera obtindre, en aquests exemples la resposta pot ser o una llista del tipus d'objecte de la petició o un tipus *RespuestaJSon, que tindrà tots els possibles membres que ens podria donar alguna de les crides invocades.
Les paraules seguides del símbol `@` donen funcionalitat als serveis, fent-los dinàmics i reutilitzables, per a més informació [http://square.github.io/retrofit/]

> 💡 Les que gestiona **Retrofit** per a invocar la url de la petició són `@GET, @POST, @PUT i @DELETE` . El paràmetre correspondrà amb la url de la petició.
> En el builder de Retrofit s'inclou la url base del api acabada amb `/` , que unida amb el paràmetre del servei crearan la url completa de la petició:
> `@GET -> http://10.0.2.2/usuaris/usuari`

- **`@Header i @Headers`** . S'usen per a especificar els valors que van en la secció *header* de la petició, com per exemple en què format seran enviats i rebuts les dades.
- **`@Path`** . Serveix per a incloure un identificador en la url de la petició, per a obtindre informació sobre una cosa específica. L'atribut en el mètode de crida que siga precedit per *@Path*, substituirà a l'identificador entre claus de la ruta que tinga el mateix nom.
- **`@Fields`** . Ens permet passar variables bàsiques en les peticions *Post i Put*.
- **`@Bodi`** . És equivalent a Fields però per a variables objecte.
- **`@*Query`** . S'usa quan la petició necessitarà paràmetres (els valors que van després del `?` en una url).

4. Demanar dades a la Api seria l'últim passe a realitzar. Retrofit ens dona l'opció de realitzar-ho de manera síncrona o asíncrona. Aprofitarem els nostres coneixements de *corrutinas per a llançar la petició en segon pla de manera senzilla.

```
1  private fun anyadirUsuario(usuarios: Usuarios) {
2          val context=this
3          var salida:String?
4          val proveedorServicios: ProveedorSersvicio = crearRetrofit()
5
6          CoroutineScope(Dispatchers.IO).launch {
7              val response = proveedorServicios.insertarUsuario(usuarios
8              if (response.isSuccessful) {
9                  val usuariosResponse = response.body()
10                 if (usuariosResponse != null) salida = usuariosRespons
11                 else salida=response.message()            }
12             else {
13                 Log.e("Error", response.errorBody().toString())
14                 salida=response.errorBody().toString()
15             }
16             withContext(Dispatchers.Main) {
17                 Toast.makeText(context, salida, Toast.LENGTH_LONG).sho
18                 if (espera != null) espera?.hide()
19                 limpiaControl()
20             }
21         }
22     }
```

📌 **Nota: Línia 4** cridem al mètode crearRetrofit, que és el que ens retorna un proveïdor de serveis del tipus de interface que hem creat i ens enllaçarà amb la url del servidor i amb el GSon. **Línies 5 - 13** es llança la corrutina amb la petició desitjada, quan s'obtinga resposta es filtra per a veure si ha

> sigut correcta i recuperar les dades necessàries, `response.body()` , en cas contrari es llança missatge d'error.

EjercicioPropuestoBDExternas

EjercicioPropuestoAgenda