



Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh  
**TRUNG TÂM TIN HỌC**

# Lập trình Android

## Bài 20: *Web Services*

Phòng LT & Mạng

<http://csc.edu.vn/lap-trinh-va-csdl>





# Nội dung

---

1. Network Permissions
2. Cleartext HTTP requests
3. Dữ liệu JSON
4. Thư viện Gson
5. Xây dựng thư viện REST API Web Service với Python
6. Thư viện Volley
7. Thư viện Retrofit
8. Thư viện Glide
9. Thư viện Picasso



# Permissions

---

- Để truy cập **Internet**, cần khai báo các **permissions** cần thiết trong file **AndroidManifest.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.csc.networking">

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

</manifest>
```



# Cleartext HTTP requests

- Các **HTTP** request (**http://**) không còn được cho phép từ Android 9 (P)
- Chỉ các **HTTPS** request (**https://**) được cho phép
- Để có thể sử dụng các **HTTP** request, cần khai báo thuộc tính **useCleartextTraffic = true** trong file **AndroidManifest.xml**

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportRtl="true"
    android:theme="@style/AppTheme"
    android:usesCleartextTraffic="true">

</application>
```



# Dữ liệu JSON

---

- **JSON**: JavaScript Object Notation
- **JSON** là cú pháp dùng để lưu trữ và trao đổi dữ liệu
- **JSON** có dạng văn bản (text)
- Ví dụ:

```
{  
  "name": "John",  
  "age": 30,  
  "city": "New York"  
}
```



# Cú pháp JSON

---

- Cú pháp **JSON**
  - Sử dụng cặp name-value để chứa 1 dữ liệu
  - Sử dụng dấu phẩy , để phân tách các dữ liệu
  - Sử dụng cặp dấu ngoặc nhọn {} để giữ một đối tượng (object)
  - Sử dụng cặp dấu ngoặc vuông [] để giữ một mảng (array)
- Kiểu dữ liệu
  - Kiểu chuỗi (string): { "name": "John" }
  - Kiểu số (number): { "age": 30 }
  - Kiểu luận lý (Boolean): { "sale": true }
  - Kiểu đối tượng (object): { "employee": { "name": "John", "age": 30, "city": "New York" } }
  - Kiểu mảng (array): { "employees": [ "John", "Anna", "Peter" ] }
  - Kiểu null: { "middlename": null }



# Thư viện Gson

---

- Thư viện **Google Gson** cung cấp khả năng chuyển đổi giữa chuỗi **JSON** và đối tượng **Java**
- Cấu hình **Gradle** để sử dụng **Gson**:

```
dependencies {  
    implementation 'com.google.code.gson:gson:2.8.5'  
}
```

- Để chuyển đổi giữa chuỗi **JSON** và đối tượng Java, cần xây dựng Model class phù hợp với dữ liệu JSON



# Xây dựng Model class phù hợp với dữ liệu JSON

## Dữ liệu JSON

```
{
  "movies": [
    {
      "id": "771305050",
      "title": "Game of Throne",
      "production": {
        "director": "Alan Taylor",
        "screenplay": "George Martin"
      },
      "year": 2015
    },
    {
      "id": "771357161",
      "title": "Mission Impossible Fallout",
      "production": {
        "director": "Christopher McQuarrie",
        "screenplay": "Christopher McQuarrie"
      },
      "year": 2015
    }
  ]
}
```

## Model class

```
public class Production {
    String director;
    String screenplay;
}
```

```
public class Movie {
    String id;
    String title;
    int year;
    Production production;
}
```

```
public class BoxOfficeMovie {
    List<Movie> movies;

    public BoxOfficeMovie() {
        movies = new ArrayList<Movie>();
    }
}
```





# Phân tách dữ liệu JSON

- Giả sử có 1 chuỗi dữ liệu **JSON**, có thể phân tách dữ liệu **JSON** bằng phương thức **fromJson** của đối tượng **Gson**
- Phương thức **fromJson** có 2 tham số:
  - **Chuỗi JSON** cần phân tách
  - **Model class** tương ứng

```
public class BoxOfficeMovie {  
    List<Movie> movies;  
  
    public BoxOfficeMovie() {  
        movies = new ArrayList<Movie>();  
    }  
  
    public static BoxOfficeMovie parseJSON(String response) {  
        Gson gson = new GsonBuilder().create();  
        BoxOfficeMovie boxOfficeMovie = gson.fromJson(response, BoxOfficeMovie.class);  
        return boxOfficeMovie;  
    }  
}
```



# So khớp tên biến và tên từ khoá JSON

- Để **Gson** có thể phân tách được dữ liệu **JSON**, các tên biến của các **Model** class cần đặt tên trùng với tên các từ khoá trong dữ liệu **JSON**
- Trường hợp tên biến không trùng với từ khoá, cần khai báo tên từ khoá tương ứng với tên biến bằng annotation **@SerializedName**

```
public class BoxOfficeMovie {  
    @SerializedName("movies")  
    List<Movie> moviesList;  
}
```



# Xây dựng thư viện REST API Web Service với Python

---

- **Python** là một ngôn ngữ lập trình bậc cao cho các mục đích lập trình đa năng, do Guido van Rossum tạo ra và lần đầu ra mắt vào năm 1991. Python được thiết kế với ưu điểm mạnh là dễ đọc, dễ học và dễ nhớ. Python là ngôn ngữ có hình thức rất sáng sủa, cấu trúc rõ ràng, thuận tiện cho người mới học lập trình. (Wikipedia)
- Cài đặt **Python**:
  - Download tại: <https://www.python.org/downloads>
  - Công cụ soạn thảo: **VSCode** - <https://code.visualstudio.com>
- Minh họa sau, sử dụng **Python 3** và **Flask** framework để xây dựng các **REST API Web Services** để quản lý các sản phẩm của một cửa hàng



# Xây dựng REST API với Flask framework

---

- **Bước 1:** Tạo project **Python** mới với môi trường ảo **venv** theo các lệnh sau:
  - Tạo project mới: **python3 -m venv store** # store là tên project
  - Khởi động môi trường ảo:
    - Trên Windows: **store\Scripts\activate.bat**
    - Trên Unix hoặc Mac: **source store/bin/activate**
- **Bước 2:** Cài đặt các thư viện cần thiết
  - Thư viện **Flask**: **pip install flask**
  - Thư viện **Flask-SQLAlchemy**: **pip install flask-sqlalchemy**
- **Bước 3:** Tạo file **app.py** là đầu vào chương trình, cũng là nơi chứa các xử lý chính
- **Bước 4:** Khởi chạy **app.py** bằng lệnh “**python app.py**” và thực hiện test với URL **http://localhost:5000**



# Xây dựng app.py

---

- Khai báo các thư viện cần thiết

```
from flask import Flask, request, jsonify, abort
from flask_sqlalchemy import SQLAlchemy
```

- Cấu hình và khai báo CSDL SQLite

```
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///products.sqlite3'
db = SQLAlchemy(app)
```

- Định nghĩa Model class (Product)

```
class Product(db.Model):
    id = db.Column('id', db.Integer, primary_key = True)
    name = db.Column(db.String(255))
    price = db.Column(db.Float(asdecimal = True))
    def __init__(self, name, price):
        self.name = name
        self.price = price
```



# Xây dựng app.py (2)

---

- Định nghĩa API endpoint trả về danh sách các products

```
@app.route('/products')
def get_products():
    prods = Product.query.all()
    response = []
    for prod in prods:
        item = {
            'id': prod.id,
            'name': prod.name,
            'price': str(prod.price)
        }
        response.append(item)
    return jsonify({
        'products': response
    })
```



# Xây dựng app.py (3)

- Định nghĩa API endpoint thực hiện lấy thông tin hoặc xóa 1 product

```
@app.route('/product/<int:id>', methods = ['GET', 'DELETE'])
def get_product(id):
    if request.method == 'GET':
        prod = Product.query.filter_by(id = id).first()
        if not prod:
            abort(404)
        else:
            return jsonify({
                'id': prod.id,
                'name': prod.name,
                'price': str(prod.price)
            })
    if request.method == 'DELETE':
        prod = Product.query.filter_by(id = id).first()
        if not prod:
            abort(404)
        else:
            Product.query.filter_by(id = id).delete()
            return jsonify({'success': True})
    else:
        abort(404)
```



# Xây dựng app.py (4)

- Định nghĩa API endpoint thực hiện tạo mới hoặc cập nhật 1 product

```
@app.route('/product', methods = ['POST', 'PUT'])
def handle_product():
    if request.method == 'POST':
        name = request.form.get('name')
        price = request.form.get('price')
        prod = Product(name, price)
        db.session.add(prod)
        db.session.commit()
        return jsonify({
            'id': prod.id,
            'name': prod.name,
            'price': str(prod.price)
        })
    # if request.method == 'PUT':
```





# Xây dựng app.py (4)

- Định nghĩa API endpoint thực hiện tạo mới hoặc cập nhật 1 product (tiếp)

```
#@app.route('/product', methods = ['POST', 'PUT'])
#def handle_product():
#    # if request.method == 'POST':
#    if request.method == 'PUT':
#        id = request.form.get('id')
#        prod = Product.query.filter_by(id = id).first()
#        if not prod:
#            abort(404)
#        else:
#            prod.name = request.form.get('name')
#            prod.price = request.form.get('price')
#            db.session.commit()
#            return jsonify({
#                'id': prod.id,
#                'name': prod.name,
#                'price': str(prod.price)
#            })
#    else:
#        abort(404)
```



# Xây dựng app.py (5)

---

- Khi khởi chạy **app.py** thực hiện tạo cơ sở dữ liệu nếu chưa tồn tại các bảng tương ứng với các Model class

```
if __name__ == '__main__':  
    db.create_all()  
    app.run(debug = True)
```

- Để kiểm thử các API endpoints, có thể cài đặt extension **Restlet Client** cho trình duyệt **Google Chrome** hoặc cài đặt các ứng dụng khác như **Postman** để test các API endpoints
- Thực hiện test trên localhost thì các API endpoint như sau (port mặc định: 5000)
  - `http://localhost:5000/products`
  - `http://localhost:5000/product`
  - `http://localhost:5000/product/<product_id>`



# Kiểm thử các API endpoints

- Ví dụ: thực hiện gọi API endpoint `/product` với method **POST**, các tham số **HTML form**:  
**name, price**

METHOD: POST SCHEME: // HOST: [ ":" PORT ] [ PATH [ "?" QUERY ] ]

http://localhost:5000/product length: 29 bytes

Send

QUERY PARAMETERS

+ Add query parameter

HEADERS 1 Form

☒ Content-Type : application/x-www-form-urlencoded x

+ Add header Add authorization

BODY 1 Form

☒ name [ Text ] = iPhone 8 x

☒ price [ Text ] = 799 x

## Response

Cache Detected - Elapsed Time: 135ms

200 OK

HEADERS 1 pretty

Content-Type: application/json

Content-Length: 92 bytes

Server: Werkzeug/0.15.4 Python/3.7.0

Date: Sat, 06 Jul 2019 11:37:20 GMT

COMPLETE REQUEST HEADERS

BODY 1 pretty

```
{
  "product": {
    "id": 3,
    "name": "iPhone 8",
    "price": "799.0000000000"
  }
}
```



# Thư viện Volley

- Thư viện **Google Volley** giúp thực hiện các network request trong ứng dụng Android đơn giản và nhanh chóng hơn. Các bước tích hợp **Volley** vào project:
- **Bước 1:** Khai báo sử dụng **Volley** trong **Gradle**

```
dependencies {  
    implementation 'com.android.volley:volley:1.1.1'  
}
```
- **Bước 2:** Sử dụng **Volley**
  - Có 2 class của **Volley** cần quan tâm:
    - **RequestQueue** – Các request được đưa vào hàng đợi và được thực thi
    - **Request** – Khởi tạo các network request
  - Request gồm các loại sau:
    - **JsonObjectRequest**: Gửi yêu cầu để nhận JSON object từ server
    - **JSONArrayRequest**: Gửi yêu cầu để nhận JSON array từ server
    - **ImageRequest**: Gửi yêu cầu để nhận hình ảnh từ server
    - **StringRequest**: Gửi yêu cầu để nhận String từ server



# Sử dụng Volley làm REST API client

- Khởi tạo **RequestQueue**: Tất cả các request trong **Volley** đều được đưa vào một hàng đợi, do đó cần tạo ra một đối tượng **RequestQueue**

```
public class MainActivity extends AppCompatActivity {  
    RequestQueue queue;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        queue = Volley.newRequestQueue(this);  
    }  
}
```

- Minh họa tiếp theo sẽ thực hiện các request tương ứng với các API endpoint đã được định nghĩa bằng Python. Các request này thuộc loại **StringRequest**, các chuỗi kết quả trả về được phân tách sử dụng thư viện **Gson**



# Volley – Minh hoạ REST API client

- Tạo các **Model** class

```
public class Product {  
    int id;  
    String name;  
    String price;  
}
```

```
{  
    "id": 1,  
    "name": "iPhone 6s",  
    "price": "599.0"  
}
```

```
public class ProductResponse {  
    ArrayList<Product> products;  
  
    public ProductResponse() {  
        products = new ArrayList<>();  
    }  
}
```

```
{  
    "products": [  
        {  
            "id": 1,  
            "name": "iPhone 7s",  
            "price": "699.0"  
        },  
        {  
            "id": 2,  
            "name": "iPhone 6s",  
            "price": "699.0"  
        }  
    ]  
}
```

```
public class BooleanResponse {  
    boolean success;  
}
```

```
{  
    "success": true  
}
```



# Volley – Minh họa lấy danh sách products

- API endpoint: `http://10.0.2.2:5000/products` (method: **GET**) với **10.0.2.2** là IP của máy tính chạy Android emulator, cũng là máy host các Web services
- Kết quả trả về trong **onResponse** được phân tách bằng **Gson**

```
private void getProducts() {  
    String url = "http://10.0.2.2:5000/products";  
    StringRequest stringRequest = new StringRequest(Request.Method.GET, url,  
        new Response.Listener<String>() {  
            @Override  
            public void onResponse(String response) {  
                Gson gson = new GsonBuilder().create();  
                ProductResponse pr = gson.fromJson(response, ProductResponse.class);  
            }  
        }, new Response.ErrorListener() {  
            @Override  
            public void onErrorResponse(VolleyError error) {  
                Toast.makeText(MainActivity.this, "Error", Toast.LENGTH_SHORT).show();  
            }  
        });  
    queue.add(stringRequest);  
}
```



# Volley – Minh hoạ lấy thông tin 1 product

---

```
private void getProduct(int id) {  
    String url = "http://10.0.2.2:5000/product/" + id;  
    StringRequest stringRequest = new StringRequest(Request.Method.GET, url,  
        new Response.Listener<String>() {  
            @Override  
            public void onResponse(String response) {  
                Gson gson = new GsonBuilder().create();  
                Product prod = gson.fromJson(response, Product.class);  
            }  
        }, new Response.ErrorListener() {  
            @Override  
            public void onErrorResponse(VolleyError error) {  
                Toast.makeText(MainActivity.this, "Error", Toast.LENGTH_SHORT).show();  
            }  
        });  
    queue.add(stringRequest);  
}
```







# Volley – Minh họa tạo 1 product mới

```
private void createProduct(final Product p) {  
    String url = "http://10.0.2.2:5000/product";  
    StringRequest stringRequest = new StringRequest(Request.Method.POST, url,  
        new Response.Listener<String>() {  
            @Override  
            public void onResponse(String response) {  
                Gson gson = new GsonBuilder().create();  
                Product prod = gson.fromJson(response, Product.class);  
            }  
        }, new Response.ErrorListener() {  
            @Override  
            public void onErrorResponse(VolleyError error) {  
                Toast.makeText(MainActivity.this, "Error", Toast.LENGTH_SHORT).show();  
            }  
        }) {  
            @Override  
            protected Map<String, String> getParams() {  
                Map<String, String> params = new HashMap<>();  
                params.put("name", p.name);  
                params.put("price", p.price);  
                return params;  
            }  
        };  
    queue.add(stringRequest);  
}
```

Override phương thức **getParams** của class **Request** để truyền các parameters tương ứng với product cần tạo mới





# Volley – Minh hoạ cập nhật 1 product

```
private void updateProduct(final Product p) {
    String url = "http://10.0.2.2:5000/product";
    StringRequest stringRequest = new StringRequest(Request.Method.PUT, url,
        new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                Gson gson = new GsonBuilder().create();
                Product prod = gson.fromJson(response, Product.class);
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                Toast.makeText(MainActivity.this, "Error", Toast.LENGTH_SHORT).show();
            }
        }) {
        @Override
        protected Map<String, String> getParams() {
            Map<String, String> params = new HashMap<>();
            params.put("id", String.valueOf(p.id));
            params.put("name", p.name);
            params.put("price", p.price);
            return params;
        }
    };
    queue.add(stringRequest);
}
```





# Volley – Minh hoạ xoá 1 product

```
private void deleteProduct(int id) {
    String url = "http://10.0.2.2:5000/product/" + id;
    StringRequest stringRequest = new StringRequest(Request.Method.DELETE, url,
        new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                Gson gson = new GsonBuilder().create();
                BooleanResponse res = gson.fromJson(response, BooleanResponse.class);
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                try {
                    Toast.makeText(MainActivity.this, new String(error.networkResponse.data, "UTF-8"),
                        Toast.LENGTH_SHORT).show();
                } catch (UnsupportedEncodingException e) {
                    e.printStackTrace();
                }
            }
        });
    queue.add(stringRequest);
}
```



# Thư viện Retrofit

---

- **Retrofit** là thư viện REST client cho Android, được phát triển bởi Square, giúp thực hiện các REST API request nhanh chóng
- Cấu hình **Gradle** file để tích hợp Retrofit vào project:

```
dependencies {  
    implementation 'com.squareup.retrofit2:retrofit:2.6.0' // 1  
    implementation 'com.squareup.retrofit2:converter-gson:2.6.0' // 2  
}
```

- (1) - Khai báo sử dụng **Retrofit**
- (2) - Khai báo thêm thư viện này nếu sử dụng thư viện **Gson** để phân tách kết quả trả về từ API



# Retrofit – Các bước tích hợp

---

- **Bước 1:** Định nghĩa các API endpoints, bằng cách tạo 1 interface và khai báo các phương thức tương ứng với các endpoint theo cách sau:

```
public interface ProductService {  
    @GET("products")  
    Call<ProductResponse> getProducts();  
  
    @GET("product/{id}")  
    Call<Product> getProduct(@Path("id") int id);  
  
    @FormUrlEncoded  
    @POST("product")  
    Call<Product> createProduct(@Field("name") String name, @Field("price") String price);  
  
    @FormUrlEncoded  
    @PUT("product")  
    Call<Product> updateProduct(@Field("id") int id, @Field("name") String name, @Field("price") String price);  
  
    @DELETE("product/{id}")  
    Call<BooleanResponse> deleteProduct(@Path("id") int id);  
}
```



# Retrofit - Annotations

---

- **Request methods:** Có 5 annotations tương ứng với các HTTP methods: GET, POST, PUT, DELETE, HEAD → `@GET`, `@POST`, `@PUT`, `@DELETE`, `@HEAD`. Khai báo các annotations này trước method tương ứng.
- **Form:** Để truyền tham số HTML form, khai báo annotation `@FormUrlEncoded` trước method tương ứng. Với mỗi tham số của form (cặp key-value), sử dụng annotation `@Field` để xác định tên key.
- **Tuỳ biến URL:** Một request URL có thể được linh động thay đổi sử dụng khối `{tên_tham_số}` trên annotation của method. Tên tham số phải khai báo với annotation `@Path`



# Retrofit – Khởi tạo đối tượng Retrofit

- **Bước 2:** Khởi tạo đối tượng Retrofit và tạo đối tượng tương ứng với interface **ProductService** quản lý các API endpoints

```
public class MainActivity extends AppCompatActivity {  
    ProductService service;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        Retrofit retrofit = new Retrofit.Builder()  
            .baseUrl("http://10.0.2.2:5000/")  
            .addConverterFactory(GsonConverterFactory.create())  
            .build();  
        service = retrofit.create(ProductService.class);  
    }  
}
```



# Retrofit – Minh họa lấy danh sách các products

- Thực hiện một API call bằng cách **enqueue** một **Call**, kết quả trả về trong **onResponse**, lỗi trả về **onFailure**

```
private void getProducts() {  
    Call<ProductResponse> call = service.getProducts();  
    call.enqueue(new Callback<ProductResponse>() {  
        @Override  
        public void onResponse(Call<ProductResponse> call, Response<ProductResponse> response) {  
            ProductResponse productResponse = response.body(); // body của response là chuỗi kết quả trả về  
            ArrayList<Product> products = productResponse.products;  
        }  
  
        @Override  
        public void onFailure(Call<ProductResponse> call, Throwable t) {  
            Toast.makeText(MainActivity.this, t.getMessage(), Toast.LENGTH_SHORT).show();  
        }  
    });  
}
```







# Retrofit – Minh hoạ lấy thông tin 1 product

---

```
private void getProduct(int id) {  
    Call<Product> call = service.getProduct(id);  
    call.enqueue(new Callback<Product>() {  
        @Override  
        public void onResponse(Call<Product> call, Response<Product> response) {  
            Product prod = response.body();  
        }  
  
        @Override  
        public void onFailure(Call<Product> call, Throwable t) {  
  
        }  
    });  
}
```



# Retrofit - Minh họa lấy tạo mới 1 product

---

```
private void createProduct(String name, String price) {  
    Call<Product> call = service.createProduct(name, price);  
    call.enqueue(new Callback<Product>() {  
        @Override  
        public void onResponse(Call<Product> call, Response<Product> response) {  
            Product prod = response.body();  
        }  
  
        @Override  
        public void onFailure(Call<Product> call, Throwable t) {  
  
        }  
    });  
}
```



# Retrofit - Minh hoạ cập nhật 1 product

---

```
private void updateProduct(Product p) {  
    Call<Product> call = service.updateProduct(p.id, p.name, p.price);  
    call.enqueue(new Callback<Product>() {  
        @Override  
        public void onResponse(Call<Product> call, Response<Product> response) {  
            Product prod = response.body();  
        }  
  
        @Override  
        public void onFailure(Call<Product> call, Throwable t) {  
  
        }  
    });  
}
```



# Retrofit – Minh hoạ xoá 1 product

---

```
private void deleteProduct(int id) {  
    Call<BooleanResponse> call = service.deleteProduct(id);  
    call.enqueue(new Callback<BooleanResponse>() {  
        @Override  
        public void onResponse(Call<BooleanResponse> call, Response<BooleanResponse> response) {  
            BooleanResponse booleanResponse = response.body();  
            boolean success = booleanResponse.success;  
        }  
  
        @Override  
        public void onFailure(Call<BooleanResponse> call, Throwable t) {  
        }  
    });  
}
```



# Thư viện Glide

- **Glide** là một thư viện tải về và hiển thị hình ảnh được phát triển bởi **bumptech**, được khuyến dùng bởi **Google**

- Để sử dụng **Glide**, khai báo trong **Gradle** file:

```
dependencies {  
    implementation 'com.github.bumptech.glide:glide:4.9.0'  
    annotationProcessor 'com.github.bumptech.glide:compiler:4.9.0'  
}
```

- Cách sử dụng **Glide** cơ bản nhất:

```
Glide.with(context)  
    .load("https://picsum.photos/300")  
    .into(ivImage);
```

- Hình ảnh cho placeholder và error; resize và crop hình ảnh:

```
Glide.with(this)  
    .load("https://picsum.photos/300")  
    .placeholder(R.drawable.icon_image)  
    .error(R.drawable.icon_error)  
    .override(300, 200)  
    .centerCrop()  
    .into(ivImage);
```



# Thư viện Picasso

---

- **Picasso** cũng là một thư viện tải về và hiển thị hình ảnh được phát triển bởi **Square**
- Để sử dụng **Picasso**, khai báo trong **Gradle** file:

```
dependencies {  
    implementation 'com.squareup.picasso:picasso:2.71828'  
}
```

- Cách sử dụng **Picasso** cơ bản nhất:

```
Picasso.get()  
    .load("https://picsum.photos/300")  
    .into(ivImage);
```

- Hình ảnh cho placeholder và error; resize và crop hình ảnh:

```
Picasso.get()  
    .load("https://picsum.photos/300")  
    .placeholder(R.drawable.icon_image)  
    .error(R.drawable.icon_error)  
    .resize(200, 200)  
    .centerCrop()  
    .into(ivImage);
```

