Webservice - RESTful

Phát triển ứng dụng trên thiết bị di động nâng cao

Giới thiệu về Webservice

- Khái niệm về Webservice
- Đặc điểm và kiến trúc của dịch vụ Web
- Các thành phần của dịch vụ Web

Khái niệm về Webservice

- Webservice là một ứng dụng hoặc giao diện lập trình cung cấp dịch vụ thông qua mạng (Internet hoặc Intranet).
- Cho phép các ứng dụng khác nhau giao tiếp và trao đổi dữ liệu mà không phụ thuộc vào nền tảng hay ngôn ngữ lập trình.
- Sử dụng các giao thức tiêu chuẩn như HTTP, XML, JSON, SOAP, REST.
- Webservice có thể được sử dụng để chia sẻ dữ liệu giữa các hệ thống phần mềm khác nhau như ứng dụng web, ứng dụng di động, hệ thống quản lý doanh nghiệp.
- Hai loại chính của Webservice:
  - **SOAP Webservice**: Sử dụng giao thức SOAP và định dạng XML để truyền dữ liệu.
  - **RESTful Webservice**: Dựa trên kiến trúc REST, thường sử dụng HTTP và JSON.

Đặc điểm của Webservice

- **Tính tương tác (Interoperability):** Có thể sử dụng giữa các nền tảng và công nghệ khác nhau nhờ các giao thức tiêu chuẩn.
- **Tính mở rộng (Scalability):** Hỗ trợ mở rộng theo nhu cầu của người dùng và hệ thống.
- **Tính bảo mật (Security):** Có thể áp dụng các phương thức bảo mật như xác thực người dùng, mã hóa dữ liệu, SSL/TLS.
- **Tính linh hoạt (Flexibility):** Có thể tích hợp vào nhiều hệ thống phần mềm khác nhau, từ ứng dụng web đến ứng dụng di động.
- **Khả năng tự động hóa (Automation):** Cho phép giao tiếp giữa các hệ thống phần mềm mà không cần can thiệp của con người.

Kiến trúc của dịch vụ Web

- **Client:** Gửi yêu cầu tới Webservice qua giao thức HTTP.
- **Webservice:** Tiếp nhận yêu cầu, xử lý và gửi phản hồi.
- **Service Provider:** Máy chủ lưu trữ Webservice, thực hiện xử lý dữ liệu.
- **Service Registry:** Kho lưu trữ thông tin về các Webservice, giúp tìm kiếm và kết nối dịch vụ.
- **Service Consumer:** Ứng dụng hoặc hệ thống sử dụng Webservice.
- Mô hình hoạt động của Webservice thường bao gồm các bước:
  - Client gửi yêu cầu tới Webservice thông qua HTTP.
  - Webservice nhận yêu cầu, xử lý dữ liệu và phản hồi.
  - Client nhận phản hồi và sử dụng dữ liệu theo nhu cầu.

Các thành phần của Webservice

- **1. Giao thức truyền thông**
- **HTTP/HTTPS:** Giao thức cơ bản dùng để trao đổi dữ liệu giữa client và server.
- **SOAP (Simple Object Access Protocol):** Giao thức dựa trên XML, hỗ trợ truyền dữ liệu có cấu trúc, thường sử dụng trong hệ thống doanh nghiệp lớn.
- **REST (Representational State Transfer):** Kiến trúc dịch vụ web phổ biến, sử dụng phương thức HTTP đơn giản (GET, POST, PUT, DELETE).
- **2. Định dạng dữ liệu**
- **XML (Extensible Markup Language):** Ngôn ngữ đánh dấu phổ biến để trao đổi dữ liệu có cấu trúc giữa các hệ thống.
- **JSON (JavaScript Object Notation):** Định dạng dữ liệu nhẹ, dễ đọc và phổ biến trong các API RESTful.

Các thành phần của Webservice

- **3. Công cụ triển khai Webservice**
- **JAX-RS (Java API for RESTful Web Services):** API hỗ trợ tạo RESTful Webservice trong Java.
- **JAX-WS (Java API for XML Web Services):** API hỗ trợ tạo SOAP Webservice trong Java.
- **Spring Boot:** Framework hỗ trợ phát triển nhanh các Webservice RESTful với ít cấu hình hơn.
- **Retrofit:** Thư viện mạnh mẽ giúp Android kết nối với RESTful Webservice dễ dàng.

Xây dựng và tích hợp Webservice

- Cách xây dựng Webservice
- Tích hợp Webservice vào ứng dụng Android
- Ví dụ: Gọi API từ ứng dụng Android

SOAP (Simple Object Access Protocol) và REST (Representational State Transfer)

- SOAP là một giao thức nhắn tin tiêu chuẩn, hoạt động dựa trên XML và thường được sử dụng trong các hệ thống yêu cầu tính bảo mật và độ tin cậy cao.
- **Ưu điểm:**
  - **Bảo mật cao:** Tích hợp sẵn WS-Security, phù hợp với các hệ thống tài chính, ngân hàng, bảo hiểm.
  - **Hỗ trợ giao dịch (ACID):** Quan trọng với các ứng dụng cần đảm bảo tính toàn vẹn dữ liệu.
  - **Hỗ trợ hợp đồng dịch vụ (WSDL):** Giúp định nghĩa rõ ràng về dịch vụ, thuận tiện trong tích hợp hệ thống.
- **Nhược điểm:**
  - **Nặng nề, chậm hơn REST:** Do sử dụng XML nên tốn băng thông và tài nguyên hơn.
  - **Khó triển khai hơn REST.**
- **Nên dùng SOAP khi:**
  - Cần bảo mật cao (dịch vụ tài chính, y tế, chính phủ).
  - Giao dịch yêu cầu đảm bảo tính nguyên tử (ví dụ: thanh toán trực tuyến).
  - Hệ thống có cấu trúc chặt chẽ và cần một giao thức chuẩn hóa.

SOAP (Simple Object Access Protocol) và REST (Representational State Transfer)

- REST là một phong cách kiến trúc web service dựa trên HTTP, thường sử dụng các định dạng JSON hoặc XML.
- **Ưu điểm:**
- **Gọn nhẹ, hiệu suất cao:** REST sử dụng JSON, tiết kiệm băng thông hơn XML của SOAP.
- **Dễ dàng triển khai:** Hoạt động tốt với API trên web, hỗ trợ các phương thức HTTP phổ biến như GET, POST, PUT, DELETE.
- **Tính mở rộng cao:** Dễ dàng phát triển thêm tính năng, phù hợp với các hệ thống phân tán.
- **Nhược điểm:**
- **Bảo mật mặc định thấp hơn SOAP:** Phải kết hợp với các giao thức bảo mật như OAuth, JWT.
- **Không hỗ trợ giao dịch phức tạp như SOAP.**
- **Nên dùng REST khi:**
- Cần hiệu suất cao, truy xuất nhanh (ứng dụng di động, web API).
- Hệ thống không yêu cầu giao dịch phức tạp hoặc bảo mật mức cao.
- Cần tương tác với nhiều nền tảng khác nhau một cách linh hoạt.

Cách xây dựng Webservice

1.Lựa chọn công nghệ phù hợp: Quyết định sử dụng SOAP hay REST dựa trên yêu cầu của dự án.

2.Tạo API Webservice bằng Java:

- Sử dụng Spring Boot để tạo Webservice RESTful.
- Cấu hình endpoints với @RestController và @RequestMapping.

3.Triển khai Webservice lên server:

- Sử dụng Tomcat hoặc các nền tảng cloud như AWS, Firebase.

4.Kiểm thử Webservice:

- Dùng Postman hoặc Swagger UI để kiểm thử API trước khi tích hợp vào ứng dụng Android.

Tích hợp Webservice vào ứng dụng Android

1. **Chọn thư viện HTTP phù hợp:**
   - Retrofit (phổ biến nhất cho Android)
   - Volley (dễ sử dụng, mạnh mẽ)
2. **Thêm dependency vào build.gradle**

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

3. Cấu hình Retrofit API Client:

```
public interface ApiService {
    @GET("users")
    Call<List<User>> getUsers();
}
```

4. Gửi yêu cầu HTTP từ ứng dụng Android:

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.example.com/")
    .addConverterFactory(GsonConverterFactory.create())
    .build();

ApiService apiService = retrofit.create(ApiService.class);
```

Ví dụ - Gọi API từ ứng dụng Android

Gửi yêu cầu GET để lấy danh sách người dùng

```
Call<List<User>> call = apiService.getUsers();
call.enqueue(new Callback<List<User>>() {
    @Override
    public void onResponse(Call<List<User>> call,
    Response<List<User>> response) {
        if (response.isSuccessful()) {
            List<User> users = response.body(); // Xử lý danh sách người dùng
        }
    }
    @Override
    public void onFailure(Call<List<User>> call, Throwable t) {
    Log.e("API_ERROR", t.getMessage());
}});
```

Hiển thị dữ liệu lên RecyclerView

- Sử dụng RecyclerView để hiển thị danh sách dữ liệu từ API.
- Cập nhật adapter khi dữ liệu được tải về thành công.

RESTful API

- Khái niệm và kiến trúc RESTful
- Cách truy xuất dịch vụ RESTful
- Thiết kế dịch vụ RESTful

RESTful API là gì?

- API (Application Programming Interface) là giao diện giữa các hệ thống.
- REST (Representational State Transfer) là một phong cách thiết kế API dựa trên giao tiếp HTTP.
- RESTful API là API tuân theo các nguyên tắc REST, cho phép truy xuất tài nguyên qua HTTP.

Kiến Trúc RESTful

- **Client-Server**: Phân tách giữa client và server.
- **Stateless**: Mỗi request tồn tại độc lập.
- **Cacheable**: Hỗ trợ caching tăng hiệu suất.
- **Layered System**: Kiến trúc phân lớp.
- **Uniform Interface**: Sử dụng giao diện nhất quán.

Các Phương Thức HTTP trong RESTful

- **GET**: Lấy dữ liệu.
- **POST**: Tạo mới tài nguyên.
- **PUT**: Cập nhật tài nguyên.
- **DELETE**: Xóa tài nguyên.

Cách Truy Xuất Dịch Vụ RESTful

1. Gửi Request HTTP

•Sử dụng các thư viện như Retrofit (Android) hoặc HttpURLConnection.

•Header thường dùng: Content-Type, Authorization.

2. Nhận và Xử Lý Response

•Dữ liệu trả về thường là JSON.

•Kiểm tra status code: 200 OK, 404 Not Found, 500 Internal Server Error

Thiết Kế RESTful API

1. Quy Định Về URL

•Sử dụng danh từ (nouns), không dùng động từ (verbs).

•VD: /users thay vì /getUsers.

2. Phân Trang, Lọc, Sắp Xếp

•Sử dụng query params: ?page=1&size=10&sort=name.

3. Xác Thực Và Phân Quyền

•Sử dụng JWT, OAuth2.

4. Xử Lý Lỗi API

•Trả về mã lỗi HTTP và message rõ ràng.

Android Gọi RESTful API

- 1. Cài Đặt Retrofit

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

- 2. Tạo Interface API

```
public interface ApiService {
    @GET("users")
    Call<List<User>> getUsers();
}
```

3. Gọi API trong Activity

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.example.com/")
    .addConverterFactory(GsonConverterFactory.create()).build();

ApiService apiService = retrofit.create(ApiService.class);

Call<List<User>> call = apiService.getUsers();

call.enqueue(new Callback<List<User>>() {

    @Override

    public void onResponse(Call<List<User>> call, Response<List<User>>
response) {

     if (response.isSuccessful()) {

         List<User> users = response.body(); }

    }

    @Override

    public void onFailure(Call<List<User>> call, Throwable t) {
    Log.e("API Error", t.getMessage());    }});
```

Thực hiện truy xuất RESTful API

- Gửi yêu cầu GET/POST bằng Retrofit
- Xử lý phản hồi JSON từ server
- Ví dụ: Hiển thị dữ liệu từ RESTful API lên RecyclerView

RESTful API và Retrofit

- **RESTful API**
- RESTful API là một giao diện lập trình ứng dụng (API) tuân theo các nguyên tắc REST (Representational State Transfer). Nó sử dụng các phương thức HTTP như GET, POST, PUT, DELETE để giao tiếp giữa client và server.
- **Retrofit**
- Retrofit là một thư viện mạnh mẽ và dễ sử dụng để thực hiện các yêu cầu HTTP trong Android. Nó giúp chuyển đổi phản hồi JSON thành các đối tượng Java một cách dễ dàng.

Cài đặt Retrofit

Thêm Retrofit và Gson vào file build.gradle của module app:

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

Sau đó, đồng bộ hóa (Sync Now).

Gửi yêu cầu GET bằng Retrofit

- **Tạo model dữ liệu**
- Giả sử API trả về danh sách người dùng với định dạng JSON như sau:

```
[
  {"id": 1, "name": "Nguyen Van A", "email": "a@example.com"},
  {"id": 2, "name": "Tran Thi B", "email": "b@example.com"}
]
```

Gửi yêu cầu GET bằng Retrofit

Tạo lớp User.java để ánh xạ dữ liệu:

```
public class User {
    private int id;
    private String name;
    private String email;

    public int getId() { return id; }
    public String getName() { return name; }
    public String getEmail() { return email; }
}
```

Gửi yêu cầu GET bằng Retrofit

Tạo interface API

```
import retrofit2.Call;
import retrofit2.http.GET;
import java.util.List;

public interface ApiService {
    @GET("users")
    Call<List<User>> getUsers();
}
```

Cấu hình Retrofit

```
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;
public class ApiClient {
    private static final String
        BASE_URL = "https://example.com/api/";
private static Retrofit retrofit;
    public static Retrofit getClient() {
        if (retrofit == null) {
            retrofit = new Retrofit.Builder()
            .baseUrl(BASE_URL)
    .addConverterFactory(GsonConverterFactory.create())
        .build();       }
    return retrofit;
}}
```

Gọi API từ Activity

```
import android.os.Bundle;
import android.util.Log;
import androidx.appcompat.app.AppCompatActivity;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import java.util.List;
```

Goi API từ Activity

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ApiService apiService = ApiClient.getClient().create(ApiService.class);
    apiService.getUsers().enqueue(new Callback<List<User>>(){
        @Override
        public void onResponse(Call<List<User>> call,
        Response<List<User>> response) {
            if (response.isSuccessful() && response.body() != null) {
                for (User user : response.body()) {
                Log.d("User", "Name: " + user.getName() + ", Email: " +
                user.getEmail());
                }
            }
            }
        @Override
        public void onFailure(Call<List<User>> call, Throwable t) {
            Log.e("API Error", t.getMessage());
        }
        });
}}
```

Gửi yêu cầu POST bằng Retrofit
Tạo lớp dữ liệu gửi lên server

```
public class UserRequest {
    private String name;
    private String email;

    public UserRequest(String name, String email) {
        this.name = name;
        this.email = email;
    }
}
```

Cập nhật interface API

```
import retrofit2.http.Body;
import retrofit2.http.POST;

public interface ApiService {
    @POST("users")
    Call<User> createUser(@Body UserRequest userRequest);
}
```

Gửi yêu cầu POST

```
UserRequest newUser = new UserRequest("Le Van C",
"c@example.com");
apiService.createUser(newUser).enqueue(new
Callback<User>() {
    @Override
    public void onResponse(Call<User> call,
    Response<User> response) {
        if (response.isSuccessful() && response.body() != null) {
          Log.d("User Created", "ID: " + response.body().getId());
        }   }
    @Override
    public void onFailure(Call<User> call, Throwable t) {
        Log.e("API Error", t.getMessage());
}});
```

Hiển thị dữ liệu từ API lên RecyclerView
Thêm RecyclerView vào layout

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Tạo Adapter

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import java.util.List;
public class UserAdapter extends RecyclerView.Adapter<UserAdapter.ViewHolder> {
	private List<User> userList;
	public UserAdapter(List<User> userList) {
		this.userList = userList;
	}
	@NonNull
	@Override
	public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent,
			int viewType) {
			View view =
LayoutInflater.from(parent.getContext()).inflate(android.R.layout.simple_list_item
_2, parent, false);
			return new ViewHolder(view);    }
```

Tạo Adapter

```
    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        User user = userList.get(position);
    holder.text1.setText(user.getName());
    holder.text2.setText(user.getEmail());
    }
    @Override
    public int getItemCount() {
        return userList.size();
    }
    public static class ViewHolder extends RecyclerView.ViewHolder {
        TextView text1, text2;
        public ViewHolder(@NonNull View itemView) {
            super(itemView);
            text1 = itemView.findViewById(android.R.id.text1);
            text2 = itemView.findViewById(android.R.id.text2);
        }
    }
}
```

Hiển thị dữ liệu

```
RecyclerView recyclerView =          findViewById(R.id.recyclerView);
recyclerView.setLayoutManager(new LinearLayoutManager(this));
apiService.getUsers().enqueue(new Callback<List<User>>() {
            @Override
            public void onResponse(Call<List<User>> call,
                  Response<List<User>> response) {
            recyclerView.setAdapter(new UserAdapter(response.body()));
            }
            @Override
            public void onFailure(Call<List<User>> call, Throwable t) {}
});
```

Bài tập thực hành

- Bài 1:
  - Tìm hiểu về RESTful API https://restcountries.com/
  - Tìm hiểu về POSTMAN test API
  - Viết ứng dụng hiển thị danh sách quốc gia
- Bài 2: Tạo ứng dụng hiển thị danh sách người dùng từ RESTful API
- Bài 3: Gửi dữ liệu đăng ký tài khoản từ ứng dụng Android lên server

Bài tập thực hành

- Bài 4: Ứng dụng sẽ gửi yêu cầu GET để lấy thông tin thời tiết hiện tại từ OpenWeatherMap API dựa vào thành phố mà người dùng nhập.

- Bài 5:Ứng dụng sẽ lấy tỷ giá chuyển đổi từ USD sang một loại tiền tệ do người dùng nhập.