



TRUNG TÂM TIN HỌC - ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

CHƯƠNG TRÌNH ĐÀO TẠO
LẬP TRÌNH VIÊN CHUYÊN NGHIỆP TRÊN THIẾT BỊ DI ĐỘNG

TÀI LIỆU



LẬP TRÌNH THIẾT BỊ DI ĐỘNG TRÊN
ANDROID

MODULE 04:
LẬP TRÌNH ỨNG DỤNG CLIENT VÀ ỨNG DỤNG ĐA
PHƯƠNG TIỆN (SMS, Camera, ...)

Bài 1

KHAI THÁC TÀI NGUYÊN INTERNET

1. TỔNG QUAN TÀI NGUYÊN INTERNET

1.1. Tài nguyên Internet trên thiết bị di động

Ứng dụng trực tuyến trên thiết bị di động được xây dựng nhằm mục đích hiển thị dữ liệu đầu cuối được xử lý từ các máy chủ cũng như gửi các yêu cầu về truy xuất dữ liệu. Ví dụ: ứng dụng Google Play Store được Google cài đặt trên thiết bị người dùng, hiển thị các ứng dụng được phát triển bởi các lập trình viên, cho phép người dùng có thể tải và cài đặt các ứng dụng đó ngay trên thiết bị của họ.

Khi xây dựng các ứng dụng trực tuyến cần xác định rõ nguồn tài nguyên được cung cấp:

- Định dạng tài nguyên (văn bản, hình ảnh...).
- Giao thức kết nối đến máy chủ (http, https, tcp, rtsp...).
- Sử dụng công nghệ kết nối không dây (Wifi, Internet Mobile...).

Các tài nguyên sau khi được truy xuất sẽ dựa trên loại tài nguyên và sử dụng các API tương ứng để nhận và chuyển đổi thành dữ liệu tường minh mà người dùng có thể đọc và hiểu được trên ứng dụng. Android cung cấp đầy đủ các phương thức thư viện cho phép lập trình viên có thể làm việc được hầu hết các loại dữ liệu phổ biến. Ví dụ, có thể sử dụng WebView và các phương thức tương ứng để có thể hiển thị thông tin của một trang Web ngay trên màn hình ứng dụng.

1.2. Các vấn đề khi kết nối Internet

Một trong những vấn đề được đặt ra khá quan trọng là liệu có nên xây dựng một ứng dụng hoàn chỉnh trên thiết bị di động (Native App), bởi vì chúng ta hoàn toàn có thể xây dựng một trang web và cung cấp giao diện riêng cho thiết bị di động (Web App). Khi đó người dùng có thể sử dụng trình duyệt trên thiết bị và truy xuất đường dẫn riêng đến trang web đó (ví dụ: <http://m.t3h.vn>). Việc xây dựng Web App giúp lập trình viên có thể tiết kiệm được thời gian xây dựng ứng dụng riêng cho từng hệ điều hành, tuy nhiên vẫn khó tránh khỏi những khuyết điểm mà chỉ khi xây dựng ứng dụng chúng ta mới có thể giải quyết được.

- **Vấn đề về băng thông:** khi sử dụng trình duyệt để duyệt đến web ứng dụng, tất cả các tài nguyên hầu như bắt buộc phải tải lại từ đầu, ví dụ: hình ảnh, giao diện, âm thanh... điều này tạo ra một lượng băng thông lớn để truy xuất các dữ liệu tĩnh, thay vào đó xây dựng ứng dụng có thể giúp chúng ta lưu trữ những dữ liệu này một cách dễ dàng.

- **Lưu Cache:** các trình duyệt chỉ có thể lưu giữ một lượng thông tin nhỏ về trang web mà chúng làm việc, trong khi đó ứng dụng hoàn toàn có thể lưu trữ bất kỳ dữ liệu nào mà lập trình viên mong muốn từ dữ liệu truy xuất đến thao tác người dùng. Khi có sự cố về kết nối, ứng dụng sẽ lưu trữ lại thông tin tại thời điểm mất kết nối và tự động xử lý khi thiết bị có lại kết nối.
- **Năng lượng:** các kết nối Internet khi được thực hiện sử dụng một lượng lớn năng lượng cho mỗi lần kết nối, tuy nhiên việc xây dựng ứng dụng sẽ cho phép chúng ta tổ chức quản lý kết nối cũng như duy trì các kết nối khi cần thiết.
- **Tính năng thiết bị:** các ứng dụng nền web hầu như không thể nào truy xuất đến phần cứng cũng như các đặc tính quan trọng trên một thiết bị di động. Ví dụ: định vị, máy ảnh, cảm biến... đương nhiên, khi xây dựng ứng dụng chúng ta hoàn toàn có thể dễ dàng truy xuất các đặc tính thiết bị thông qua các API mà bộ SDK cung cấp.

1.3. Các hình thức kết nối Internet

Các thiết bị di động ở thời điểm hiện tại hỗ trợ khá nhiều hình thức kết nối đến Internet, tuy nhiên ta có thể quy chung về dạng sau:

- **Mobile Internet:** một dạng kết nối thông qua hình thức cung cấp dịch vụ băng thông của nhà mạng, bao gồm một số chuẩn phổ biến sau: GPRS, EDGE, 3G, 4G, LTE...
- **Wifi:** kết nối không dây và truyền dữ liệu ra Internet thông các thiết bị mạng (router, switch...), ngoài ra thiết bị di động Android có khả năng phát tín hiệu mạng cho các thiết bị khác.

Việc xây dựng ứng dụng thực hiện kết nối đến Internet phụ thuộc vào khá nhiều hình thức kết nối mà thiết bị người dùng đang sử dụng. Do đó, để tối ưu hóa kết nối và tăng trải nghiệm người dùng cần thực hiện sử dụng hình thức kết nối một cách hợp lý. Ta có thể thấy các mạng di động thường có băng thông thấp và chi phí cao để sử dụng, trong khi đó Wifi lại tùy thuộc và băng thông của nguồn kết nối.

1.4. Lớp khai báo kết nối

Việc khai báo kết nối tùy thuộc vào chuẩn giao thức kết nối và địa chỉ URL của máy chủ cần kết nối:

- URL: lớp giải quyết phân giải tên miền thành địa chỉ IP.
 - o Định dạng: **http://username:password@host:8080/directory/file?query#ref:**
 - o Phương thức kết nối
 - openConnection
- URLConnection: lớp thực hiện kết nối đến URL được chỉ định cho việc đọc hoặc ghi dữ liệu, hỗ trợ các giao thức:
 - o File: URIs
 - o FTP
 - o HTTP & HTTPS
 - o Jar

- **URLConnection:**
 - Các phương thức quan trọng:
 - `connect()`
 - `getContent()`
 - `getContentType()`
 - `getInputStream()`
 - `getOutputStream()`
 - `setDoInput` – `getDoInput`
 - `setDoOutput` – `getDoOutput`
 - Các lớp giao thức HTTP & HTTPS:
 - `HttpURLConnection`
 - `HttpsURLConnection`
 - Các phương thức quan trọng:
 - . `disconnect()`
 - . `getContentEncoding()`
 - . `getResponseCode()`
 - . `getResponseMessage()`

1.4.1. Thực hiện kết nối Internet (HTTP)

- Bao gồm các bước sau:
 - Thực hiện mở kết nối đến địa chỉ URL được chỉ định → **`HttpURLConnection`**.
 - Thực hiện khai báo các header, content-type, cookies, ...
 - Gọi phương thức **`setDoOutput(true)`** thực hiện xây dựng phần dữ liệu cần gửi lên máy chủ, dữ liệu được thiết lập được trả thông qua phương thức **`getOutputStream()`**. (Optional).
 - Thực hiện truy xuất dữ liệu thông qua phương thức **`getInputStream()`**, để lấy các thông tin về nội dung được trả, độ dài, thời gian...
 - Gọi phương thức **`disconnect()`** để đóng các kết nối khi kết thúc phiên làm việc và giải phóng tài nguyên.
- Một số lưu ý khi thực hiện:
 - Cần thực hiện xin cấp quyền truy cập Internet cho ứng dụng trong tập tin `AndroidManifest`.

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

- Trong một vài trường hợp cần xin cấp quyền kiểm soát trạng thái Internet

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

- Không được thực hiện kết nối Internet trực tiếp trên tiến trình chính của ứng dụng. Sử dụng AsyncTask hoặc Thread để thay thế.
- Ví dụ:

```
URL url = new URL("http://t3h.vn");
URLConnection urlConnection = url.openConnection();
HttpURLConnection httpURLConnection =
    (HttpURLConnection) urlConnection;

int responseCode = httpURLConnection.getResponseCode();

if (responseCode == HttpURLConnection.HTTP_OK) {
    InputStream inputStream =
        httpURLConnection.getInputStream();
    // Xử lý đọc dữ liệu từ InputStream
}
```

1.4.2. Thực hiện kết nối Internet (HTTPS)

- Bao gồm các bước sau:
 - Khai báo **KeyStore** dùng để chứng thực.
 - Chứng thực KeyStore thông qua **X509TrustManager** hoặc **SSLConnectionFactory**.
 - Thực hiện mở kết nối đến địa chỉ URL được chỉ định → **HttpsURLConnection**.
 - Thực hiện gắn SSLContext cho việc chứng thực.
 - Gọi phương thức **setDoOutput(true)** thực hiện xây dựng phần dữ liệu cần gửi lên máy chủ, dữ liệu được thiết lập được trả thông qua phương thức **getOutputStream()**.
 - Thực hiện truy xuất dữ liệu thông qua phương thức **getInputStream()**, để lấy các thông tin về nội dung được trả, độ dài, thời gian...
 - Gọi phương thức **disconnect()** để đóng các kết nối khi kết thúc phiên làm việc và giải phóng tài nguyên.
- Ví dụ:

```
KeyStore keyStore = ...;
String algorithm = TrustManagerFactory.getDefaultAlgorithm();
TrustManagerFactory tmf = TrustManagerFactory.getInstance(algorithm);
tmf.init(keyStore);

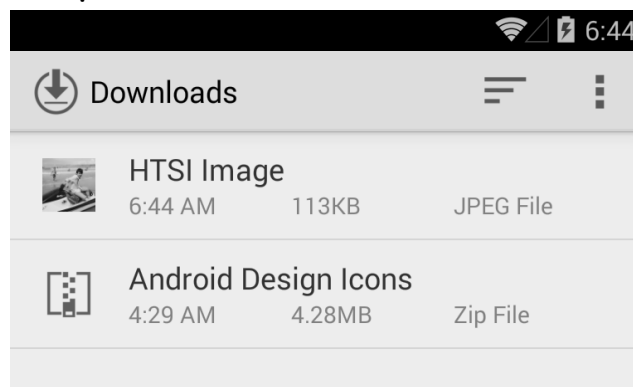
SSLContext context = SSLContext.getInstance("TLS");
context.init(null, tmf.getTrustManagers(), null);
```

```
URL url = new URL("https://www.example.com/");
HttpsURLConnection urlConnection = (HttpsURLConnection)
url.openConnection();
urlConnection.setSSLSocketFactory(context.getSocketFactory());
InputStream in = urlConnection.getInputStream();
```

2. SỬ DỤNG DỊCH VỤ DOWNLOAD MANAGER

2.1. Giới thiệu dịch vụ Download Manager

- Download Manager là trình quản lý các tác vụ tải trên thiết bị Android được Google tích hợp từ phiên bản Android 2.3 (API 9). Download Manager được thiết kế hoạt động như một dịch vụ chạy ngầm, cho phép quản lý và giám sát sự thay đổi của các kết nối HTTP cũng như các hoạt động khởi động lại của thiết bị để đảm bảo các nội dung được hoàn thành một cách triệt để.
- Download Manager được sử dụng trong các trường hợp cần thực hiện tải một nội dung trong một thời gian dài ở chế độ ngầm giữa các phiên tương tác của người dùng và việc hoàn thành nội dung tải được đặt lên hàng đầu.
- Tất cả các tập tin được tải thông qua Download Manager được quản lý bởi ứng dụng Download trên thiết bị.



Hình 1.1. Ví dụ minh họa dịch vụ Download Manager

2.2. Khai báo và sử dụng Download Manager

Việc truy xuất Download Manager được thông qua phương thức `getSystemService()`.

```
DownloadManager manager=(DownloadManager)getSystemService(DOWNLOAD_SERVICE);
```

Để khởi tạo yêu cầu tải một nội dung, tiến hành tạo đối tượng Request với tham số truyền vào là một Uri, một dạng địa chỉ kết nối đến máy chủ chứa nội dung cần tải. Đối tượng này sau đó sẽ được xếp vào hàng đợi để bắt đầu tải thông qua phương thức **enqueue()**.

```
DownloadManager manager = (DownloadManager)
getSystemService(DOWNLOAD_SERVICE);
Uri uri = Uri.parse
```

```

        ("http://developer.android.com/downloads/design/roboto-1.2.zip");

Request request = new Request(uri);

long id_reference = manager.enqueue(request);

```

- Khi phương thức enqueue() được gọi, việc tải nội dung sẽ được bắt đầu thực hiện khi kết nối mạng khả dụng và hàng đợi của Download Manager đang rỗng.
- Phương thức enqueue() cũng trả về một biến tham chiếu id_reference, được sử dụng để thực hiện các thao tác liên quan đến yêu cầu tải đang được thực hiện trong hàng đợi. Ví dụ, ta có thể thực hiện truy vấn xem trạng thái của yêu cầu tải đã hoàn thành chưa hoặc đơn giản có thể hủy yêu cầu tải dựa trên tham chiếu này.
- Khi thực hiện tải một nội dung, chúng ta có thể cân nhắc về dung lượng của nội dung và cho phép sử dụng các kết nối phù hợp nội dung đó thông qua phương thức **setAllowedNetworkTypes()**, lựa chọn tải bằng Wi-fi hay mạng di động.
- Từ phiên bản Android 3.0, phương thức **getRecommendedMaxBytesOverMobile()** được bổ sung cho phép ứng dụng có thể nhận biết dung lượng lớn nhất có thể mà thiết bị có thể thực hiện tải thông qua kết nối của mạng di động.
- Để nhận biết việc hoàn thành tải nội dung, ta tiến hành xây dựng một BroadcastReceiver để bắt lại hành động ACTION_DOWNLOAD_COMPLETE được truyền thông bởi dịch vụ chạy ngầm DownloadManager. Trong dữ liệu bắt được, cũng bao gồm tham số EXTRA_DOWNLOAD_ID chứa biến tham chiếu đến nội dung tải. Có thể xem đoạn code sau để hiểu thêm.

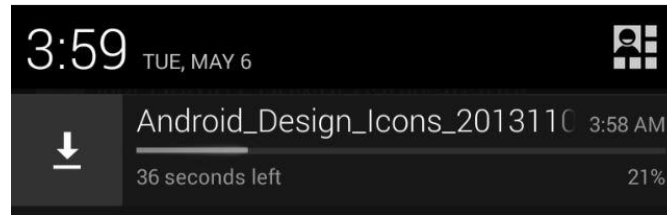
```

IntentFilter filter = new IntentFilter
                                (DownloadManager.ACTION_DOWNLOAD_COMPLETE);
BroadcastReceiver receiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        long id_reference =
            intent.getLongExtra(DownloadManager.EXTRA_DOWNLOAD_ID, -1);
        Log.d("HTSI", "File's id: " + id_reference);
    }
};
registerReceiver(receiver, filter);

```

2.3. Tùy chỉnh thông báo cho Download Manager

Mặc định khi thực hiện tải nội dung thông qua dịch vụ chạy ngầm Download Manager, trên thanh trạng thái (Status Bar) của thiết bị sẽ xuất hiện một hộp thoại thông báo hoạt động hiện tại, hiển thị các thông tin như: tên tập tin được tải, ngày giờ tải, dung lượng đã tải, ...



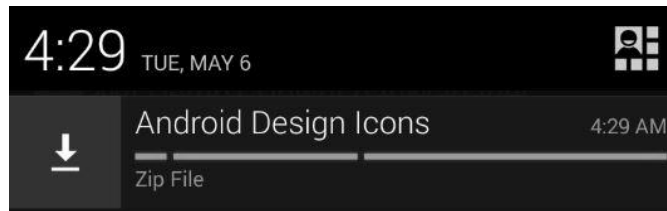
Hình 1.2

Tuy nhiên chúng ta hoàn toàn có thể tùy chỉnh thông tin hiển thị trên hộp thoại thông báo này tường minh hơn thông qua các phương thức hỗ trợ trong đối tượng Request.

Ví dụ:

```
request.setTitle("Android Design Icons");  
request.setDescription("Zip File");
```

Kết quả:



Hình 1.3

Hộp thoại thông báo mặc định sẽ xuất hiện khi bắt đầu tải và tự đóng khi hoàn tất quá trình tải, để thiết lập lại trạng thái này chúng ta có thể sử dụng phương thức **setVisibility()** với các biến cờ tương ứng sau:

- Request.VISIBILITY_VISIBLE: tham số mặc định, chỉ hiển thị trong quá trình tải.
- Request.VISIBILITY_VISIBLE_NOTIFY_COMPLETED: hiển thị trong quá trình tải và cả khi hoàn tất.
- Request.VISIBILITY_VISIBLE_NOTIFY_ONLY_COMPLETION: chỉ hiển thị khi hoàn tất quá trình tải.
- Request.VISIBILITY_HIDDEN: không hiển thị hộp thoại thông báo.

Đối với biến cờ VISIBILITY_HIDDEN cần cấp quyền trong AndroidManifest.xml

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

2.4. Chỉ định nơi lưu trữ

Mặc định các nội dung tải được hệ thống lưu trữ trong Content Providers của ứng dụng Download:

```
data/user/0/com.android.providers.downloads/cache/<Tên tập tin>
```


- Các tập tin trong thư mục này sẽ không thể truy xuất bằng đường dẫn của tập tin, chỉ có thể truy xuất thông qua Uri của tập tin đó.
- Tuy nhiên ta có thể thiết lập lại vị trí lưu trữ nội dung tải thông qua các phương thức sau:

- `setDestinationUri(Uri uri)`: chỉ định vị trí lưu trữ bất kỳ thông qua Uri.

- Ví dụ:

```
File f = new File(Environment.getExternalStorageDirectory(),  
                  uri.getLastPathSegment());  
request.setDestinationUri(Uri.fromFile(f));
```

- Đường dẫn lưu trữ:

```
/storage/emulated/0/<file-name>
```

- `setDestinationInExternalFilesDir(Context context, String dirType, String subPath)`: chỉ định vị trí lưu trữ trong thư mục ứng dụng trên bộ nhớ ngoài.

- Ví dụ:

```
request.setDestinationInExternalFilesDir(this, "My Files",  
    uri.getLastPathSegment());
```

- Đường dẫn lưu trữ:

```
/storage/emulated/0/Android/data/<package-name>/files/My Files/<file-name>
```

- `setDestinationInExternalPublicDir(String dirType, String subPath)`: chỉ định vị trí lưu trữ trong thư mục ứng dụng trên bộ nhớ ngoài.

- Ví dụ:

```
request.setDestinationInExternalPublicDir  
    (Environment.DIRECTORY_DOWNLOADS, uri.getLastPathSegment());
```

- Đường dẫn lưu trữ:

```
/storage/emulated/0/Download/<file-name>
```

- Các hành động truy xuất đến bộ nhớ ngoài cần cấp quyền trong tập tin `AndroidManifest.xml`

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

Mặc định, đối với các trường hợp tải tập tin đa truyền thông (ví dụ: hình ảnh, âm nhạc, video, ...) các ứng dụng đa phương tiện như Galery, Music hay Video không thể tự động quét được các nội dung này. Tuy nhiên, chúng ta có thể cho phép tập tin được quét lại khi hoàn tất việc tải thông qua phương thức ***allowScanningByMediaScanner()*** trong đối tượng Request.

2.5. Truy vấn nội dung tải trong DownloadManager

- Để ứng dụng có thể quản lý các thông tin của nội dung tải như: trạng thái, tiến độ, dung lượng, ... ta có thể thực hiện truy vấn Content Provider của Download Manager thông qua đối tượng ***Query***, kết quả trả về là một đối tượng Cursor. Có thể xem chi tiết ở đoạn mã sau:

```
Query query = new Query();
query.setFilterById(id_reference);
Cursor cursor = manager.query(query);
if (cursor.moveToFirst()) {
    String path = cursor.
        getString(cursor.getColumnIndex(DownloadManager.COLUMN_LOCAL_FILENAME));
    if (path != null)
        imageView.setImageURI(Uri.parse(path));
}
```

Trong đoạn mã trên, để thực hiện truy vấn đúng nội dung đã tải ta gọi phương thức ***setFilterById()*** với tham số là biến tham chiếu id_reference ta truy xuất được từ phương thức enqueue() (Xem lại phần III.1).

Việc truy vấn được thực hiện thông qua phương thức query() trong đối tượng DownloadManager, tham số truyền vào là đối tượng Query đã thiết lập bộ lọc. Kết quả trả về thông qua đối tượng Cursor, để truy xuất dữ liệu trong Cursor, tiến hành truyền vào tên cột của những trường dữ liệu tương ứng cần truy xuất. Tên cột của những trường dữ liệu được định dạng COLUMN_*.

Trong ví dụ trên, chúng ta truy xuất trường dữ liệu COLUMN_LOCAL_FILENAME trong Cursor để lấy đường dẫn lưu trữ nội dung tấm ảnh được tải, sau đó thiết lập cho điều khiển ImageView.

Bài 2

KẾT NỐI CÁC DỊCH VỤ WEB THAO TÁC VỚI DỮ LIỆU XML VÀ JSON

1. GIỚI THIỆU VỀ CÁC DỊCH VỤ WEB

1.1. Dịch vụ Web là gì?

Dịch vụ Web (Web Service) được coi là một công nghệ mang đến cuộc cách mạng trong cách thức hoạt động của các dịch vụ B2B (Business to Business) và B2C (Business to Customer). Giá trị cơ bản của dịch vụ Web dựa trên việc cung cấp các phương thức theo chuẩn trong việc truy nhập đối với hệ thống đóng gói và hệ thống kế thừa. Các phần mềm được viết bởi những ngôn ngữ lập trình khác nhau và chạy trên những nền tảng khác nhau có thể sử dụng dịch vụ Web để chuyển đổi dữ liệu thông qua mạng Internet theo cách giao tiếp tương tự bên trong một máy tính. Tuy nhiên, công nghệ xây dựng dịch vụ Web không nhất thiết phải là các công nghệ mới, nó có thể kết hợp với các công nghệ đã có như XML, SOAP, WSDL, UDDI... Với sự phát triển và lớn mạnh của Internet, dịch vụ Web thật sự là một công nghệ đáng được quan tâm để giảm chi phí và độ phức tạp trong tích hợp và phát triển hệ thống.

Theo định nghĩa của W3C (World Wide Web Consortium), dịch vụ Web là một hệ thống phần mềm được thiết kế để hỗ trợ khả năng tương tác giữa các ứng dụng trên các máy tính khác nhau thông qua mạng Internet, giao diện chung và sự gắn kết của nó được mô tả bằng XML. Dịch vụ Web là tài nguyên phần mềm có thể xác định bằng địa chỉ URL, thực hiện các chức năng và đưa ra các thông tin người dùng yêu cầu. Một dịch vụ Web được tạo nên bằng cách lấy các chức năng và đóng gói chúng sao cho các ứng dụng khác dễ dàng nhìn thấy và có thể truy cập đến những dịch vụ mà nó thực hiện, đồng thời có thể yêu cầu thông tin từ dịch vụ Web khác. Nó bao gồm các mô đun độc lập cho hoạt động của khách hàng và doanh nghiệp và bản thân nó được thực thi trên server.

Trước hết, có thể nói rằng ứng dụng cơ bản của Dịch vụ Web là tích hợp các hệ thống và là một trong những hoạt động chính khi phát triển hệ thống. Trong hệ thống này, các ứng dụng cần được tích hợp với cơ sở dữ liệu (CSDL) và các ứng dụng khác, người sử dụng sẽ giao tiếp với CSDL để tiến hành phân tích và lấy dữ liệu. Trong thời gian gần đây, việc phát triển mạnh mẽ của thương mại điện tử và B2B cũng đòi hỏi các hệ thống phải có khả năng tích hợp với CSDL của các đối tác kinh doanh.

Các Web Services cho phép các tổ chức thực hiện truyền thông dữ liệu mà không cần phải có kiến thức về hệ thống IT phía sau tường lửa. Một số Web Services hiện nay có sẵn miễn phí và càng ngày càng hướng dần vào các doanh nghiệp.

Một ví dụ về Web Service sẵn có là dịch vụ được cung cấp bởi PayPal cho phép những người có tài khoản có thể thanh toán hoặc trả một phần hoặc thực hiện các giao dịch tìm kiếm, và lấy lại các thông tin của từng giao dịch cụ thể.

1.2. Các chuẩn dịch vụ WEB

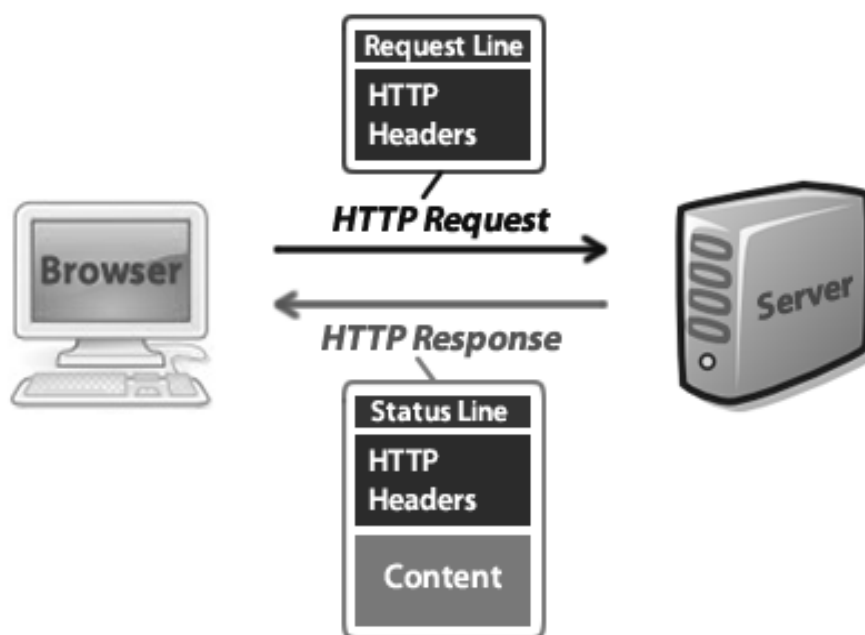
Việc tìm hiểu về các chuẩn dịch vụ WEB giúp chúng ta có thể đưa ra các lựa chọn phù hợp với hệ thống triển khai, dữ liệu và các thiết bị truy xuất dữ liệu đầu cuối đầu cuối. Có thể liệt kê một số chuẩn chính sau:

- UDDI (Universal Description, Discovery and Integrated).
- WSDL (Web Services Description Language).
- WSIL (Web Services Inspection Language).
- SOAP (Simple Object Access Protocol).

2. CÁC LOẠI WEBSERVICE

2.1. Giao thức HTTP

- HTTP (Hypertext Transfer Protocol) là giao thức mạng cho phép các hệ thống thông tin phân phối và cộng tác với nhau. HTTP là nền tảng giao tiếp dữ liệu cho WWW.
- HTTP hoạt động trên cơ chế giao thức request – response trong mô hình điện toán client – server.



Hình 2.1. Minh họa cơ chế hoạt động của giao thức HTTP

- Các định nghĩa đi kèm:
 - URI (Uniform Resource Identifier) là một chuỗi để xác định một tài nguyên trên internet.
 - URL (Uniform Resource Locator) là một URI cho biết sự tồn tại của một tài nguyên và cách thức để nhận tài nguyên đó.
 - URN (Uniform Resource Name) là một URI nhằm xác định tài nguyên bằng tên và độc lập với vị trí lưu trữ.
- HTTP Status code :
 - 2xx Success (Trạng thái thành công)
Ví dụ :
 - 200 – OK.
 - 201 – Created.
 - 3xx Redirection (Báo chuyển hướng)
Ví dụ :
 - 304 – Not Modified
 - 4xx Client Error (Báo lỗi client)
Ví dụ :
 - 403 Forbidden
 - 404 – Not Found
 - 5xx Server Error (Báo lỗi server)
Ví dụ :
 - 503 – Service Unavailable
 - 504 – Gateway Timeout

- **Ví dụ về HTTP – Request**

```
GET http://www.fit.hcmus.edu.vn/vn/ HTTP/1.1
Accept: application/x-ms-application, image/jpeg,
application/xaml+xml, image/gif, image/pjpeg, application/x-ms-xbap,
application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-
powerpoint, application/msword,
Accept-Language: vi-VN
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1;
WOW64; Trident/4.0; GTB6.5; Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1; SV1) ; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR
3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; OfficeLiveConnector.1.5;
OfficeLivePatch.1.3; InfoPath.3; AskTbGGSV5/5.8.0.12217)
chrome/6.0.472.63
Accept-Encoding: gzip, deflate Connection: Keep-Alive
Host: www.fit.hcmus.edu.vn
Cookie:
```

```
.ASPXANONYMOUS=MG6LCiuSywEAAAANTA2ZWNiYTAtYThiNy00MDA1LTkyN  
jUtYT1lYzAxNTA3MTU10
```

- Ví dụ về HTTP – Response

```
HTTP/1.0 200 OK  
Cache-Control: private  
Content-Type: text/html; charset=utf-8  
Server: Microsoft-IIS/7.0  
X-AspNet-Version: 2.0.50727  
Set-Cookie: DotNetNukeAnonymous=7db3c001-c407-4adb-a60f-053b5dc76dc2;  
expires=Thu, 30-Sep-2010 03:12:33 GMT; path=/; HttpOnly  
Set-Cookie: language=vi-VN; path=/; HttpOnly X-Powered-By: ASP.NET  
Date: Thu, 30 Sep 2010 02:52:33 GMT Content-Length: 24404  
X-Cache: MISS from vweb.hcmuns.edu.vn  
Via: 1.0 vweb.hcmuns.edu.vn:80 (squid/2.6.STABLE16) Connection: keep-alive  
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"> ...  
// Nội dung trang web
```

- Các phương thức chính của HTTP



Hình 2.1. Các phương thức chính của HTTP

- Phương thức GET:
 - Dùng để tải một biểu diễn của tài nguyên.
 - Được sử dụng nhiều nhất.
 - Biểu diễn của tài nguyên có thể bao gồm HTML, JPG, XML, ...
- Phương thức HEAD:
 - Tương tự như GET thay vì tải toàn bộ thể hiện thì chỉ tải HTTP Header.
 - Dùng để kiểm tra thay đổi khi muốn tải lại tài nguyên có kích thước lớn.
- Phương thức DELETE:
 - Dùng để xóa tài nguyên.

- Phương thức PUT :
 - Dùng để lưu một biểu diễn vào một tài nguyên.
 - Đăng tải một tập tin vào một vị trí xác định trên website.
 - Nếu PUT thực hiện hai lần thì lần sau sẽ đè lên lần trước.
- Phương thức POST :
 - Tạo một tài nguyên tương tự như PUT, nhưng server sẽ quyết định cách lưu trữ thay vì client như PUT.

2.2. SOAP

- SOAP - Simple Object Access Protocol: là một giao thức giao tiếp có cấu trúc như XML và mã hóa thành định dạng chung cho các ứng dụng trao đổi với nhau.
- SOAP là một đặc tả việc sử dụng các tài liệu XML theo dạng các thông điệp. Bản thân SOAP không định ra các ngữ nghĩa ứng dụng hoặc cách cài đặt chi tiết. SOAP cung cấp một cơ chế đơn giản và gọn nhẹ cho việc trao đổi thông tin có cấu trúc và định dạng giữa các thành phần trong một môi trường phân tán sử dụng XML. SOAP được thiết kế dựa trên những chuẩn nhằm giảm chi phí tích hợp các hệ thống phân tán xây dựng trên nhiều nền tảng khác nhau ở mức càng thấp càng tốt. Đặc tả về SOAP định nghĩa một mô hình trao đổi dữ liệu dựa trên 3 khái niệm cơ bản: Các thông điệp là các tài liệu XML, chúng được truyền đi từ bên gửi đến bên nhận, bên nhận có thể chuyển tiếp dữ liệu đến nơi khác.
- Khái niệm cơ bản nhất của mô hình SOAP là việc sử dụng các tài liệu XML như những thông điệp trao đổi. Điều này có nhiều ưu điểm hơn các giao thức truyền dữ liệu khác. Các thông điệp XML có thể được tổng hợp và đọc với một bộ soạn thảo text đơn giản, ta có thể làm việc với XML trên hầu hết mọi nền tảng.
- Đặc trưng của SOAP:
 - SOAP được thiết kế đơn giản và dễ mở rộng.
 - Tất cả các message SOAP đều được mã hóa sử dụng XML.
 - SOAP sử dụng giao thức truyền dữ liệu riêng.
 - Không có garbage collection phân tán, và cũng không có cơ chế tham chiếu. Vì thế SOAP client không giữ bất kỳ một tham chiếu đầy đủ nào về các đối tượng ở xa.
 - SOAP không bị ràng buộc bởi bất kỳ ngôn ngữ lập trình nào hoặc công nghệ nào.

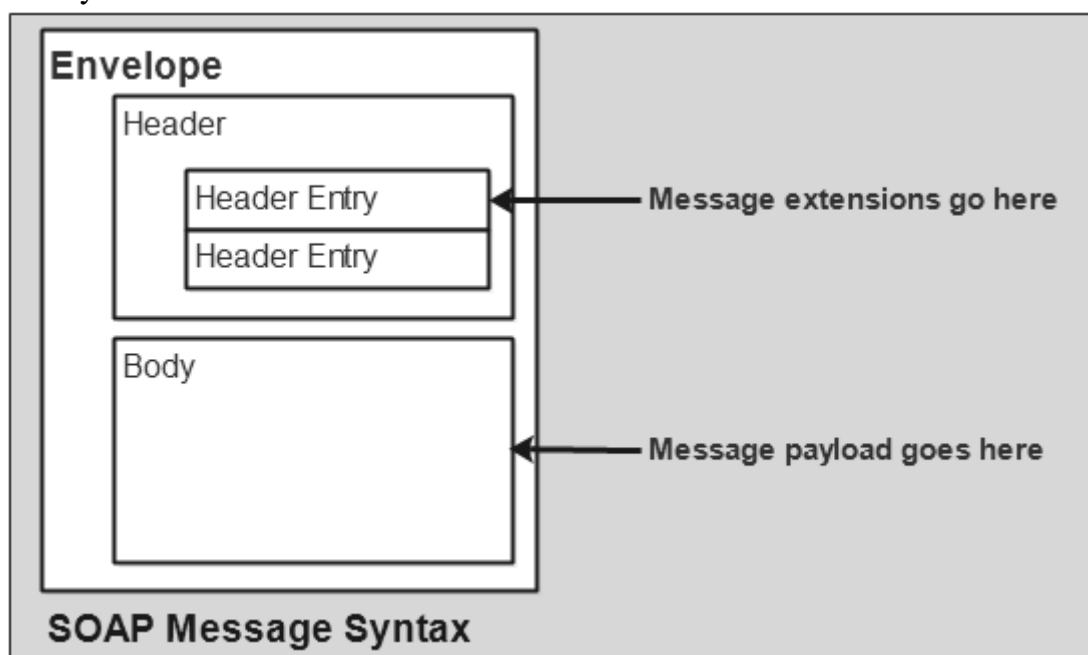
Vì những đặc trưng này, nó không quan tâm đến công nghệ gì được sử dụng để thực hiện miễn là người dùng sử dụng các message theo định dạng XML. Tương tự, service có thể được thực hiện trong bất kỳ ngôn ngữ nào, miễn là nó có thể xử lý được những message theo định dạng XML.
- Khi trao đổi các thông điệp SOAP, có hai thành phần liên quan: Bên gửi và bên nhận. Thông điệp sẽ được chuyển từ bên gửi sang bên nhận. Đây là ý niệm đơn giản nhất trong trao đổi thông điệp SOAP. Trong nhiều trường hợp, kiểu trao đổi này không

cung cấp đủ chức năng. Nhưng đây là mô hình cơ bản, dựa trên đó sẽ phát triển các mô hình trao đổi phức tạp hơn.



Hình 2.2. Hệ thống Soap đơn giản.

- Một cấu trúc SOAP được định nghĩa gồm các phần: <Envelope>, <Header> và <Body>.



Hình 2.3. Cấu trúc thông điệp SOAP.

2.3. REST

- REST (Representational State Transfer) đã được chọn sử dụng rộng rãi thay cho Web service dựa trên SOAP và WSDL. Bằng chứng quan trọng của sự thay đổi này chính là việc các công ty dẫn đầu trong lĩnh vực cung cấp dịch vụ mạng 2.0 như Yahoo, Google và Facebook đã phản đối các giao thức dựa trên SOAP hoặc WSDL và ủng hộ phương thức hướng đến tài nguyên và dễ sử dụng đối với các dịch vụ của họ. Trong bài viết này, chúng ta sẽ tìm hiểu các nguyên lý cơ bản của REST.
- REST (Representational State Transfer) là một kiểu kiến trúc phần mềm cho các hệ thống phân tán siêu truyền thông như là WWW.
- Đặc trưng của REST
 - o Là dạng client – server.
 - o Phân tách giao diện của client ra khỏi dữ liệu.

- Cho phép mỗi thành phần phát triển độc lập.
- Hỗ trợ đa nền tảng.
- Mỗi yêu cầu từ client phải có đủ thông tin cần thiết để server có thể hiểu được mà không cần phải lưu trữ thêm thông tin nào trước đó.
- Tất cả tài nguyên được truy cập thông qua một interface thống nhất (HTTP GET, PUT, POST, DELETE, ...).

2.4. RESTful Service

- Là một web service đơn giản sử dụng HTTP và tính chất của REST.
- Là một tập tài nguyên các thành phần được định nghĩa:
 - URI gốc cho web service.
 - MIME type hỗ trợ bởi web service.
 - Tập hành động hỗ trợ bởi web service sử dụng HTTP method (GET, POST, PUT, DELETE).

3. XÂY DỰNG ỨNG DỤNG KẾT NỐI DỊCH VỤ WEB RESTFUL

Ứng dụng trực tuyến trên thiết bị di động được xây dựng nhằm mục đích hiển thị dữ liệu đầu cuối được xử lý từ các máy chủ cũng như gửi các yêu cầu về truy xuất dữ liệu. Ví dụ: ứng dụng Google Play Store được Google cài đặt trên thiết bị người dùng, hiển thị các ứng dụng được phát triển bởi các lập trình viên, cho phép người dùng có thể tải và cài đặt các ứng dụng đó ngay trên thiết bị của họ.

Trong phần hướng dẫn này ta sẽ truy xuất đến OnlineShop Service, một dịch vụ chuyên cung cấp các thông tin về các mặt hàng sản phẩm như giá cả, chất liệu, tính năng...

3.1. Khai báo và kiểm soát các yêu cầu kết nối

- Việc đầu tiên cần thực hiện là xin quyền truy cập Internet cho ứng dụng, có thể bổ sung các quyền có liên quan như kiểm soát kết nối Internet.

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

- Trong việc truy xuất tài nguyên Internet trên thiết bị di động cần kiểm soát được các vấn đề về kết nối. Ví dụ: ta cần kiểm soát việc cho phép người dùng tương tác với ứng dụng ở chế độ ngoại tuyến và thực hiện cập nhật dữ liệu khi có kết nối. Có thể xem ví dụ về việc sử dụng Broadcast Receiver để kiểm soát vấn đề này:

```
BroadcastReceiver mNetworkStateIntentReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d("HTSI", "CONNECTION CHANGED");
        ConnectivityManager cm = ((ConnectivityManager) context
            .getSystemService(Context.CONNECTIVITY_SERVICE));
        if (cm.getActiveNetworkInfo() != null) {
```

```

        // Thực hiện xử lý khi không có kết nối.
    } else { // Và khi không có kết nối.}
    }
};

```

3.2. Thực hiện kết nối

- Việc kết nối được thông qua 2 cách: sử dụng kết nối URL thông thường hoặc sử dụng kết nối thông qua thư viện của gói org.apache.http.
- Sử dụng kết nối URL thông thường:

```

HttpURLConnection connection = null;
Url url = new URL("http://221.132.37.130:6969/OnlineShopWebService");
connection = (HttpURLConnection) url.openConnection();
connection.setConnectTimeout(10000);
connection.setReadTimeout(10000);
connection.setRequestMethod("GET");

```

- Ở phương thức ***setRequestMethod*** ta có thể tự do thêm vào tên của cách thức gọi phương thức từ dịch vụ RESTful như GET, POST, PUT, ... Tùy thuộc vào loại phương thức mà dịch vụ đó cung cấp. Ở ví dụ trên, do thực hiện gọi theo loại phương thức GET, cũng như không có tham số truyền vào nên không khác mấy so với cách thức mà chúng ta thực hiện ở phần đầu chương.
- Tuy nhiên hãy khảo sát cách thức để đăng nhập vào dịch vụ này. Ở đây, cần thực hiện gửi dữ liệu người dùng cho dịch vụ theo dạng JSON và dạng sử dụng POST để bảo mật dữ liệu.

```

HttpURLConnection connection = null;
Url url = new URL("http://221.132.37.130:6969/OnlineShopWebService");
Conn-ection = (HttpURLConnection) url.openConnection();
connection.setConnectTimeout(10000);
connection.setReadTimeout(10000);
connection.setRequestMethod("POST");
connection.setRequestProperty("Content-Type", "application/json");
connection.setDoOutput(true);
connection.setInstanceFollowRedirects(true);
// Gửi dữ liệu thông qua OutputStream
OutputStream outputStream = new
    BufferedOutputStream(connection.getOutputStream());
outputStream.write(incomingParams.getBytes());
outputStream.flush();
outputStream.close();

```

- Dữ liệu ***incomingParams*** được thực hiện chia thành từng byte nhỏ bằng phương thức `getBytes()` trước khi được đưa vào `OutputStream` của kết nối. Dữ liệu được định dạng là JSON khi dịch vụ nhận được thông qua phương thức `setRequestProperty()`.
- Sử dụng các lớp trong gói **org.apache.http**:
 Đối với cách sử dụng các lớp HTTP trong Apache giúp chúng ta tường minh hơn việc gọi đến Service. Ví dụ sử dụng hai lớp `HttpGet` và `HttpPost`:
 - **HttpGet**: sau khi thực hiện có thể thực hiện truy xuất dữ liệu dạng `InputStream` thông qua phương thức `getEntity` đối tượng `HttpResponse`.

```
public static InputStream getInputStreamFromUrl(String url) {
    InputStream content = null;
    try {
        HttpGet httpGet = new HttpGet(url));
        HttpClient httpClient = new DefaultHttpClient();
        HttpResponse response = httpClient.execute(httpGet);
        content = response.getEntity().getContent();
    } catch (Exception e) {
        Log.d("[GET REQUEST]", "Network exception" + e);
    }
    return content;
}
```

- **HttpPost**: đối với POST cần xác định dạng dữ liệu cần thực hiện gửi lên dịch vụ, ví dụ sau đây mô tả cách thức gửi dữ liệu dạng JSON, được đóng gói vào đối tượng `NameValuePair`. Dữ liệu được trả về vẫn được truy xuất thông qua đối tượng `HttpResponse`.

```
HttpClient httpClient = new DefaultHttpClient();
HttpPost httpPost = new HttpPost(URL_CONNECT);
JSONObject jsonObject = ModuleCore.moduleCoreToJson(ModuleCore.LOGIN, params);
StringBuilder builder = new StringBuilder();
List<NameValuePair> nvps = new ArrayList<NameValuePair>();
nvps.add(new BasicNameValuePair("jsonParam", jsonObject.toString()));
httpPost.setEntity(new UrlEncodedFormEntity(nvps, "UTF-8"));
HttpResponse httpResponse = httpClient.execute(httpPost);
```

3.3. Truy xuất dữ liệu trả về

- Việc truy xuất dữ liệu không phụ thuộc vào cách thức gọi dịch vụ cho lắm, bởi dữ liệu luôn là `InputStream`, chúng ta chỉ cần thực đọc dòng dữ liệu này và chuyển đổi thành dạng dữ liệu tương ứng cần tương tác. Hãy xem ví dụ về cách thức đọc dữ liệu `InputStream` thành `String`:

```
HttpEntity httpEntity = httpResponse.getEntity();
InputStream inputStream = httpEntity.getContent();
BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
String line;
while ((line = reader.readLine()) != null) {
    builder.append(line);
}
inputStream.close();
```

4. ĐỌC GHI DỮ LIỆU XML

4.1. Định dạng XML

- XML (viết tắt từ tiếng Anh eXtensible Markup Language, "Ngôn ngữ Đánh dấu Mở rộng") là ngôn ngữ đánh dấu với mục đích chung do W3C đề nghị, để tạo ra các ngôn ngữ đánh dấu khác. Đây là một tập con đơn giản của SGML, có khả năng mô tả nhiều loại dữ liệu khác nhau. Mục đích chính của XML là đơn giản hóa việc chia sẻ dữ liệu giữa các hệ thống khác nhau, đặc biệt là các hệ thống được kết nối với Internet. Các ngôn ngữ dựa trên XML (thí dụ: RDF, RSS, MathML, XHTML, SVG, GML và cXML) được định nghĩa theo cách thông thường, cho phép các chương trình sửa đổi và kiểm tra hợp lệ bằng các ngôn ngữ này mà không cần có hiểu biết trước về hình thức của chúng.
- Cú pháp sơ lược:

```
<ten thẻ thuộc tính= "Giá trị">Nội dung thẻ</ten thẻ>
```

- Ví dụ:

```
<data>
  <weather>
    <date>2014-06-20</date>
    <tempMaxC>13</tempMaxC>
    <tempMaxF>56</tempMaxF>
    <tempMinC>5</tempMinC>
    <tempMinF>42</tempMinF>
    <windspeedMiles>13</windspeedMiles>
    <windspeedKmph>20</windspeedKmph>
```

```
<winddirection>WNW</winddirection>
<weatherCode>113</weatherCode>
<precipMM>0.0</precipMM>
</weather>
</data>
```

4.2. Đọc ghi dữ liệu XML

- Để thực hiện đọc dữ liệu từ XML trong Android, ta có thể sử dụng DOM (Document Object Model) Parser hoặc XML Pull Parser.

4.2.1. Ghi dữ liệu XML

- **DOM Parser**: giao diện lập trình ứng dụng (API) có dạng một cây cấu trúc dữ liệu, các đối tượng cần khởi tạo khi sử dụng:
 - o **Element**: đại diện cho một thẻ trong XML.
 - o **Document**: tập tin tài liệu được khởi tạo từ dữ liệu XML thông qua **DocumentBuilder**.
 - o **DocumentBuilder**: đối tượng hỗ trợ chuyển đổi dữ liệu XML thành cấu trúc tập tin XML cho việc đọc ghi dữ liệu.
 - o **DocumentBuilderFactory**: khởi tạo đối tượng **DocumentBuilder**.
 - o **Transformer**: đối tượng cho phép thực hiện chuyển đổi dữ liệu sang nhiều dạng XML khác nhau.
 - o **TransformerFactory**: khởi tạo đối tượng **Transformer**.
 - o **DOMSource**.
- Ví dụ: thực hiện tổ chức ghi dữ liệu XML

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.newDocument();
```

```
// Khởi tạo thẻ gốc <data>
```

```
Element rootElement = document.createElement("data");
document.appendChild(rootElement);
```

```
// Khởi tạo thẻ <weather>
```

```
Element weather = document.createElement("weather");
// Thiết lập <weather> là thẻ con trong thẻ <data>
rootElement.appendChild(weather);
```

```
// Khởi tạo thẻ <date>
```

```
Element date = document.createElement("date");
// Khởi tạo giá trị thẻ <date>
date.appendChild(document.createTextNode("20-06-2014"));
// Thiết lập <date> là thẻ con trong thẻ <weather>
weather.appendChild(date);
```

```

Element tempMaxC = document.createElement("tempMaxC");
tempMaxC.appendChild(document.createTextNode("13"));
weather.appendChild(tempMaxC);

Element tempMaxF = document.createElement("tempMaxF");
tempMaxF.appendChild(document.createTextNode("56"));
weather.appendChild(tempMaxF);

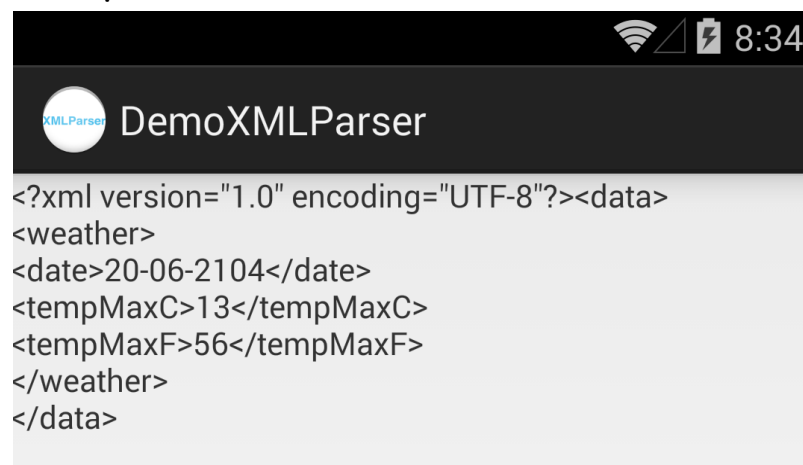
// Định dạng các thuộc tính dữ liệu XML
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
Properties outFormat = new Properties();
outFormat.setProperty(OutputKeys.INDENT, "yes");
outFormat.setProperty(OutputKeys.METHOD, "xml");
outFormat.setProperty(OutputKeys.OMIT_XML_DECLARATION, "no");
outFormat.setProperty(OutputKeys.VERSION, "1.0");
outFormat.setProperty(OutputKeys.ENCODING, "UTF-8");
transformer.setOutputProperties(outFormat);

// Chuyển đổi nguồn dữ liệu theo chuẩn DOM
DOMSource domSource = new DOMSource(document.getDocumentElement());
OutputStream output = new ByteArrayOutputStream();
StreamResult result = new StreamResult(output);
transformer.transform(domSource, result);

String xmlString = output.toString();
textXML.setText(xmlString);

```

- Kết quả của dữ liệu XML:



Hình 2.4.

- **XML Pull Parser:** cho phép trình bày các thành phần trong tập tin theo dạng chuỗi các thẻ (tag), việc ghi dữ liệu XML được biểu diễn thông qua hai đối tượng *XmlSerializer* và *StringWriter*.

```
// Khởi tạo đối tượng XmlSerializer
XmlSerializer xmlSerializer = Xml.newSerializer();
// Khởi tạo đối tượng StringWriter
StringWriter stringWriter = new StringWriter();

xmlSerializer.setOutput(stringWriter);
xmlSerializer.startDocument("UTF-8", true);
xmlSerializer.startTag("", "data");
xmlSerializer.attribute("", "source", "weather.t3h.vn");
xmlSerializer.startTag("", "weather");

xmlSerializer.startTag("", "date");
xmlSerializer.text("20-06-2014");
xmlSerializer.endTag("", "date");

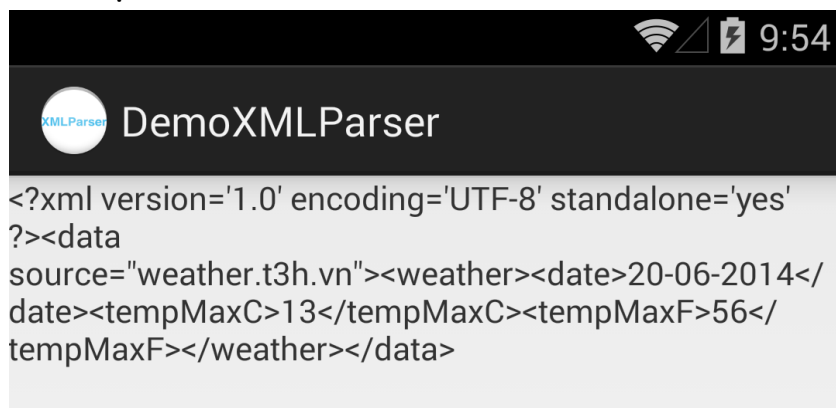
xmlSerializer.startTag("", "tempMaxC");
xmlSerializer.text("13");
xmlSerializer.endTag("", "tempMaxC");

xmlSerializer.startTag("", "tempMaxF");
xmlSerializer.text("56");
xmlSerializer.endTag("", "tempMaxF");

xmlSerializer.endTag("", "weather");
xmlSerializer.endTag("", "data");
xmlSerializer.endDocument();

textXML.setText(stringWriter.toString());
```

- Kết quả của dữ liệu XML:



Hình 2.5

4.2.2. Đọc dữ liệu XML

- **DOM Parser:** giao diện lập trình ứng dụng (API) có dạng một cây cấu trúc dữ liệu, các đối tượng cần khởi tạo khi sử dụng:
 - o **Element:** đại diện cho một thẻ trong XML.
 - o **NodeList:** đại diện cho một thẻ có chứa nhiều thẻ con.

- **Document**: tập tin tài liệu được khởi tạo từ dữ liệu XML thông qua **DocumentBuilder**.
- **DocumentBuilder**: đối tượng hỗ trợ chuyển đổi dữ liệu XML thành cấu trúc tập tin XML cho việc đọc ghi dữ liệu.
- **DocumentBuilderFactory**: khởi tạo đối tượng **DocumentBuilder**.

sample_xml.xml:

```
<data>
  <weather>
    <date>2014-06-20</date>
    <tempMaxC>13</tempMaxC>
    <tempMaxF>56</tempMaxF>
    <tempMinC>5</tempMinC>
    <tempMinF>42</tempMinF>
    <windspeedMiles>13</windspeedMiles>
    <windspeedKmph>20</windspeedKmph>
    <winddirection>WNW</winddirection>
    <weatherCode>113</weatherCode>
    <precipMM>0.0</precipMM>
  </weather>
</data>
```

- Ví dụ xử lý XML với DOM: truy xuất dữ liệu thẻ `<weather>` trong đoạn XML trên.

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
try {
    InputStream inputStream = getAssets().open("sample_xml.xml");
    DocumentBuilder db = dbf.newDocumentBuilder();
    Document document = db.parse(inputStream);
    Element element = document.getDocumentElement();
    NodeList nodeList = element.getElementsByTagName("weather");
    if (nodeList == null)
        return;
    for (int i=0; i<nodeList.getLength(); i++) {
        Element weather = (Element) nodeList.item(i);
        for (int x=0; x < weather.getChildNodes().getLength(); x++) {
```



```

        Node weatherNode = weather.getChildNodes().item(x);
        Log.d("HTSI",weatherNode.getNodeName()+"="+weatherNode.getTextContent());
    }
}
} catch (ParserConfigurationException e) {
    e.printStackTrace();
} catch (SAXException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();}

```

- **XML Pull Parser:** cho phép trình bày các thành phần trong tập tin theo dạng chuỗi các thẻ (tag) và các đánh dấu (event), để làm việc với XML Pull Parser cần khảo sát các thuộc tính và các đối tượng sau:
 - **XmlPullParserFactory:** khởi tạo đối tượng XmlPullParser từ tập tin tài liệu XML.
 - **XmlPullParser:** đối tượng kiểm soát việc duyệt và truy xuất dữ liệu.
 - **START_DOCUMENT:** điểm đánh dấu bắt đầu của tập tin XML.
 - **END_DOCUMENT:** điểm đánh dấu kết thúc của tập tin XML.
 - **START_TAG:** điểm đánh dấu bắt đầu cặp thẻ XML.
 - **END_TAG:** điểm đánh dấu kết thúc cặp thẻ XML.
- Ví dụ xử lý XML với DOM: truy xuất dữ liệu các thẻ con trong thẻ <weather> đoạn XML trên.

```

InputStream inputStream = getAssets().open("sample_xml.xml");
XmlPullParserFactory xmlPullParserFactory = XmlPullParserFactory.newInstance();
XmlPullParser xpp = xmlPullParserFactory.newPullParser();
xpp.setInput(inputStream, null);
int eventType = xpp.getEventType();
while (eventType != XmlPullParser.END_DOCUMENT) {
    if (eventType == XmlPullParser.START_TAG) {
        String tagName = xpp.getName();
        // Bỏ qua dữ liệu của thẻ data & weather
        if (!tagName.equalsIgnoreCase("data") &&
            !tagName.equalsIgnoreCase("weather"))
            Log.d("HTSI", tagName + " = " + xpp.nextText());
    }
}

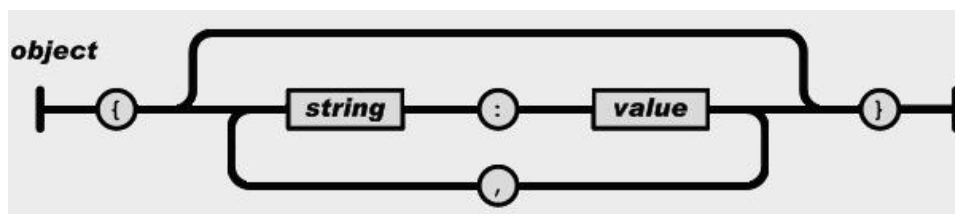
```

```
} eventType = xpp.next();
}
```

5. ĐỌC GHI DỮ LIỆU JSON

5.1. Định dạng JSON

- JSON (JavaScript Object Notation) được định nghĩa dựa theo ngôn ngữ JavaScript, tiêu chuẩn ECMA-262 năm 1999, cấu trúc là một định dạng văn bản đơn giản với các trường dữ liệu được lồng vào nhau. JSON được sử dụng để trao đổi dữ liệu giữa các thành phần của một hệ thống tương thích với hầu hết các ngôn ngữ C, C++, C#, Java, JavaScript, Perl, Python...
- JSON được xây dựng dựa trên hai cấu trúc chính:
 - Tập hợp cặp giá trị **name/value**, trong nhiều ngôn ngữ khác nhau cặp giá trị này có thể là **object**, **record**, **struct**, **dictionary**, **hash table**, **keyed list**...
 - Tập hợp danh sách các giá trị, có thể là **array**, **vector**, **list** hay **sequence**.
- Tùy thuộc vào dữ liệu cần trao đổi, JSON có thể có nhiều dạng khác nhau, tuy nhiên có thể tổng hợp ở những hai dạng chính sau:
 - Một đối tượng Object chứa các cặp giá trị string/value không cần thứ tự, được bao trong cặp "{}", các giá trị bên trong được định dạng "string:value" và chia cách nhau bởi dấu ",". Value ở đây có thể là chuỗi, số, true- false, null...Có thể xem mô tả cùng ví dụ sau:

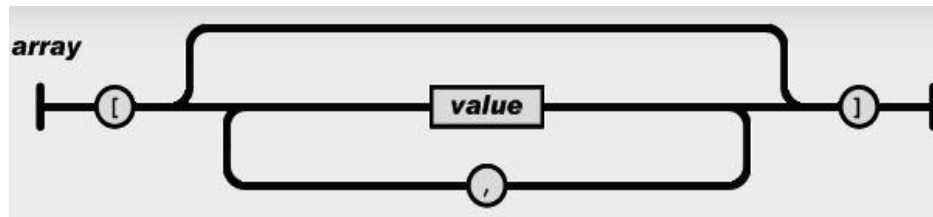


Hình 2.6. Minh họa cấu trúc dữ liệu JSON 1

- Ví dụ:

```
{"date":"20-06-2014", "tempMaxC":13, "tempMaxF":56}
```

- Một đối tượng mảng có bao gồm nhiều phần tử con có thứ tự. Các phần tử con được bao trong cặp "[]" và chia cách nhau bởi dấu ",". Mỗi phần tử con có thể là một giá trị đơn lẻ như: số, chuỗi, true-false, null hoặc một object khác, thậm chí có thể là một mảng.



Hình 2.7. Minh họa cấu trúc dữ liệu JSON 2

- Ví dụ 1:

```
[
  {
    "source": "weather.t3h.vn",
    "data": [
      {
        "date": "20-06-2014",
        "tempMaxC": 13,
        "tempMaxF": 56
      },
      {
        "date": "21-06-2014",
        "tempMaxC": 20,
        "tempMaxF": 60
      }
    ]
  }
]
```

- Ví dụ 2:

```
{
  "product": "Google Nexus 5",
  "color": [
    "grey", "white", "black"
  ],
  "memory": [
    16, 32, 64
  ]
}
```

5.2. Đọc ghi dữ liệu JSON

- Việc thực hiện đọc ghi dữ liệu JSON trong Android có thể thông qua nhiều thư viện khác nhau như GSON, Json.Smart, Jackson, ... Tuy nhiên, trong tài liệu này, chúng ta sẽ khảo sát các lớp JSON trong gói **org.json** được tích hợp sẵn trong Android SDK. Trong gói này bao gồm bốn lớp chính:
 - **JSONObject**: đối tượng quản lý JSON ở dạng một Object.
 - **JSONArray**: đối tượng quản lý JSON ở dạng tập hợp các Object hoặc Array.
 - **JSONStringer**: đối tượng chuyển dữ liệu JSON thành dạng chuỗi.
 - **JSONTokener**: chuyển đổi đối tượng JSON (chuẩn RFC-4627) mã hoá chuỗi một thành đối tượng tương ứng.

5.2.1. Ghi dữ liệu JSON

- Để thực hiện ghi dữ liệu JSON, cần xác định rõ cấu trúc của dữ liệu cần lưu trữ. Nếu dữ liệu cần ghi là một đối tượng, dữ liệu sẽ được ghi vào một JSONObject; nếu dữ liệu là một mảng, dữ liệu sẽ được ghi vào một JSONArray.
- Ví dụ 1: ghi dữ liệu có cấu trúc đơn giản dạng JSONObject:

```
{
    "date": "17-06-2015",
    "tempMaxC": 13,
    "tempMaxF": 56
}
// Khởi tạo đối tượng JSONObject
JSONObject jsonObject = new JSONObject();

// Thiết lập các cặp giá trị name/value
jsonObject.put("date", "17-06-2015");
jsonObject.put("tempMaxC", 13);
jsonObject.put("tempMaxF", 56);

// Chuyển dữ liệu thành chuỗi để sử dụng
String jsonString = jsonObject.toString();
```

- Ví dụ 2: ghi dữ liệu có cấu trúc dạng JSONArray.

```
// Khởi tạo đối tượng JSONArray
JSONArray jsonArray = new JSONArray();

// Khởi tạo và đặt giá trị đối tượng JSONObject "source"
JSONObject jsonObjectSource = new JSONObject();
jsonObjectSource.put("source", "weather.t3h.vn");

// Khởi tạo đối tượng JSONArray "data"
JSONArray jsonArrayData = new JSONArray();

// Khởi tạo và đặt giá trị phần tử JSONObject 1
JSONObject jsonObjectDate1 = new JSONObject();
jsonObjectDate1.put("date", "17-06-2015");
jsonObjectDate1.put("tempMaxC", 13);
jsonObjectDate1.put("tempMaxF", 56);

// Khởi tạo và đặt giá trị phần tử JSONObject 2
JSONObject jsonObjectDate2 = new JSONObject();
jsonObjectDate2.put("date", "18-06-2015");
jsonObjectDate2.put("tempMaxC", 20);
jsonObjectDate2.put("tempMaxF", 60);
```

```
// Đặt giá trị vào mảng "data"
jsonArrayData.put(jsonObjectDate1);
jsonArrayData.put(jsonObjectDate2);

// Đặt giá trị mảng vào "data"
JSONObject jsonObjectData = new JSONObject();
jsonObjectData.put("data", jsonArrayData);

// Đặt giá trị vào mảng
jsonArray.put(jsonObjectSource);
jsonArray.put(jsonObjectData);

// Xuất giá trị mảng
Log.d("HTSI", jsonArray.toString());
```

5.2.2. Đọc dữ liệu JSON

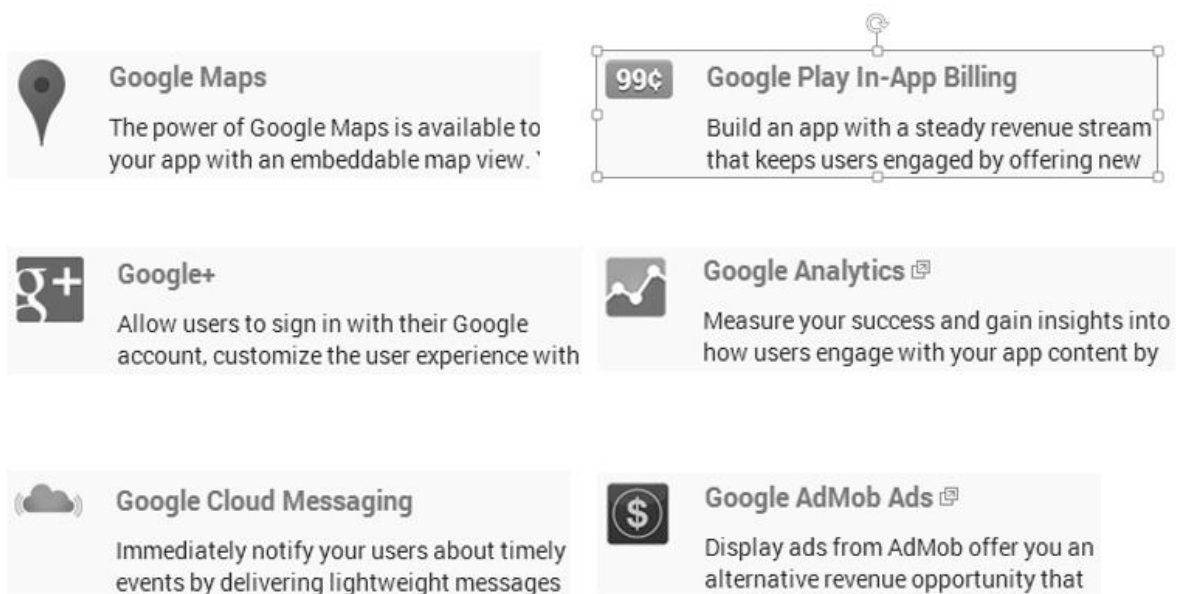
- Việc đọc dữ liệu JSON khá gần giống với việc ghi, chỉ khác nhau việc đọc dữ liệu có nhận dữ liệu đầu vào.

Bài 3

GOOGLE MAP

1. GOOGLE PLAY SERVICE SDK

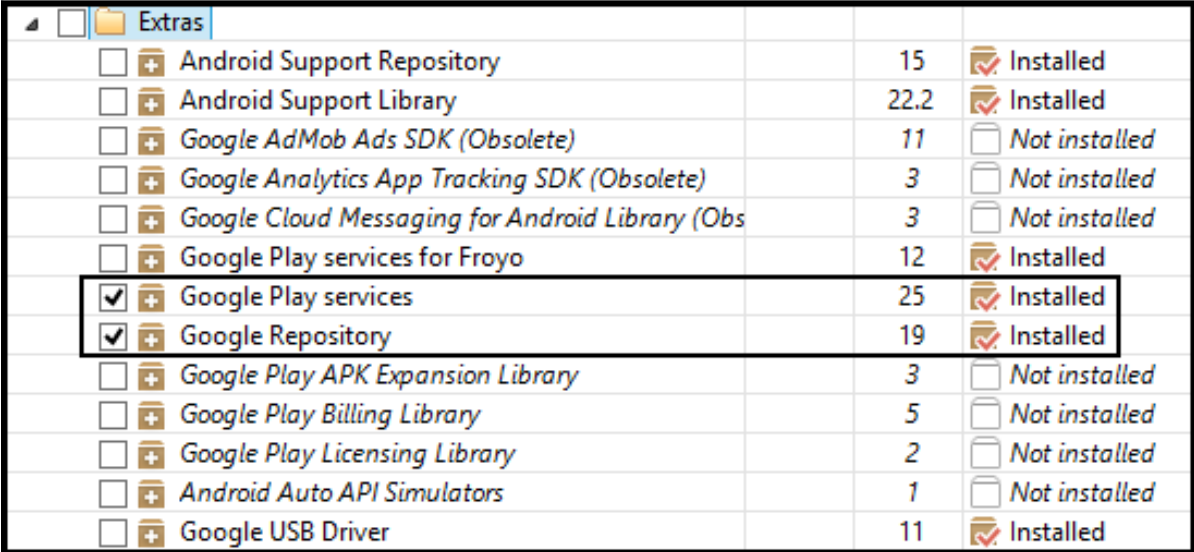
- Google Play Service SDK (GPSDK) là bộ công cụ phát triển tích hợp các dịch vụ Google đang sở hữu và phát triển vào trong các ứng dụng hoạt động trên các thiết bị di động.
- GPSDK là gói API riêng biệt với bộ Android SDK.
- Phiên bản Android, GPSDK chỉ hoạt động trên các thiết bị có version từ 2.2 trở lên.
- Một số dịch vụ trong GPSDK:



Hình 3.1. Một số dịch vụ trong GPSDK

- Cài đặt GPSDK: thực hiện cài đặt gói API từ công cụ Android SDK Manager





Package Name	Version	Status
Android Support Repository	15	Installed
Android Support Library	22.2	Installed
Google AdMob Ads SDK (Obsolete)	11	Not installed
Google Analytics App Tracking SDK (Obsolete)	3	Not installed
Google Cloud Messaging for Android Library (Obs)	3	Not installed
Google Play services for Froyo	12	Installed
Google Play services	25	Installed
Google Repository	19	Installed
Google Play APK Expansion Library	3	Not installed
Google Play Billing Library	5	Not installed
Google Play Licensing Library	2	Not installed
Android Auto API Simulators	1	Not installed
Google USB Driver	11	Installed

Hình 3.2. Cài đặt gói API từ công cụ Android SDK Manager

- GPSDK trên thiết bị, thường có 4 trạng thái:
 - o Google Play Store đã được cài đặt cùng các dịch vụ đi kèm (phổ biến).
 - o Google Play Store đã được cài đặt nhưng các dịch vụ chưa được cài đặt.
 - o Google Play Store chưa được cập nhật phiên bản mới nhất.
 - o Google Play Store không được cài đặt trên thiết bị (máy ảo).
 - o Google Play Store tự động cập nhật các dịch vụ khi có kết nối internet.
- Kiểm tra trạng thái của GPSDK:
 - o Thực hiện kiểm tra trong **onResume** bằng phương thức **isGooglePlayServiceAvailable()**.
 - o Kết quả trả về:
 - SUCCESS
 - SERVICE_MISSING
 - SERVICE_VERSION_UPDATE_REQUIRED
 - SERVICE_DISABLE
- Thực hiện gọi phương thức **getErrorDialog()** để hướng dẫn người cập nhật GPSDK hoặc kích hoạt trong phần Setting.

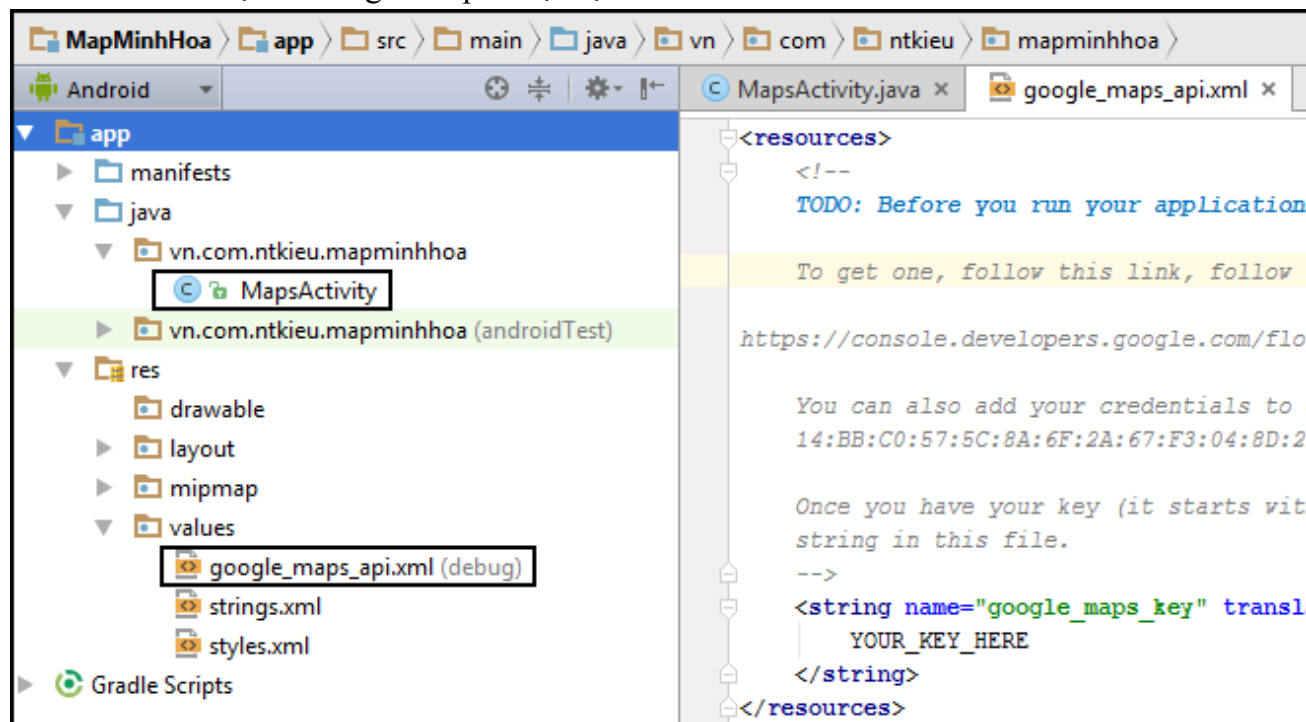
2. GOOGLE MAPS ANDROID API

- **Google Maps Android API (GMAA)** bao gồm các dữ liệu bản đồ được phát triển bởi Google Inc cho phép lập trình viên tích hợp vào các ứng dụng thông qua các phương thức được cung cấp sẵn.
- GMAA hỗ trợ các thao tác với giao diện đồ họa của bản đồ bao gồm:
 - o Vẽ biểu tượng trên bản đồ (Marker).
 - o Đồ họa đường thẳng (Polylines).
 - o Đồ họa hình đa giác (Polygons).
 - o Bitmap trên bản đồ (Ground & Tile Overlay).

2.1. Tạo dự án Google Maps (Google Maps project)

Để tạo dự án Google Maps, chúng ta tạo dự án Android nhưng ở hộp thoại “Add an activity to mobile”, chúng ta chọn **Google Maps Activity**.

- Sau khi dự án Google Maps được tạo sẽ có cấu trúc như sau:



Hình 3.3. Cấu trúc dự án Google Maps

Dự án Google Maps được tạo có chứa hai tập tin quan trọng mà chúng ta sẽ thao tác, chỉnh sửa với chúng là google_maps_api.xml và MapsActivity.java.

2.2. Google Maps Android API Key

- **Google Maps Android API Key:** chuỗi mã hóa được Google cung cấp miễn phí để quản lý và chứng thực việc truy xuất dữ liệu bản đồ trên ứng dụng.
- GMAA Key được liên kết thông qua **Digital Certificate** (DC-Chứng thư số) và **Package name** (Tên đóng gói) của ứng dụng.
- Tạo GMAA Key bao gồm 3 bước:
 - o Truy xuất thông tin DC bằng mã SHA-1.
 - o Đăng ký Project trong Google API Console.
 - o Tích hợp Google Map Service vào Project và gửi yêu cầu cấp GGMA Key.
- Cấu hình tập tin AndroidManifest: cần thực hiện khai báo một số quyền truy cập phần cứng và sử dụng dữ liệu người dùng.
 - o Xác định quyền sử dụng dữ liệu bản đồ thông qua GMAA KEY:

```
<string name="google_maps_key" translatable="false"
        templateMergeStrategy="preserve">YOUR_KEY_HERE</string>
```


- Chú ý: YOUR_KEY_HERE được thay thế bằng GMAA KEY.
- Một số quyền cần thiết để sử dụng Google Maps:

```
<!-- Xác định quyền truy cập, kiểm soát Internet. -- >
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="com.vidu.minhhoamap.permission.MAPS_RECEIVE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<!-- Xác định quyền truy cập, kiểm soát dữ liệu phần cứng GPS -- >
<uses-permission
    android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<!-- 2 quyền dưới đây không bắt buộc phải khai báo để sử dụng
    Google Maps Android API v2 nhưng được khuyến khích nên khai báo. -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

- Sử dụng OpenGL 2.0 cho việc đồ họa bản đồ:

```
<uses-feature android:glEsVersion="0x00020000" android:required="true"/>
```

2.3. GoogleMap & Xây dựng Đối tượng

- Tạo các đối tượng để thực hiện tương tác giữa ứng dụng với người dùng bao gồm:
 - **GoogleMap:**
 - Kết nối đến Google Map Service
 - Tải dữ liệu bản đồ theo từng mảng nhỏ (tiles).
 - Thử nghiệm dữ liệu bản đồ trên màn hình thiết bị.
 - Thử nghiệm các điều khiển giao tiếp như thu phóng, la bàn...
 - Xử lý các tương tác thu phóng, xoay, góc nhìn...
 - **MapFragment:** xây dựng giao diện bản đồ bằng cách xây dựng Fragment.
 - **MapView:** xây dựng giao diện bản đồ như một điều khiển và tương tác với Activity.
- Truy xuất và sử dụng đối tượng GoogleMap từ thẻ fragment trong XML:

```
GoogleMap mMap =
    ((MapFragment) getFragmentManager().findFragmentById(R.id.map)).getMap();
mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
```

- Hoặc tạo trực tiếp bằng câu lệnh Java:

```
MapFragment mapFragment = MapFragment.newInstance();
GoogleMap mMap = mapFragment.getMap();
mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
getFragmentManager().beginTransaction().add(R.id.container,
                                                    mapFragment).commit();
```

- Cần thực hiện kiểm tra đối tượng GoogleMap trước khi tương tác. Mã lệnh kiểm tra như sau:

```
private void setUpMapIfNeeded(){
//Khởi tạo đối tượng GoogleMap nếu chưa có giá trị
    if( mMap == null){
        mMap = mapFragment.getMap();
        // Kiểm tra đối tượng GoogleMap đã được khởi tạo thành công
        if( mMap != null) {
            // Thực hiện các tùy chỉnh trên đối tượng GoogleMap
            mMap.setMapType (GoogleMap.MAP_TYPE_TERRAIN);
        }
    }
}
```

- Các giao diện bản đồ:
 - MAP_TYPE_NORMAL
 - MAP_TYPE_HYBRID
 - MAP_TYPE_NONE
 - MAP_TYPE_SATELLITE
 - MAP_TYPE_TERRAIN
- Để thực hiện thay đổi giao diện gọi phương thức **setMapType(int)** và truyền vào tham số tương ứng.



Hình 3.4. Minh họa một số loại giao diện bản đồ

- Thiết lập các giá trị ban đầu và điều khiển cho GoogleMap bao gồm:
 - o Góc nhìn (Camera).
 - Thu phóng (zoom).
 - Chuyển điểm (location).
 - Xoay (bearing).
 - Góc nghiêng (titl).
 - o La bàn và điều khiển thu phóng.
 - o Các điều khiển cử chỉ trên bản đồ.
- Thiết lập các giá trị ban đầu và điều khiển cho GoogleMap trong XML:

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/map"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:name="com.google.android.gms.maps.MapFragment"/>
```

- Thực hiện tùy chỉnh GoogleMap thông qua đối tượng **GoogleMapOptions**:

```
GoogleMapOptions mapOptions = new GoogleMapOptions();
```

```
mapOptions.mapType(GoogleMap.MAP_TYPE_HYBRID)
        .scrollGesturesEnabled(true)
        .rotateGesturesEnabled(true)
        .zoomControlsEnabled(true);
```

- Thiết lập đối tượng GoogleMapOptions:
 - o MapFragment.newInstance(GoogleMapOptions).
 - o MapView(Context, GoogleMapOptions).

2.4. Đồ họa trên Google Map

2.4.1. **Marker:**

- Marker là lớp được xây dựng sẵn cho việc sử dụng định vị tọa độ trên bản đồ, hiển thị thông tin địa điểm và tương tác với người dùng. Ví dụ minh họa sử dụng lớp Marker:

```
MarkerOptions markerOptions = new MarkerOptions();
LatLng position = new LatLng(lat, lng);
markerOptions.position(position);
markerOptions.draggable(true);
Marker marker = map.addMarker(markerOptions);
marker.setTitle("Nhà của tui");
```



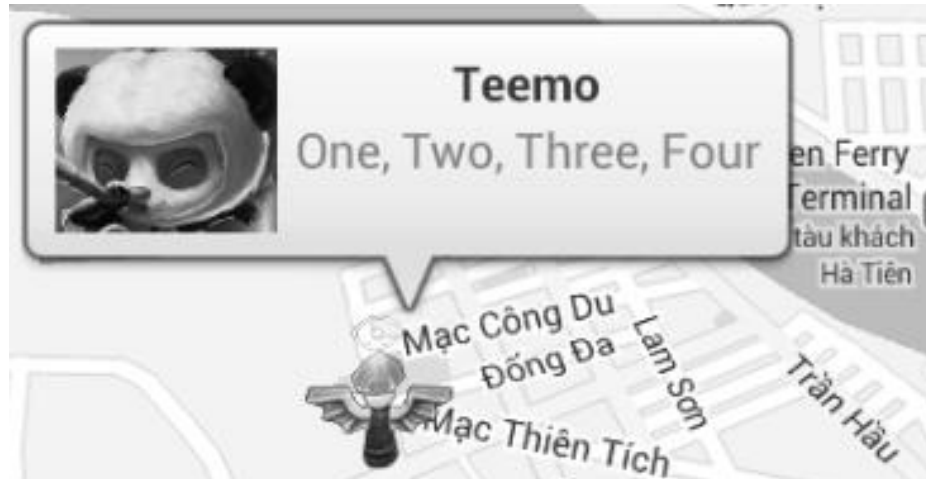
Hình 3.5. Minh họa sử dụng lớp Marker

- Tùy chỉnh Marker bằng các thông số:
 - Position.
 - Title.
 - Snippet.
 - Draggable.
 - Visible.
 - Anchor.
 - Icon.
- Ví dụ tùy chỉnh Marker bằng các thông số position, title, snippet, draggable, anchor, visible và icon:

```
mMap.addMarker(new MarkerOptions()  
    .position(new LatLng(10.385474, 104.488199))  
    .title("Nhà của tôi")  
    .snippet("10 Mạc Thiên Tích");  
    .draggable(true)  
    .anchor(0.7f, 0.6f)  
    .visible(BitmapDescriptorFactory  
        .fromResource(R.drawable.vision_ward)));
```

2.4.2. InfoWindow:

- **InfoWindow** được thể hiện phía trên đối tượng Marker thể hiện các thông tin cần thiết về vị trí đang định vị.
- Chỉ một đối tượng InfoWindow hiển thị ở một thời điểm và có thể điều khiển thông qua hai phương thức:
 - showInfoWindow().
 - hideInfoWindow().
- Để tùy chỉnh InfoWindow trong lớp GoogleMap hỗ trợ giao diện **InfoWindowAdapter** bao gồm 2 phương thức:
 - getInfoWindow(Marker).
 - getInfoContents(Marker).
- Gọi phương thức **setInfoWindowAdapter** để thiết lập InfoWindow cho đối tượng GoogleMap.



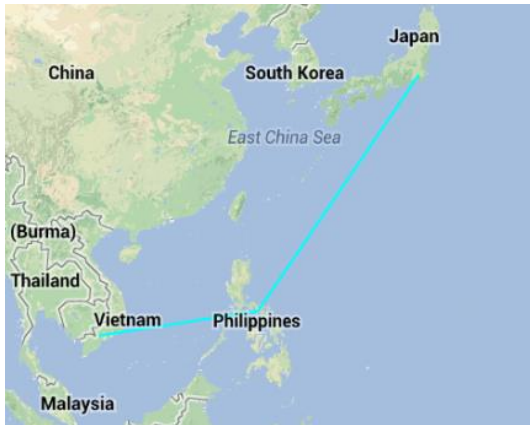
Hình 3.6. Minh họa sử dụng Marker và InfoWindow

- Google Maps API cung cấp các giao diện cho phép bắt lại các sự kiện tương với Marker và InfoWindow.
 - OnMarkerClickListener.
 - OnMarkerDragListener.
 - OnInfoWindowClickListener.

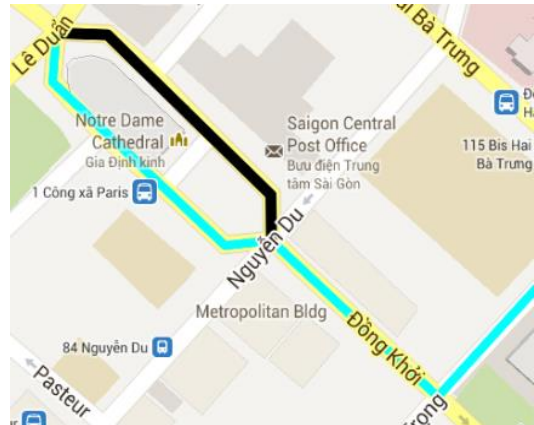
2.4.3. Shape:

- Shape bao gồm các đối tượng hình học được vẽ trên bản đồ bao gồm các dạng sau:
 - Polyline
 - Polygons
 - Circle
- **Polyline:** được tạo bằng các nối các điểm dựa trên toạ độ điểm. Ví dụ minh họa:

```
PolylineOptions options = new PolylineOptions()
    .add(new LatLng(10.919618, 106.523438))
    .add(new LatLng(13.410994, 123.046875))
    .add(new LatLng(35.746512, 139.833984));
Polyline polyline = mMap.addPolyline(options);
```



Hình 3.7. Minh họa sử dụng Polyline



Hình 3.8. Minh họa sử dụng Polyline

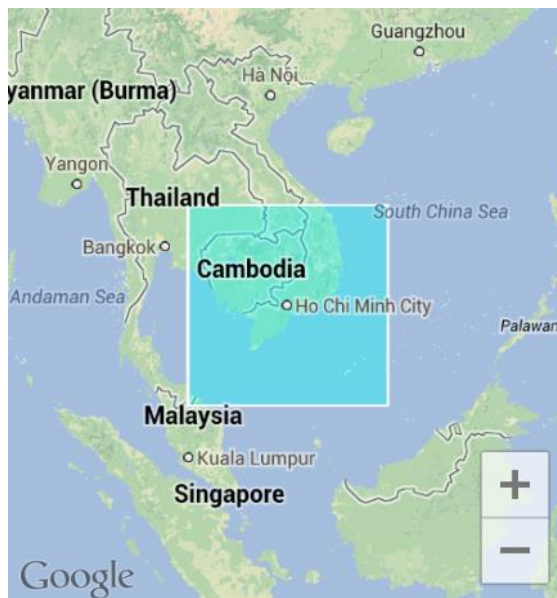
- Polyline có các phương thức tùy chỉnh sau:

boolean	equals (Object <i>other</i>)
int	getColor ()
String	getId ()
List< LatLng >	getPoints ()
float	getWidth ()
float	getZIndex ()
int	hashCode ()
boolean	isGeodesic ()
boolean	isVisible ()
void	remove ()
void	setColor (int <i>color</i>)
void	setGeodesic (boolean <i>geodesic</i>)
void	setPoints (List< LatLng > <i>points</i>)
void	setVisible (boolean <i>visible</i>)
void	setWidth (float <i>width</i>)
void	setZIndex (float <i>zIndex</i>)

- **Polygon:** giống với Polyline được tạo bằng các nối các điểm dựa trên toạ độ điểm, tuy nhiên Polygon cho phép khép kín các toạ độ và thao tác vùng đã khép kín.

- Ví dụ minh họa sử dụng Polygon:

```
double height = 5, width = 5;
LatLng center = new LatLng(10.780231, 106.698579);
PolygonOptions options = new PolygonOptions();
options.add(new LatLng(center.latitude - height, center.longitude - width))
        .add(new LatLng(center.latitude - height, center.longitude + width))
        .add(new LatLng(center.latitude + height, center.longitude + width))
        .add(new LatLng(center.latitude - height, center.longitude + width));
Polygon polygon = mMap.addPolygon(options);
```



Hình 3.9. Minh họa sử dụng Polygon

- Polygon có các phương thức tùy chỉnh sau:

PolygonOptions	add(LatLng point)
PolygonOptions	add(LatLng... points)
PolygonOptions	addAll(Iterable<LatLng> points)
PolygonOptions	addHole(Iterable<LatLng> points)
int	describeContents()
PolygonOptions	fillColor(int color)
PolygonOptions	geodesic(boolean geodesic)
int	getFillColor()
List<List<LatLng>>	getHoles()

List<LatLng>	getPoints()
int	getStrokeColor()
float	getStrokeWidth()
float	getZIndex()
boolean	isGeodesic()
boolean	isVisible()
PolygonOptions	strokeColor(int color)
PolygonOptions	strokeWidth(float width)
PolygonOptions	visible(boolean visible)
void	writeToParcel(Parcel out, int flags)
PolygonOptions	zIndex(float zIndex)

- **Circle:** đối tượng hình học (hình tròn) được xác định trên bản đồ thông qua bán kính và tâm.

```
CircleOptions circleOptions = new CircleOptions();
circleOptions.center(HCM).radius(5);
Circle circle = mMap.addCircle(circleOptions);
circle.setFillColors(Color.CYAN);
circle.setStrokeColor(Color.WHITE);
```



Hình 3.10. Minh họa sử dụng Circle

- Circle có các phương thức tùy chỉnh sau:

<u>LatLng</u>	<u>getCenter()</u>
int	<u>getFillColor()</u>
String	<u>getId()</u>
double	<u>getRadius()</u>
int	<u>getStrokeColor()</u>
float	<u>getStrokeWidth()</u>
float	<u>getZIndex()</u>
boolean	<u>isVisible()</u>
void	<u>remove()</u>
void	<u>setCenter</u> (<u>LatLng</u> <i>center</i>)
void	<u>setFillColor</u> (int <i>color</i>)
void	<u>setRadius</u> (double <i>radius</i>)
void	<u>setStrokeColor</u> (int <i>color</i>)
void	<u>setStrokeWidth</u> (float <i>width</i>)
void	<u>setVisible</u> (boolean <i>visible</i>)
void	<u>setZIndex</u> (float <i>zIndex</i>)

Bài 4

GOOGLE CLOUD MESSAGING

1. GIỚI THIỆU

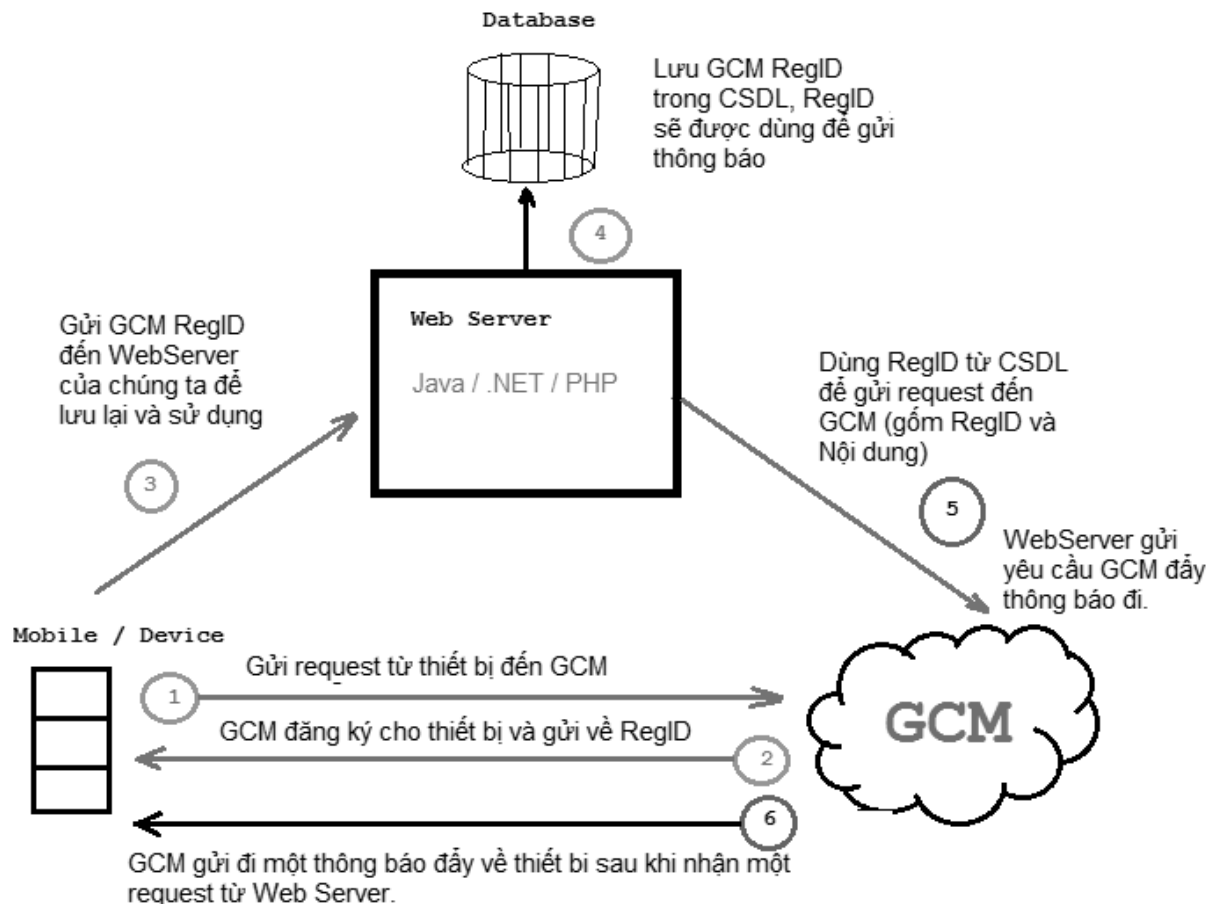
- Google Cloud Messaging (GCM) là một dịch vụ cho phép gửi dữ liệu từ máy chủ của bạn đến các thiết bị Android của người dùng, và ngược lại.
- GCM được sử dụng cho hoạt động giao dịch giữa các ứng dụng và máy chủ hỗ trợ đầu cuối. Cloud Messaging hiện đang được tích hợp vào Google Play Services và nhờ những cải tiến mạnh mẽ, Cloud Messaging giờ đây tương trợ kết nối liên tục và truyền tải thông điệp ngược dòng (upstream). Do đó, thiết bị có thể gửi phản hồi trở lại máy chủ mà không còn chỉ một chiều từ máy chủ đến thiết bị như trước đây.
- Nhờ tính năng truyền tải ngược, các thông báo (Notification) mà bạn nhận được và bỏ lỡ trên thiết bị này có thể được đồng bộ với thiết bị khác. Bạn sẽ không phải soát lại hàng loạt các thông báo tương tự trên điện thoại khi chuyển sang sử dụng chiếc máy tính bảng của mình.



Hình 4.1. Logo Google Cloud Messaging.

- GCM là hoàn toàn miễn phí và không giới hạn băng thông. Dịch vụ hoạt động trên các gói dữ liệu có dung lượng nhỏ hơn 4kb và tin nhắn tới thiết bị Android là tức thời (Push-notification).
- GCM là phù hợp cho việc xây dựng các ứng dụng nhắn tin tức thời hoặc tương tác giữa người dùng và nhà phát triển ứng dụng.
- Ví dụ:
 - o Thông báo tới thiết bị Android của người khi có một **email** mới. GCM gửi thông báo tới người quản lý khi có một đơn đặt hàng trên trang Web của họ.

- Gửi tin nhắn tới cả một cộng đồng như: cộng đồng nhân viên thuộc cùng một công ty, học sinh hoặc phụ huynh của một trường học với chi phí thấp nhất và hiệu quả nhất.
- Nguyên tắc hoạt động:
Hãy cùng xem hình dưới để hiểu rõ hơn.



Hình 4.2. Nguyên tắc hoạt động khi Web Server gửi thông báo về thiết bị thông qua Google Cloud Messaging.

2. CẤU HÌNH CHO GOOGLE CLOUD MESSAGING

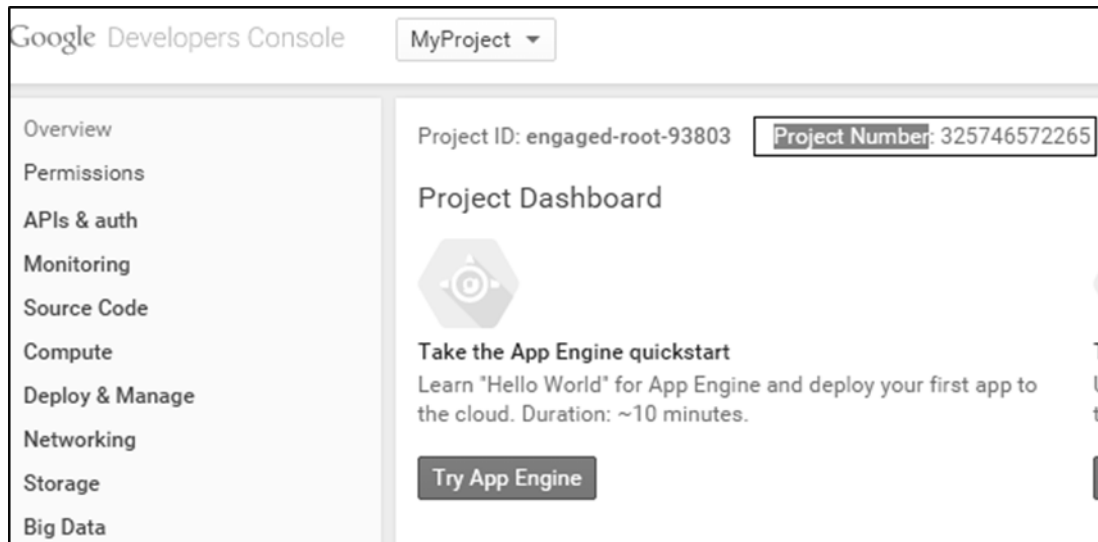
2.1. Đăng ký dịch vụ GCM

- Đầu tiên, bạn cần đăng ký Google Cloud Messaging bằng tài khoản Google của bạn.
 - o Truy cập Google Console tại: <https://cloud.google.com/console/project>
 - o Nếu bạn chưa tạo một dự án API nào, nhấp vào "Create Project". Điền tên và nhấn Create.



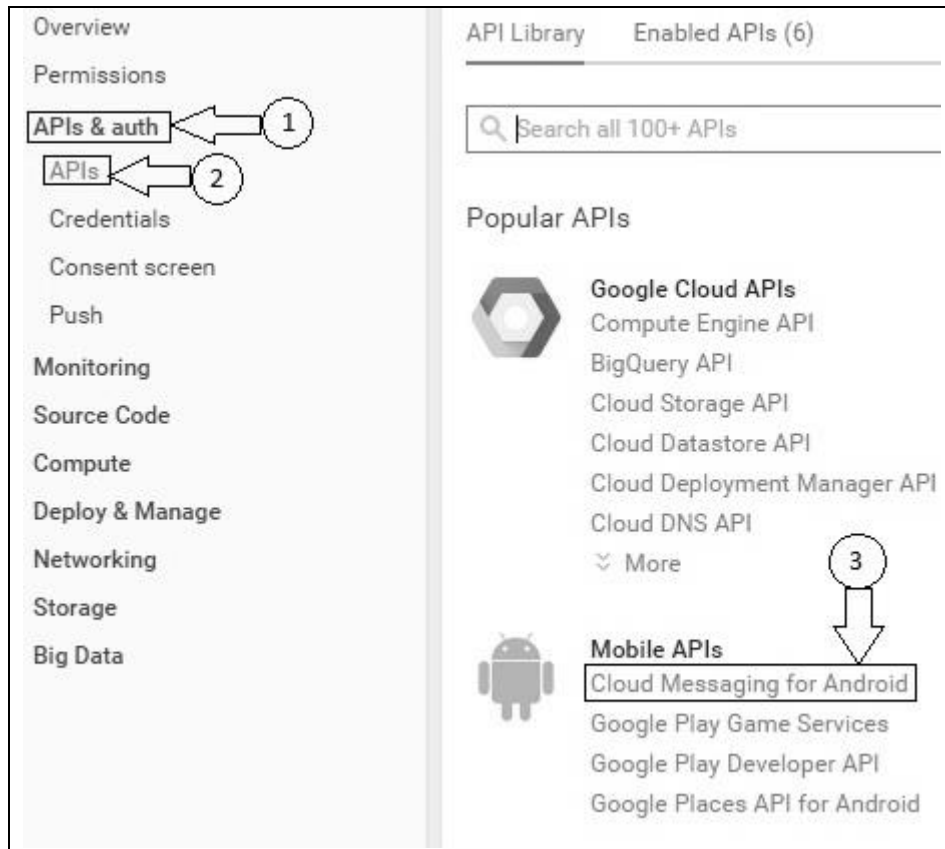
Hình 4.3. Tạo một Project trên Google Developers Console.

- Một khi dự án đã được tạo ra, một trang xuất hiện hiển thị ID dự án của bạn. Ví dụ: id dự án của bạn là *engaged-root-93803*

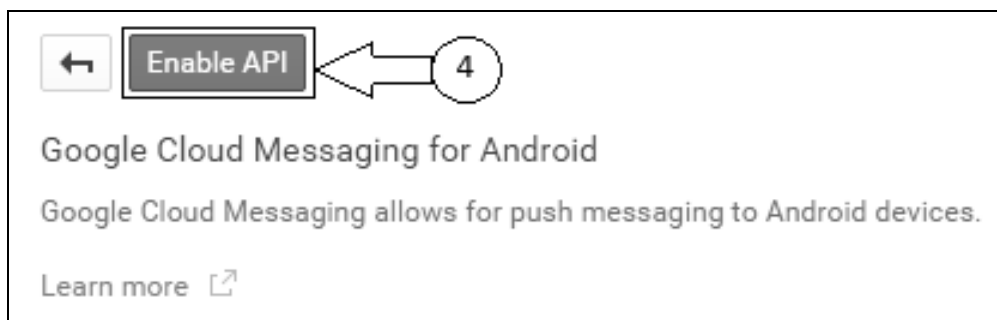


Hình 4.4. Minh họa màn hình tổng quan của dự án trên Google Developers Console.

- Sao chép số ID Project này lưu lại. Bạn sẽ sử dụng nó như GCM sender ID.
- Sau đó, bạn cần kích hoạt API cho GCM
 - Trong thanh bên trên bên trái, chọn mục API & auth → Chọn tiếp APIs
 - Tìm đến mục **Google Cloud Messaging for Android** và “Enable API”.



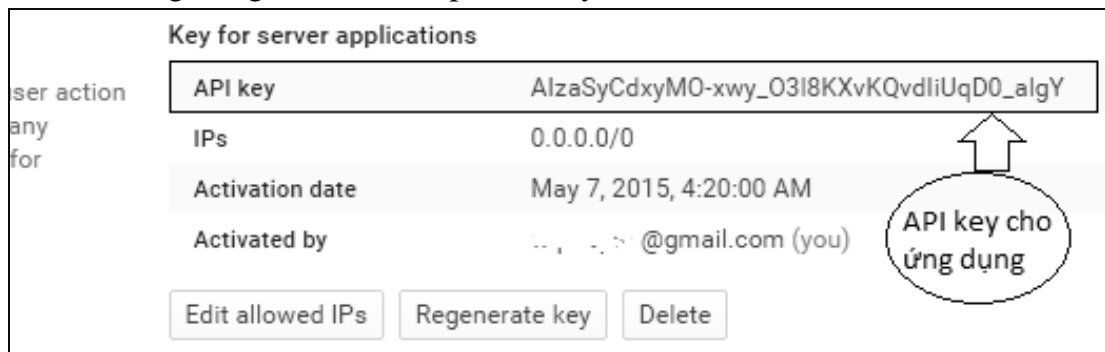
Hình 4.5 Kích hoạt Google Cloud Messaging trong dự án của bạn 1



Hình 4.6. Kích hoạt Google Cloud Messaging trong dự án của bạn 2

- Bây giờ bạn đã có thể lấy API Key với các bước sau:
 - o Trong thanh bên trên bên trái, chọn API & auth → Sau đó chọn Credentials.
 - o Trong phần Public API access, nhấp vào Create new key.
 - o Trong hộp thoại Create a new key, nhấp vào Server key.
 - o Trong hộp thoại IP address bên dưới nhập vào địa chỉ IP Web Server của bạn. Dùng địa chỉ 0.0.0.0/0 cho mục đích thử nghiệm.
 - o Nhấp vào Create.

- Trong trang mới, sao chép API Key:



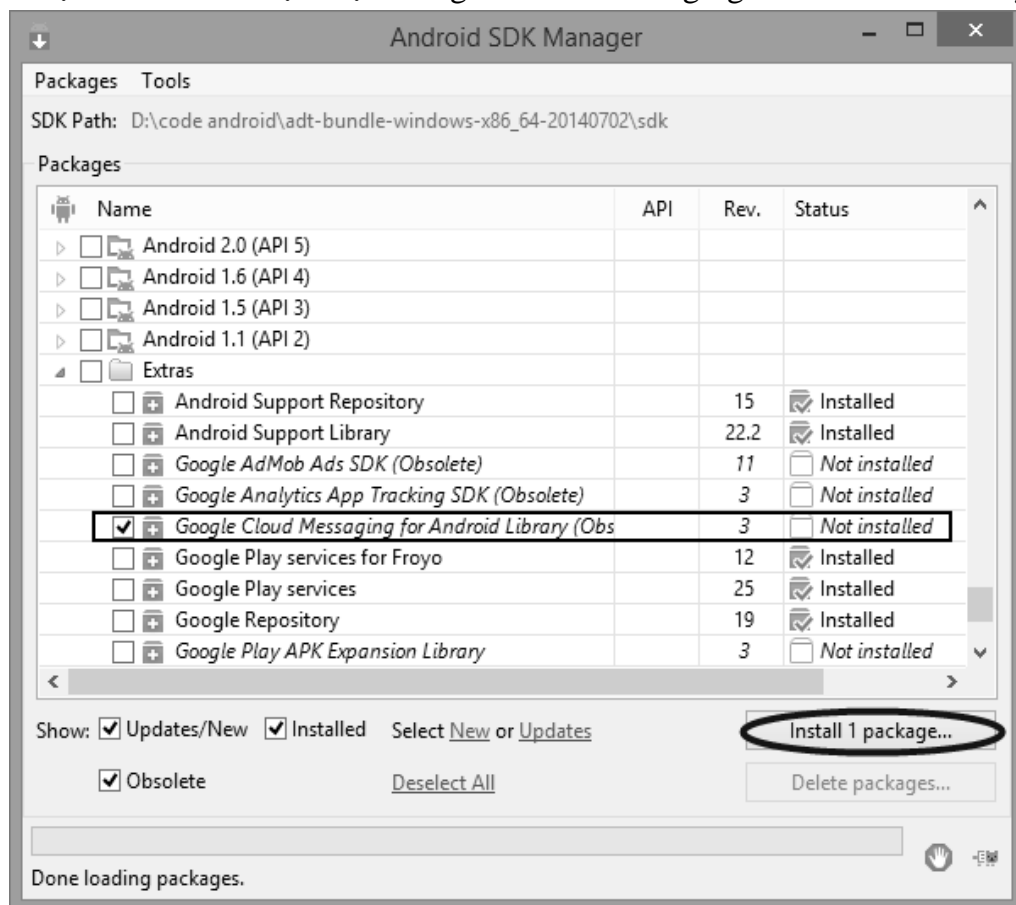
Hình 4.7. Sao chép lại API key.

- Mở tập tin appengine-web.xml và tìm đến dòng lệnh bên dưới và thay thế “YOUR_KEY_HERE” là API key vừa lấy được:

```
<property name="gcm.api.key" value="YOUR_KEY_HERE" />
```

2.2. Cấu hình môi trường hoạt động cho GCM

- Download thư viện hỗ trợ GCM
- Mở SDK Manager lên và tiến hành tải thư viện hỗ trợ GCM.
- Vào mục Extras → Chọn mục Google Cloud Messaging for Android Library:



Hình 4.8. Tải GCM helper library.

2.3. Chuẩn bị máy ảo với Google API:

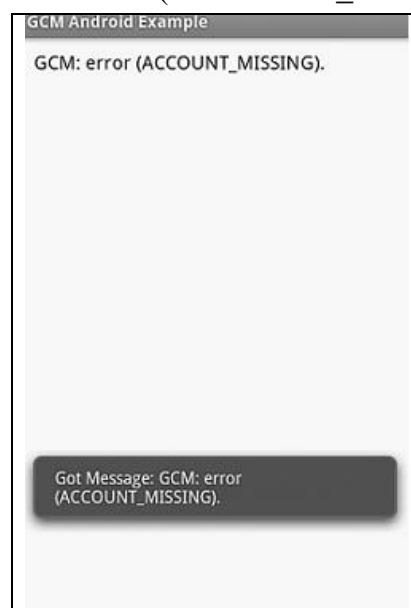
- Mở ADV Manager và tạo một máy ảo Google API emulator.

Release Name	API Level ▾	ABI	Target
KitKat Wear	L	armeabi-v7a	Android L (Preview)
KitKat Wear	L	x86	Android L (Preview)
Lollipop	21	x86	Android 5.0
Lollipop	21	x86_64	Google APIs (Google Inc.) google_8
<i>Lollipop Download</i>	21	armeabi-v7a	Android SDK Platform 5.0
<i>Lollipop Download</i>	21	x86_64	Android SDK Platform 5.0
<i>Lollipop Download</i>	21	armeabi-v7a	System Image armeabi-v7a with Go
<i>Lollipop Download</i>	21	x86	System Image x86 with Google APIs
KitKat	19	x86	Google APIs (x86 System Image) Gc
<i>KitKat Download</i>	19	armeabi-v7a	Android SDK Platform 4.4.2
<i>Jelly Bean Download</i>	18	armeabi-v7a	Android SDK Platform 4.3
<i>Jelly Bean Download</i>	18	x86	Android SDK Platform 4.3
<i>Jelly Bean Download</i>	17	armeabi-v7a	Android SDK Platform 4.2.2
<i>Jelly Bean Download</i>	17	x86	Android SDK Platform 4.2

☒ Show downloadable system images

Hình 4.9. Tạo máy ảo Google API emulator

- Chạy máy ảo lên.
- Sau khi máy ảo chạy lên. Vào mục Setting, chọn Select Account & Sync. Và chọn Add Account sau đó điền tài khoản Google của bạn vào.
- *Lưu ý:* Nếu bạn không đăng nhập tài khoản google vào và chạy ứng dụng sử dụng dịch vụ GCM sẽ bị lỗi GCM: error (ACCOUNT_MISSING).



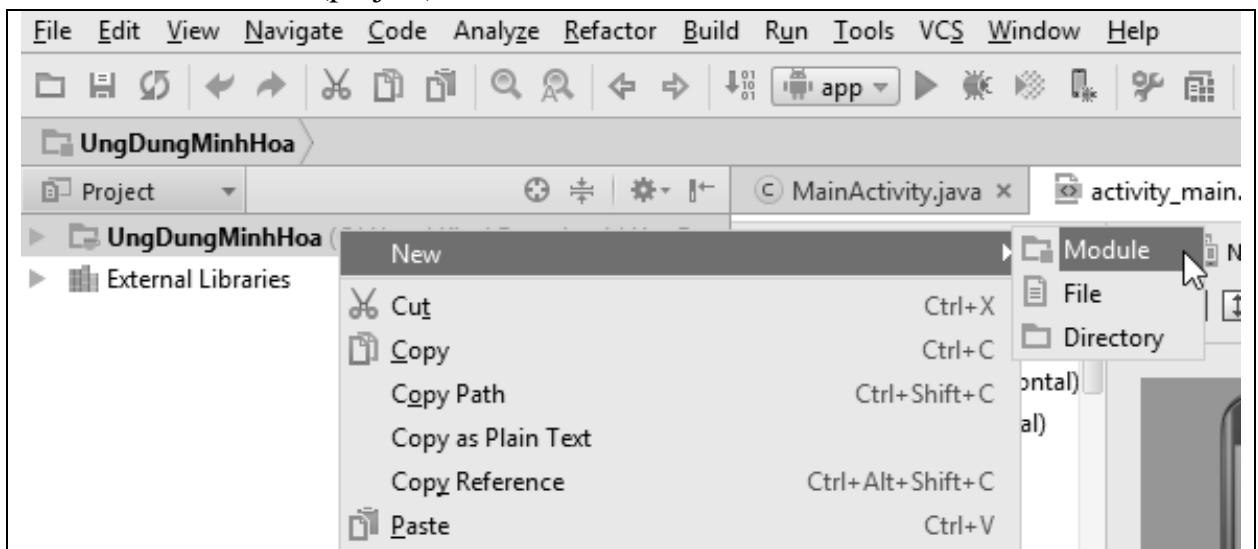
Hình 4.10. Lỗi Account Missing.

2.4. Tạo Project để đăng ký GCM

- Tạo một Android Application Project.
- Trong tập tin AndroidManifest.xml cần đăng ký các quyền sau:
 - o **INTERNET** – để kết nối mạng.
 - o **ACCESS_NETWORK_STATE** – để kiểm tra trạng thái mạng.
 - o **GET_ACCOUNTS** – cần bởi vì GCM truy cập Google account của bạn.
 - o **WAKE_LOCK** – Cần để gọi thiết bị của bạn khi ở trạng thái sleep.
 - o **VIBRATE** – Xin quyền rung thiết bị.

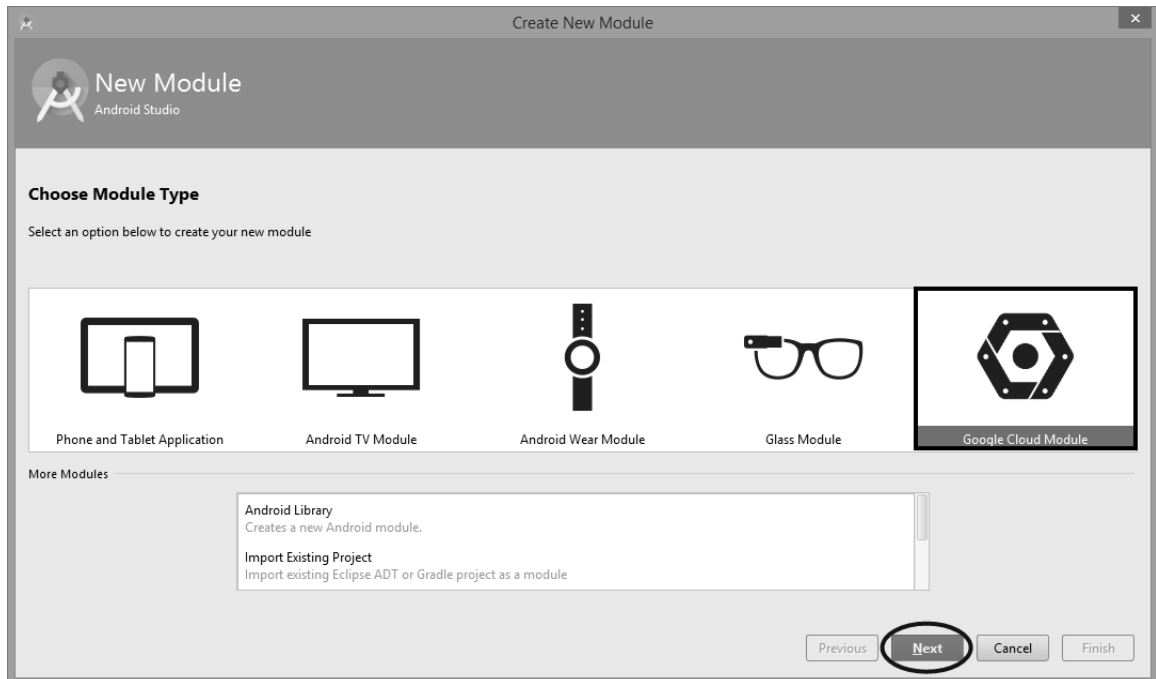
2.5. Sử dụng “backend” trong Android Studio

- Backend được hiểu như một server. Backend minh họa dưới đây sử dụng Google Cloud Endpoints để định nghĩa một RESTfull API.
- Sau đây là minh họa cụ thể từng bước thực hiện tạo một “backend” trong Android Studio:
 - o Để tạo một “backend” từ một dự án (project) Android đã có sẵn, chúng ta khởi động Android Studio và chọn File → New → Module hoặc nhấp chuột phải vào dự án (project) và chọn New → Module.



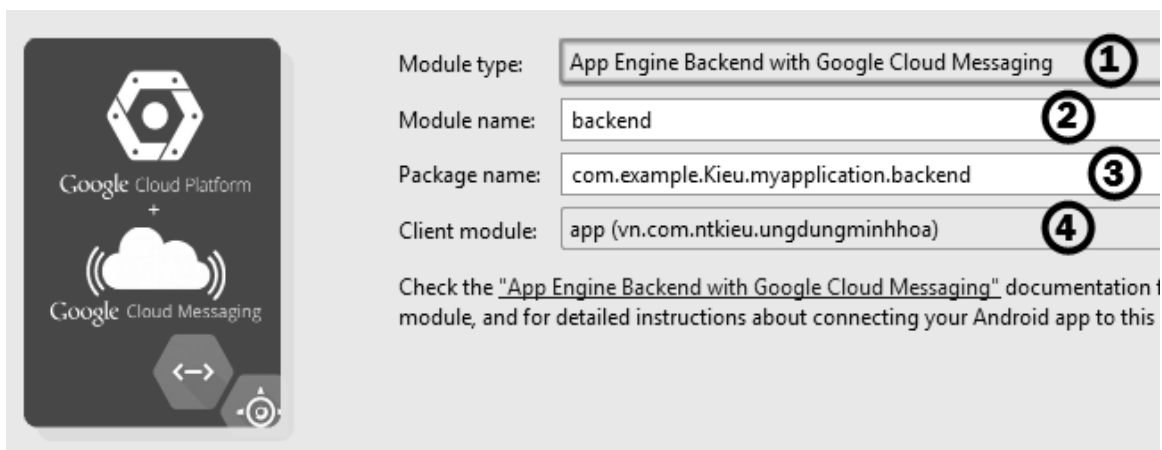
Hình 4.11. Tạo “backend”

- o Xuất hiện hộp thoại “Create New Module” → Chọn “Google Cloud Module” → Next



Hình 4.12. Tạo backend

- Xuất hiện hộp thoại “New Google Cloud Module” và điền các thông tin như sau:

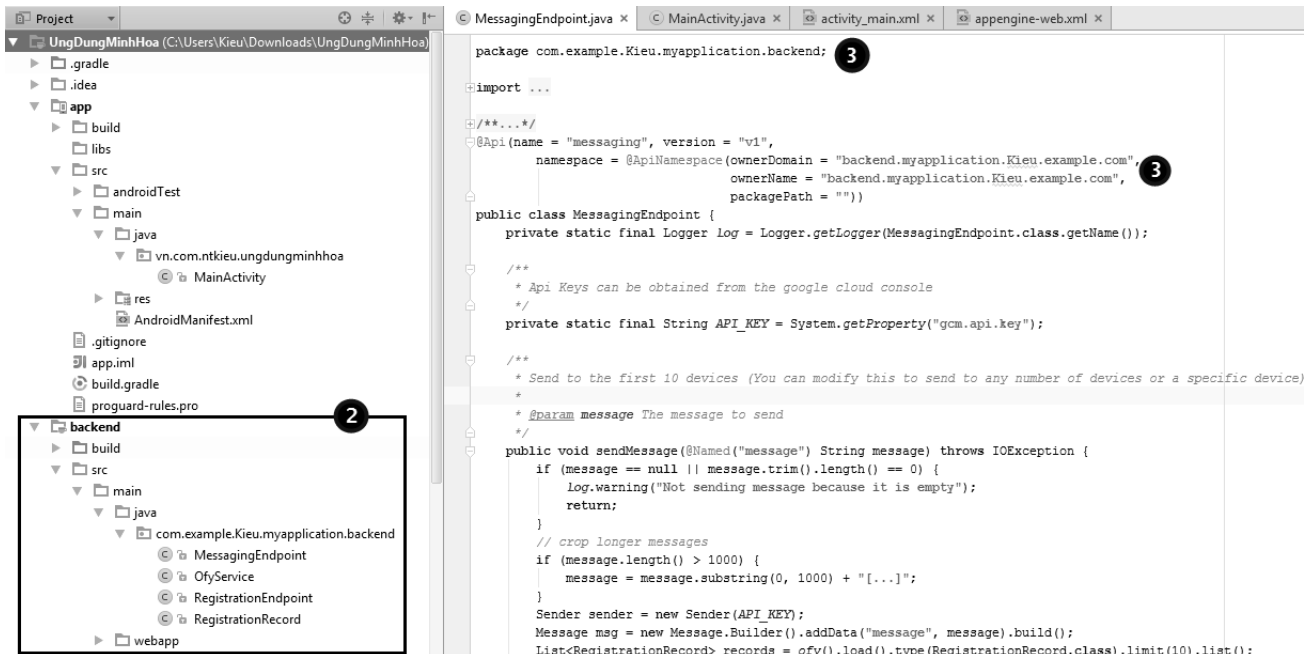


Hình 4.13. Tạo backend

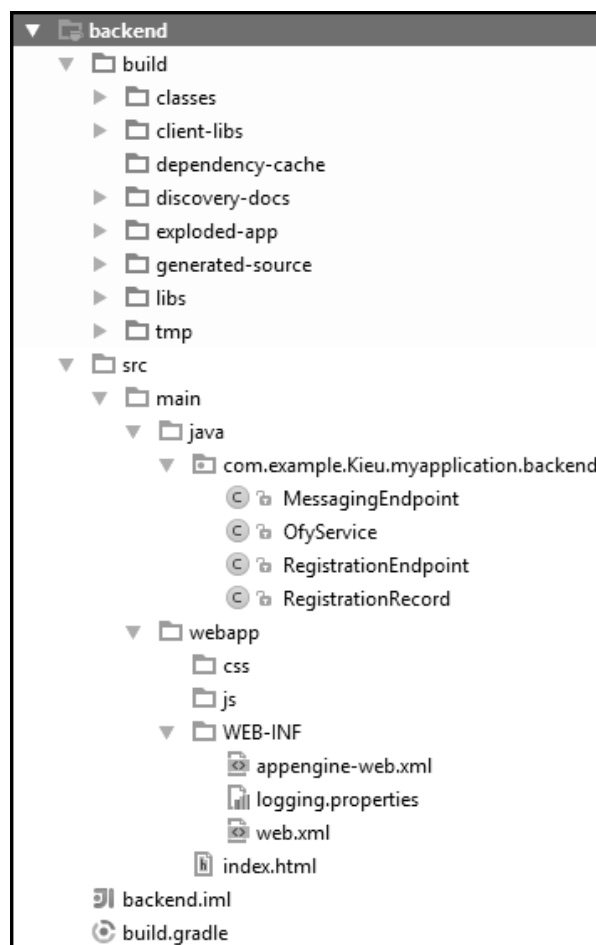
Trong đó:

- Module type (1): chúng ta chọn “App Engine Backend with Google Cloud Messaging”.
- Module name (2): là tên của backend mà chúng ta muốn tạo, ở đây tôi để mặc định là backend.
- Package name (3): tên gói để chứa backend.
- Client module (4): backend được tạo cho ứng dụng này.

Cuối cùng, chúng ta chọn Finish để kết thúc quá trình tạo backend. Và backend sau khi tạo có cấu trúc như sau:

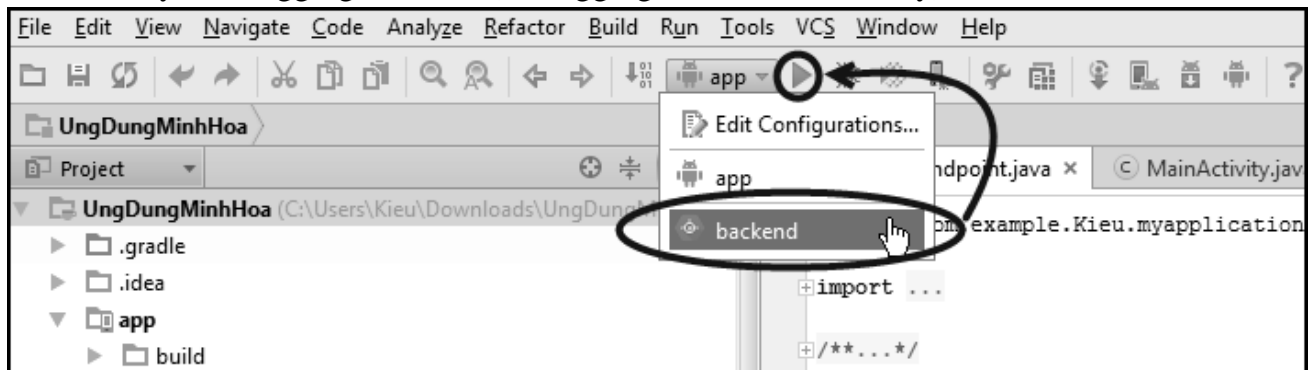


Hình 4.14. Cấu trúc backend sau khi tạo

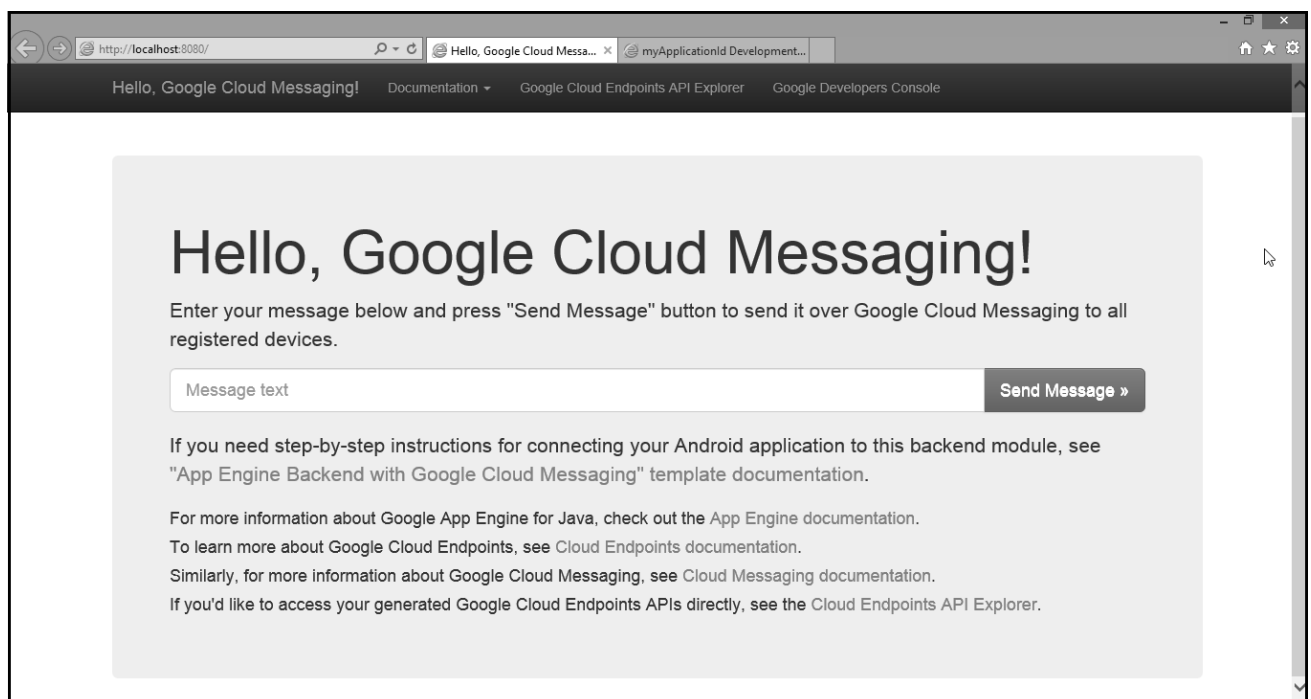


Hình 4.15. Cấu trúc backend sau khi tạo

- Chạy Debbugging backend (Debugging the backend locally).



Hình 4.16. Chạy Debbugging backend



Hình 4.17. Kết quả sau khi chạy Debbug

- Kết nối ứng dụng Android của bạn tới backend

Trước khi những tin nhắn có thể được gửi từ Google Cloud Messaging backend đến các thiết bị thì những thiết bị này cần được đăng ký với GCM backend. Ví dụ sau đây sẽ minh họa cho bạn cách tạo AsyncTask để đăng ký các thiết bị với một GCM backend:

```
class GcmRegistrationAsyncTask extends AsyncTask<Void, Void, String> {  
    private static Registration regService = null;  
    private GoogleCloudMessaging gcm;  
    private Context context;  
  
    // Gán vào SENDER_ID số dự án (Project number) đã có được khi ta đăng ký  
    // dịch vụ GCM  
    private static final String SENDER_ID = "325746572265";
```

```

public GcmRegistrationAsyncTask(Context context) {
    this.context = context;
}
@Override
protected String doInBackground(Void... params) {
    if (regService == null) {
        Registration.Builder builder = new
            Registration.Builder(AndroidHttp.newCompatibleTransport(),
                                new AndroidJsonFactory(), null)
        // setRootUrl và setGoogleClientRequestInitializer để kiểm
        // tra cục bộ (local)
        .setRootUrl("http://10.0.2.2:8080/_ah/api/")
        .setGoogleClientRequestInitializer(new
            GoogleClientRequestInitializer() {
                @Override
                public void initialize(AbstractGoogleClientRequest<?>
                    abstractGoogleClientRequest) throws IOException {
                    abstractGoogleClientRequest.setDisableGZipContent(true);
                }
            });

        regService = builder.build();
    }
    String msg = "";
    try {
        if (gcm == null) {
            gcm = GoogleCloudMessaging.getInstance(context);
        }
        String regId = gcm.register(SENDER_ID);
        msg = "Device registered, registration ID=" + regId;

        //Gửi ID đăng ký đến server của bạn qua HTTP. Như vậy, nó có
        //thể sử dụng GCM/ HTTP hoặc CSS để gửi tin nhắn đến các
        //ứng dụng của bạn
        regService.register(regId).execute();

    } catch (IOException ex) {
        ex.printStackTrace();
        msg = "Error: " + ex.getMessage();
    }
    return msg;
}
@Override
protected void onPostExecute(String msg) {
    Toast.makeText(context, msg, Toast.LENGTH_LONG).show();
    Logger.getLogger("REGISTRATION").log(Level.INFO, msg);
}
}

```

○ Chú ý:

- Trong lớp MainActivity.java phải có phương thức ***new GcmRegistrationAsyncTask(this).execute();*** thì lớp GcmRegistrationAsyncTask.java mới được thực thi.
- Chạy kiểm tra đăng ký thiết bị với máy ảo Google API emulator.
- Hiện thị các thông báo (notifications) được đẩy (push) từ GCM backend:
 - Để sắp xếp thứ tự hiện thị các thông báo (notifications) được gửi đi từ backend, chúng ta cần viết thêm 2 lớp GcmIntentService.java và GcmBroadcastReceiver.java vào ứng dụng Android (phía Client):
 - Mã nguồn cho lớp GcmIntentService.java

```
public class GcmIntentService extends IntentService {

    public GcmIntentService() {
        super("GcmIntentService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        Bundle extras = intent.getExtras();
        GoogleCloudMessaging gcm = GoogleCloudMessaging.getInstance(this);

        String messageType = gcm.getMessageType(intent);

        if (extras != null && !extras.isEmpty()) {
            if (GoogleCloudMessaging.MESSAGE_TYPE_MESSAGE.equals(messageType)) {
                Logger.getLogger("GCM_RECEIVED").log(Level.INFO,
                                                            extras.toString());
                showToast(extras.getString("message"));
            }
        }

        GcmBroadcastReceiver.completeWakefulIntent(intent);
    }

    protected void showToast(final String message) {
        new Handler(Looper.getMainLooper()).post(new Runnable() {

            @Override
            public void run() {

                Toast.makeText(getApplicationContext(), message,
                                                            Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

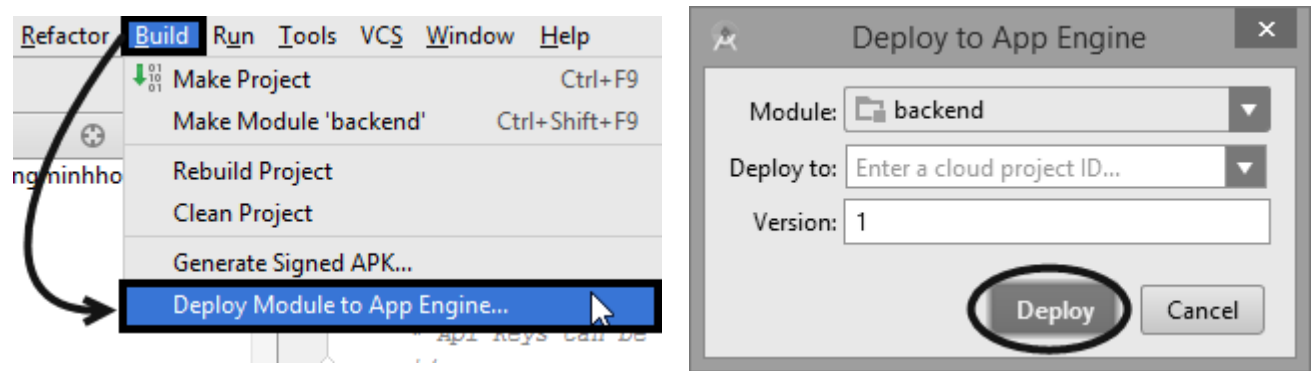
- Mã nguồn cho lớp GcmBroadcastReceiver.java

```
public class GcmBroadcastReceiver extends WakefulBroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        ComponentName comp = new ComponentName(context.getPackageName(),
            GcmIntentService.class.getName());
        startWakefulService(context, (intent.setComponent(comp)));
        setResultCode(Activity.RESULT_OK);
    }
}
```

Sau khi thêm 2 lớp GcmIntentService.java và lớp GcmBroadcastReceiver.java vào ứng dụng Android, chúng ta đăng ký cho chúng bằng cách thêm đoạn mã vào tập tin AndroidManifest.xml (đặt trong thẻ <application>) như sau:

```
<receiver
    android:name=".GcmBroadcastReceiver"
    android:permission="com.google.android.c2dm.permission.SEND" >
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
        <category android:name="TEN_PACKAGE_CHUA_UNG_DUNG_CUA_BAN" />
    </intent-filter>
</receiver>
<service android:name=".GcmIntentService" />
```

- Triển khai backend trực tiếp đến Google App Engine
 - o Nếu backend đang chạy thì dừng backend bằng cách chọn Run → Stop.
 - o Chọn Build → Deploy Module to App Engine. Xuất hiện hộp thoại Deploy to App Engine → chọn dự án Google Developers Console (phải đăng nhập tài khoản Google) → chọn Deploy. Nếu chưa có dự án Google Developers Console nào thì nhấn chọn “create a new Google Developers Console project” để tạo dự án Google Developers Console mới.



Hình 4.18. Triển khai backend đến App Engine

- Mở tập tin appengine-web.xml theo đường dẫn src/main/webapp/WEB-INF/appengine-web.xml và tìm đến thẻ <application> và thay thế “myApplicationId” bằng ID của dự án bạn vừa tạo (dự án Google Developers Console).

Bài 5

CÁC ĐIỀU KHIỂN ĐA TRUYỀN THÔNG

1. MEDIA PLAYER

1.1. Giới thiệu

- Android cung cấp lớp MediaPlayer để quản lý các tác vụ đa truyền thông bao gồm các tập tin âm thanh, hình ảnh, video...
- Có thể truy xuất các tập tin media thông qua việc lưu trữ như tài nguyên ứng dụng, bộ nhớ thiết bị, content provider hoặc thông qua URL.
- MediaPlayer quản lý các tập tin media và luồng xử lý thông qua một tập các trạng thái sau:
 - o Khởi tạo đối tượng MediaPlayer.
 - o Chuẩn bị bộ thu phát MediaPlayer.
 - o Bắt đầu thực hiện thu phát.
 - o Thực hiện các thao tác Pause và Stop trên tập tin media trong khi đang thu phát.
 - o Hoàn thành quá trình thu phát.

1.2. Xây dựng MediaPlayer Audio

- Android hỗ trợ truy xuất các tập tin Audio thông qua các lưu trữ như: tài nguyên ứng dụng, bộ nhớ thiết bị, content provider và xử lý các luồng URL.
- Có thể đóng gói tập tin Audio vào thư mục res/raw như một dạng tài nguyên của ứng dụng.
- Để thu phát một tập tin Audio cần tạo một đối tượng MediaPlayer và thiết lập nguồn dữ liệu cho đối tượng này.
- Thực hiện thu phát bằng phương thức Create() truyền vào 2 tham số: context của ứng dụng và một trong những dạng tài nguyên sau:
 - o Định danh của tài nguyên.
 - o URI trỏ đến nơi lưu trữ của tập tin trên thiết bị.
 - o URI trỏ đến tập tin trực tuyến thông qua URL.
 - o URI trỏ đến dòng dữ liệu trong bảng của content provider.
- Ví dụ tạo đối tượng MediaPlayer

```
Context appContext = getApplicationContext();  
MediaPlayer filePlayer = MediaPlayer.create(appContext,
```

```
Uri.parse(file:///sdcard/localfile.mp3));
MediaPlayer urlPlayer = MediaPlayer.create(appContext,
Uri.parse("http://site.com/audio/audio.mp3"));
MediaPalyer contentPlayer = MediaPlayer.create(appContext,
Settings.System.DEFAULT_RINGTONE_URI);
```

- Để thực thi việc thu phát tập tin Video cần tạo một màn hình cho việc trình chiếu. Có hai cách để thực hiện công việc này:
 - o Sử dụng thành phần Video View, đóng gói các thao tác trên đối tượng MediaPlayer (tạo trình chiếu, chuẩn bị và cấp phát tập tin) vào trong thành phần View này.
 - o Xây dựng màn hình hiển thị riêng và gắn kết dữ liệu thông qua đối tượng MediaPlayer.

1.3. Xây dựng MediaPlayer Video

- Để thực thi việc thu phát tập tin Video cần tạo một màn hình cho việc trình chiếu. Có hai cách để thực hiện công việc này:
 - o Sử dụng thành phần Video View, đóng gói các thao tác trên đối tượng MediaPlayer (tạo trình chiếu, chuẩn bị và cấp phát tập tin) vào trong thành phần View này.
 - o Xây dựng màn hình hiển thị riêng và gắn kết dữ liệu thông qua đối tượng MediaPlayer.
- Ví dụ tạo bộ thu phát bằng Video View:

```
VideoView videoView = (VideoView) findViewById(R.id.surface);
videoView.setKeepScreenOn(true);
videoView.setVideoPath("/sdcard/test2.2gp");
if(videoView.canSeekForward())
    videoView.seekTo(videoView.getDuration()/2);
videoView.start();
[ ... do something ... ]
videoView.stopPlayback();
```

- Ví dụ tạo màn hình hiển thị trong Video View:

```
<LinearLayout xmlns:android= "http://schemas.android.com/apk/res/android"
android:orientation= "vertical"
android:layout_width= "fill_parent"
android:layout_height= "fill_parent">
    <SurfaceView
        android:id= "@+id/surface"
        android:layout_width= "wrap_content"
```

```
        android:layout_height = "wrap_content"
        android:layout_gravity= "center">
    </SurfaceView>
</LinearLayout>
```

- Tạo đối tượng Surface Holder để hỗ trợ việc cập nhật các nguồn xử lý bên dưới:

```
public class MyActivity extends Activity implements SurfaceHolder.Callback
{
    private MediaPlayer mediaPlayer;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mediaPlayer = new MediaPlayer();
        SurfaceView surface = (SurfaceView) findViewById(R.id.surface);
        SurfaceHolder holder = surface.getHolder();
        holder.addCallback(this);
        holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
        holder.setFixedSize(400, 300);
    }
}
```

- Khởi tạo nội dung của tập tin cần trình chiếu:

```
public void SurfaceCreated(SurfaceHolder holder)
{
    try{
        mediaPlayer.setDisplay(holder);
        mediaPlayer.setDataSource("/sdcard/test2.3gp");
        mediaPlayer.prepare();
        mediaPlayer.start();
    }
    catch(IllegalArgumentException e)
    {
        Log.d("MEDIA_PALYER", e.getMessage());
    }
    catch(IllegalStateException e){
        Log.d("MEDIA_PALYER", e.getMessage());
    }
    catch(IOException e)
    {
        Log.d("MEDIA_PALYER", e.getMessage());
    }
}
```

1.4. Các điều khiển trên MediaPlayer

- Khởi chạy đối tượng bằng phương thức **start()**.
- Ví dụ:

```
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.start();
```

- MediaPlayer cung cấp một số phương thức cho việc điều khiển như *getDuration*, *getCurrentPosition*, *seekTo*...

```
mediaPlayer.start();
int pos = mediaPlayer.getCurrentPosition();
int duration = mediaPlayer.getDuration();
mediaPlayer.seekTo(pos + (duration-pos)/10);
[ ... wait for a duration ... ]
mediaPlayer.stop();
```

- Một số phương thức khác như hỗ trợ âm thanh, chống khóa màn hình trong khi thu phát, thiết lập các chế độ phát lại...
 - o Sử dụng phương thức **isLooping()** và **setLooping()** để thiết lập chế độ phát lại cho tập tin media.
 - o Ví dụ:

```
if (!mediaPlayer.isLooping())
    mediaPlayer.setLooping(true);
```

- o Thiết lập Wake Clock giữ cho màn hình luôn mở khi đang phát bằng phương thức **setScreenOnWhilePlaying()**
 - o Ví dụ:

```
mediaPlayer.setScreenOnWhilePlaying(true);
```

- o Điều khiển âm thanh bằng phương thức **setVolume()**
 - o Ví dụ:

```
mediaPlayer.setVolume(1f, 0.5f);
```

2. THU ÂM THANH VÀ HÌNH ẢNH (RECODING)

- Android cung cấp 2 lựa chọn trong việc thu âm thanh và hình ảnh:
 - o Sử dụng Intent để khởi chạy ứng dụng Video Camera. Lựa chọn này cho phép chỉ định nơi lưu trữ, định dạng và chất lượng của hình ảnh thu được.
 - o Sử dụng lớp Media Recorder để xây dựng các thành phần UI, các thiết lập record cho ứng dụng.
 - o Sử dụng Intent để thu hình ảnh: bằng cách truyền action ACTION_VIDEO_CAPTURE vào một Intent và gửi Intent này đến một Activity khác xử lý và trả về kết quả thu được.

- Ví dụ:

```
Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
startActivityForResult(intent, RECORD_VIDEO);
```

- MediaStore hỗ trợ hai URI cho phép lựa chọn trong quá trình thu hình ảnh:
 - o EXTRA_OUTPUT: cho phép tùy chọn nơi lưu trữ.
 - o EXTRA_VIDEO_QUALITY: cho phép tùy chọn chất lượng hình ảnh thu được.
- Ví dụ sử dụng Intent để thu hình ảnh:

```
private static int RECORD_VIDEO = 1;
private static int HIGH_VIDEO_QUALITY = 1;
private static int MMS_VIDEO_QUALITY = 0;
private void recordVideo(Uri outputpath)
{
    Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
    if(outputpath != null)
        Intent.putExtra(MediaStore.EXTRA_OUTPUT, output);
    Intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, HIGH_VIDEO_QUALITY);
    startActivityForResult(intent, RECORD_VIDEO);
}
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    if(requestCode == RECORD_VIDEO){
        Uri recordVideo = data.getData();
    }
}
```

- Sử dụng Media Recorder để thu âm thanh và hình ảnh: cần tạo đối tượng thuộc lớp MediaRecorder để sử dụng.
- Ví dụ:

```
MediaRecorder mediaRecorder = new MediaRecorder();
```

- Cần đăng ký quyền sử dụng cho ứng dụng trong file AndroidManifest.xml

```
<uses-permission android:name="android.permission.RECORD_AUDIO"/>  
<uses-permission android:name="android.permission.RECORD_VIDEO"/>
```

- MediaRecorder quản lý các tập tin media và luồng xử lý thông qua một tập các trạng thái sau:
 - o Khởi tạo đối tượng MediaPlayer.
 - o Chỉ định nguồn vào của thiết bị thu.
 - o Thiết lập định dạng của tập tin đầu ra.
 - o Thiết lập các chỉ số như: bộ mã hóa, chất lượng hình ảnh, dung lượng xuất ra...
 - o Chọn tập tin để xuất.
 - o Chuẩn bị thu âm & hình ảnh.
 - o Tiến hành thu âm & hình ảnh.
 - o Hoàn thành quá trình thu.
- Ví dụ tạo MediaRecorder

```
MediaRecorder mediaRecorder = new MediaRecorder();  
  
// Cấu hình các thông tin đầu vào  
mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);  
mediaRecorder.setVideoSource(mediaRecorder.VideoSource.CAMERA);  
  
// Thiết lập định dạng các thông tin đầu ra  
mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);  
  
//Chỉ định mã hóa audio và video  
mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);  
mediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.DEFAULT);  
  
// Chỉ định tập tin đầu ra  
mediaRecorder.setOutputFile("/sdcard/taptinxuatra.mp4");  
  
//chuẩn bị record  
mediaRecorder.prepare();
```

- o Tiến hành thu, dừng thu và giải phóng tài nguyên

```
mediaRecorder.start();  
mediaRecorder.stop();
```

```
mediaRecorder.release();
```

- Tạo màn hình Preview trong lúc record

```
public class MyActivity extends Activity implements SurfaceHolder.Callback
{
    private MediaRecorder mediaRecorder;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        SurfaceView surface = (SurfaceView) findViewById(R.id.surface);
        SurfaceHolder holder = surface.getHolder();
        holder.addCallback(this);
        holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
        holder.setFixedSize(400, 300);
    }
    public void surfaceCreated(SurfaceHolder holder)
    {
        if(mediaRecorder == null)
        {
            try
            {
                mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
                mediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
                mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);
                mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
                mediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.DEFAULT);
                mediaRecorder.setOutputFile("/sdcard/taptinxuatra.mp4");
                mediaRecorder.setPreviewDisplay(holder.getSurface());
                mediaRecorder.prepare();
            }
            catch(IllegalArgumentException e){
                Log.d("MEDIA_PALYER", e.getMessage());
            }
            catch(IllegalStateException e){
                Log.d("MEDIA_PALYER", e.getMessage());
            }
            catch(IOException e)
            {
                Log.d("MEDIA_PALYER", e.getMessage());
            }
        }
    }
}
```

3. CAMERA

3.1. Điều khiển chụp hình với Camera

- Sử dụng Intent để gửi hành động ACTION_IMAGE_CAPTURE đến activity xử lý và nhận kết quả trả về.

- Ví dụ:

```
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
startActivityForResult(intent, TAKE_PICTURE);
```

- Hành động CAPTURE trả về 2 kiểu hình:

- o **Thumbnail**: định dạng ảnh bitmap được trả về cho người dùng xử lý trong ứng dụng.
- o **FullImage**: sử dụng URI để lấy kết quả trả về một ảnh hoàn chỉnh và chỉ định lưu trong MediaStore.

- Ví dụ sử dụng Intent để lấy về dữ liệu của ACTION_IMAGE_CAPTURE

```
private static int TAKE_PICTURE = 1;
private Uri outputFileUri;
private void getThumbnailPicture()
{
    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    File file=new File(Enviroment.getExternalStorageDicrectory(),"hinh.jpg");
    outputFileUri = Uri.fromFile(file);
    intent.putExtra(MediaStore.EXTRA_OUTPUT, outputFileUri);
    startActivityForResult(intent, TAKE_PICTURE);
}
```

- Nhận kết quả trả về và xử lý

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    if( requestCode == TAKE_PICTURE){
        //Kiểm tra nếu kết quả bao gồm 1 thumbail Bitmap
        if(data != null){
            if(data.hasExtra("data")){
                Bitmap thumbail = data.getParcelableExtra("data");
            }
        }
        else{
            //int outputFileUri
        }
    }
}
```

- Cần đăng ký trong file AndroidManifest để sử dụng Camera:

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


- Khởi tạo đối tượng Camera để tùy chỉnh và sử dụng:

```
Camera camera = Camera.open();  
[....Làm việc với camera...]  
camera.release();
```

- Tùy chỉnh Camera thông qua đối tượng của Parameter trong lớp Camera:

```
Camera.Parameters parameters = camera.getParameters();  
[... make changes ... ]  
Camera.setParameter(parameters);
```

- Một số thiết lập đối tượng Parameter

- [get/set]ScreenMode
- [get/set]FlashMode
- [get/set]WhiteBalance
- [get/set]ColorEffect
- [get/set]FocusMode

- Tạo màn hình Preview

```
public class MyActivity extends Activity implements SurfaceHolder.Callback  
{  
    private MediaRecorder mediaRecorder;  
    @Override  
    public void onCreate(Bundle savedInstanceState)  
    {  
        Super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        SurfaceView surface = (SurfaceView) findViewById(R.id.surface);  
        SurfaceHolder holder = surface.getHolder();  
        holder.addCallback(this);  
        holder.setFixedSize(400, 300);  
    }  
    public void surfaceCreate(SurfaceHolder holder)  
    {  
        if(mediaRecorder == null)  
        {  
            try  
            {  
                camera = camera.open();  
                camera.setPreviewDisplay(holder);  
                camera.startPreview();  
            }  
            catch (Exception e)  
            {  
                // Handle exception  
            }  
        }  
    }  
}
```

```

        [ ... draw on surface ... ]
    }
    catch (IOException e)
    {
        Log.d("CAMERA", e.getMessage());
    }
}
}
public void surfaceDestroyed(SurfaceHolder holder)
{
    camera.stopPreview();
    camere.release();
}
}

```

- Ví dụ tiến hành chụp ảnh và lưu trên SDCard

```

private void takePicture()
{
    camera.takePicture(shutterCallback, rawCallback, jpegCallback);
}
shutterCallback shutterCallback = new ShutterCallback()
{
    public void onShutter()
    {

    }
};

```

- Ví dụ tiến hành chụp ảnh và lưu trên SDCard

```

pictureCallback jpegCallback = new PictureCallback()
{
    public void onPictureTaken(byte[] data, Camera camera)
    {
        //Lưu dữ liệu hình vào SD Card
        FileOutputStream outputStream = null;
        try
        {
            outputStream = new FileOutputStream(" /sdcard/ hinh.jpg");
            outputStream.write(data);
            outputStream.close();
        }
        catch(FileNotFoundException e)
        {
            Log.d("CAMERA", e.getMessage());
        }
    }
}

```

```
    }  
    catch(IOException e)  
    {  
        Log.d("CAMERA", e.getMessage());  
    }  
}  
};
```

3.2. Lưu tập tin Media vào MediaStore

- Có thể lưu tập tin media vào MediaStore bằng 2 cách:
 - o Sử dụng MediaScanner để thông dịch tập tin vào thêm vào một cách tự động.
 - o Thêm mới vào một record trong một Content Provider thích hợp.
- Ví dụ tạo MediaScanner để thêm tập tin media

```
MediaScannerConnectionClient mediaScannerClient = new  
MediaScannerConnectionClient()  
{  
    private MediaScannerConnection msc = null;  
    {  
        msc = new MediaScannerConnection(getApplicationContext(), this);  
        msc.connect ();  
    }  
    public void onMediaScannerConnected(){  
        msc.scanFile(" /sdcard/hinh1.jpg", null);  
    }  
    public void onScanCompleted(String path, Uri uri){  
        msc.disconnect();  
    }  
}  
};
```

- Ví dụ thêm media bằng Content Provider

```
ContentValues content = new ContentValues(3);  
Content.put(Audio.AudioColumns.TITLE, "TrheSoundandtheFury");  
Content.put(Audio.AudioColumnd.DATE_ADDED, System.currentTimeMillis()/1000);  
Content.put(Audio.Media.MIME_TYPE, "audio/ amr");  
ContentResolver resolver = getContentResolver();  
Uri uri = resolver.insert(mediaStore.Video.Media.EXTERNAL_CONTENT_URI, content);  
sendBroadcast(new Intent(Intent.ACTION_MEDIA_SCAN_FILE, uri));
```

Bài 6

TELEPHONY & SMS

1. TELEPHONE

- Tạo cuộc gọi bằng các sử dụng Intent để gọi Dialer có sẵn trong thiết bị. Ví dụ:

```
Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:123456"));
startActivity(intent);
```

- Thực hiện hành động gọi ACTION_CALL bằng cách nhấn nút Call trong thiết bị.
- Một số thông tin cần thiết cho việc quản lý chức năng Phone trên thiết bị:
 - o PhoneType (GSM - CDMA)
 - o UniqueID (IMEI – MIED)
 - o Software version
 - o Number
- Để truy xuất được các thông tin này cần thực hiện đăng ký trong AndroidManifest.xml

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

- Việc truy xuất quản lý bởi lớp **TelephonyManager** với phương thức **getSystemService()**. Ví dụ:

```
String svcName = Context.TELEPHONY_SERVICE;
TelephonyManager telephonyManager =
    (TelephonyManager) getSystemService(svcName);
int phoneType = telephonyManager.getPhoneType();
switch (phoneType) {
    case (TelephonyManager.PHONE_TYPE_CDMA): break;
    case (TelephonyManager.PHONE_TYPE_GSM) : break;
    case (TelephonyManager.PHONE_TYPE_NONE): break;
    default: break;
}
String deviceId = telephonyManager.getDeviceId();
```

```
String softwareVersion = telephonyManager.getDeviceSoftwareVersion();
String phoneNumber = telephonyManager.getLine1Number();
```

- Quản lí trạng thái:
 - o Ví dụ:

```
class MyPhoneStateListener extends PhoneStateListener
{
    @Override
    public void onCallStateChanged(int state, String incomingNumber) {
        super.onCallStateChanged(state, incomingNumber);
        switch(state)
        {
            case TelephonyManager.CALL_STATE_IDLE: break;
            case TelephonyManager.CALL_STATE_RINGING: break;
            case TelephonyManager.CALL_STATE_OFFHOOK: break;
            default: break;
        }
    }
}
```

- o Đọc dữ liệu kết nối và các thay đổi trạng thái thông qua các phương thức ***getDataSet()*** và ***getDataActivity()***

```
int dataActivity = telephonyManager.getDataActivity();
int dataState = telephonyManager.getDataState();
switch (dataActivity)
{
    case TelephonyManager.DATA_ACTIVITY_INT : break;
    case TelephonyManager.DATA_ACTIVITY_OUT : break;
    case TelephonyManager.DATA_ACTIVITY_INOUT : break;
    case TelephonyManager.DATA_ACTIVITY_NONE : break;
}
```

- o Đọc các dữ liệu Network cần thiết:

```
String networkCountry = telephonyManager.getNetworkCountryIso();
String networkOperatorId = telephonyManager.getNetworkOperator();
String networkName = telephonyManager.getNetworkOperatorName();
int networkType = telephonyManager.getNetworkType();
switch (networkType) {
    case (TelephonyManager.NETWORK_TYPE_1xRTT) : break;
    case (TelephonyManager.NETWORK_TYPE_CDMA) : break;
    case (TelephonyManager.NETWORK_TYPE_EDGE) : break;
    case (TelephonyManager.NETWORK_TYPE_EVDO_0) : break;
    case (TelephonyManager.NETWORK_TYPE_EVDO_A) : break;
    case (TelephonyManager.NETWORK_TYPE_GPRS) : break;
    case (TelephonyManager.NETWORK_TYPE_HSDPA) : break;
    case (TelephonyManager.NETWORK_TYPE_HSPA) : break;
    case (TelephonyManager.NETWORK_TYPE_HSUPA) : break;
    case (TelephonyManager.NETWORK_TYPE_UMTS) : break;
    case (TelephonyManager.NETWORK_TYPE_UNKNOWN) : break;
    default: break;
}
```

○ Đọc các dữ liệu SIM:

```
int simState = telephonyManager.getSimState();
switch (simState) {
    case (TelephonyManager.SIM_STATE_ABSENT): break;
    case (TelephonyManager.SIM_STATE_NETWORK_LOCKED): break;
    case (TelephonyManager.SIM_STATE_PIN_REQUIRED): break;
    case (TelephonyManager.SIM_STATE_PUK_REQUIRED): break;
    case (TelephonyManager.SIM_STATE_UNKNOWN): break;
    case (TelephonyManager.SIM_STATE_READY): {
        String simCountry = telephonyManager.getSimCountryIso();
        String simOperatorCode = telephonyManager.getSimOperator();
        String simOperatorName = telephonyManager.getSimOperatorName();
        String simSerial = telephonyManager.getSimSerialNumber();
        break;
    }
    default: break;
}
```

○ Các thay đổi trạng thái Phone được quản lý bởi lớp PhoneStateListener

```
PhoneStateListener phoneStateListener = new PhoneStateListener() {  
    public void onCallForwardingIndicatorChanged(boolean cfi) {}  
    public void onCallStateChanged(int state, String incomingNumber) {}  
    public void onCellLocationChanged(CellLocation location) {}  
    public void onDataActivity(int direction) {}  
    public void onDataConnectionStateChanged(int state) {}  
    public void onMessageWaitingIndicatorChanged(boolean mwi) {}  
    public void onServiceStateChanged(ServiceState serviceState) {}  
    public void onSignalStrengthChanged(int asu) {}  
}
```

- Cần đăng ký với Telephony Manager để khởi tạo các sự kiện muốn lắng nghe để xử lý:

```
telephonyManager.listen(phoneStateListener,  
    PhoneStateListener.LISTEN_CALL_FORWARDING_INDICATOR |  
    PhoneStateListener.LISTEN_CALL_STATE |  
    PhoneStateListener.LISTEN_CELL_LOCATION |  
    PhoneStateListener.LISTEN_DATA_ACTIVITY |  
    PhoneStateListener.LISTEN_DATA_CONNECTION_STATE |  
    PhoneStateListener.LISTEN_MESSAGE_WAITING_INDICATOR |  
    PhoneStateListener.LISTEN_SERVICE_STATE |  
    PhoneStateListener.LISTEN_SIGNAL_STRENGTH);
```

- Hủy đăng ký bằng LISTEN_NONE

```
telephonyManager.listen(phoneStateListener,  
    PhoneStateListener.LISTEN_NONE);
```

- Xây dựng ứng dụng thay thế ứng dụng cuộc gọi có sẵn trong ứng dụng. Bao gồm 2 bước:
 - Chặn những Intent đến của ứng dụng Dialer và xử lý.
 - Tổ chức quản lý các cuộc gọi ra bên ngoài.
- Cần đăng ký các Intent cho Activity xử lý:
 - Intent.ACTION_CALL_BUTTON
 - Intent.ACTION_DIAL

- Intent.ACTION_VIEW

- Ví dụ đăng kí trong Intent trong AndroidManifest

```
<activity
    android:name= ".MyDialerActivity"
    android:label ="@string/app_name">
<intent-filter>
    <action android:name = "android.intent.action.CALL_BUTTON"/>
    <category android:name = "android.intent.category.DEFAULT"/>
</intent-filter>
<intent-filter>
    <action android:name = "android.intent.action.VIEW"/>
    <action android:name = "android.intent.action.DIAL" />
    <category android:name = "android.intent.category.DEFAULT" />
    <category android:name = "android.intent.category.BROWSABLE" />
    <data android:scheme = "tel" />
</intent-filter>
</activity>
```

- Ví dụ xử lý cuộc gọi tới

```
phoneStateListner callStateListener = new PhoneStateListner()
{
    public void onCallStateChanged( int state, String incomingNumber)
    {
    }
};
telephonyManager.listen(callStateListener,
phoneStateListener.LISTEN_CALL_STATE;
```

2. SMS

- Gửi tin nhắn bằng cách tạo một Intent để gọi ứng dụng Message trong thiết bị.

- Ví dụ:

- Tạo tin nhắn SMS

```
Intent smsIntent = new Intent(Intent.ACTION_SENDTO,Uri.parse("sms:123456"));
smsIntent.putExtra("sms_body", "Press send to send me");
startActivity(smsIntent);
```

- Gửi tin nhắn MMS có chứa tập tin Media.

- Ví dụ:

//Tạo tin nhắn MMS

```
Uri attached Uri = Uri.parse("content://media/external/images/media/1");
Intent mmsIntent = new Intent(Intent.ACTION_SEND, attached Uri);
mmsIntent.putExtra("sms_body", "Please see the attached image");
mmsIntent.putExtra("address", "07912355432");
mmsIntent.putExtra(Intent.EXTRA_STREAM, attached Uri);
mmsIntent.setType("image/png");
startActivity(mmsIntent);
```

- SMS tin nhắn được điều khiển bởi SmsManager
 - o Tạo đối tượng SmsManager

```
SmsManager smsManager = SmsManager.getDefault();
```

- o Yêu cầu quyền truy cập khi gửi tin nhắn

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

- Việc gửi tin nhắn được thực hiện bởi phương thức *sendTextMessage* của lớp SmsManager
 - o Cú pháp:

```
sendTextMessage(String destination Address,
                String scAddress,
                String text,
                PendingIntent sentIntent,
                PendingIntent deliveryIntent)
```

- o Ví dụ:

```
String sendTo = "5551234";
String myMessage = "Android supports programmatic SMS messaging!";
smsManager.sendTextMessage(sendTo, null, myMessage, null, null);
```

- Theo dõi, xác nhận SMS tin nhắn đã được gửi bằng cách đăng ký BroadcastReceiver để lắng nghe các hoạt động từ PendingIntent được tạo khi gửi tin nhắn.
 - o sentIntent được gọi lên khi tin nhắn được gửi thành công hay lỗi.

- Activity.RESULT_OK.
- SmsManager.RESULT_ERROR_GENERIC_FAILURE.
- SmsManager.RESULT_ERROR_RADIO_OFF.
- SmsManager.RESULT_ERROR_NULL_PDU.
- deliveryIntent được gọi lên một khi thiết bị đích nhận được tin nhắn.
- Ví dụ về kiểm soát gửi tin nhắn
 - Tạo Pending Intent

```
String SENT_SMS_ACTION = "SENT_SMS_ACTION " ;
String DELIVERED_SMS_ACTION = "DELIVERED_SMS_ACTION " ;
Intent sentIntent = new Intent(SENT_SMS_ACTION);
PendingIntent sentPI = PendingIntent.getBroadcast(getApplicationContext(),0,
                                                                    sentIntent,0);

Intent deliveryIntent = new Intent(DELIVERED_SMS_ACTION);
PendingIntent deliverPI =
    PendingIntent.getBroadcast(getApplicationContext(),0, deliveryIntent,0);
```

- Tạo Broadcast Receiver cho Send_Action

```
registerReceiver(new BroadcastReceiver() {
    @Override
    public void onReceive(Context _context, Intent _intent){
        switch (getResultCode()) {
            case Activity.RESULT_OK: break;
            case SmsManager.RESULT_ERROR_GENERIC_FAILURE: break;
            case SmsManager.RESULT_ERROR_RADIO_OFF: break;
            case SmsManager.RESULT_ERROR_NULL_PDU:break;
        }
    }
},
new IntentFilter(SENT_SMS_ACTION));
```

- Tạo Broadcast Receiver cho Delivery_Action

```
registerReceiver(new BroadcastReceiver() {
    @Override
    public void onReceive(Context _context, Intent _intent){
        [ ... SMS delivered actions ... ]
    }
},
new IntentFilter(DELIVERED_SMS_ACTION));
smsManager.sendTextMessage(sendTo, null, myMessage, sentPI, deliverPI);
```

- Xử lý các tin nhắn dài hơn 160 ký tự bằng phương thức `divideMessage()` và `sendMultipartTextMessage()`.

- o Cú pháp:

```
sendMultipartTextMessage(String destination Address,  
                          String scAddress,  
                          ArrayList<String> parts,  
                          ArrayList<PendingIntent> sentIntents,  
                          ArrayList<PendingIntent> deliveryIntents)
```

- o Ví dụ:

```
ArrayList<String> messageArray = smsManager.divideMessage(myMessage);  
ArrayList<PendingIntent> sentIntents = new ArrayList<PendingIntent>();  
for (int i = 0; i < messageArray.size(); i++)  
    sentIntents.add(sentPI);  
smsManager.sendMultipartTextMessage(sendTo, null, messageArray, sentIntents, null);
```

- Gửi tin nhắn chứa dữ liệu bằng mảng byte

- o Cú pháp:

```
sendDataMessage(String destination Address, String scAddress,  
                short destination Port, byte[] data,  
                PendingIntent sentIntent,  
                PendingIntent deliveryIntent)
```

- o Ví dụ:

```
Intent sentIntent = new Intent(SENT_SMS_ACTION);  
PendingIntent sentPI =  
    PendingIntent.getBroadcast(getApplicationContext(), 0, sentIntent, 0);  
short destinationPort = 80;  
byte[] data = [... your data ... ];  
smsManager.sendDataMessage(sendTo, null, destinationPort, data, sentPI, null);
```

- Để ứng dụng nhận được SMS, cần đăng ký:

- o BroadcastReceiver với action

```
<receiver android:name="MySMSMonitor">
<intent-filter>
<action
    android:name="android.provider.Telephony.SMS_RECEIVED"/>
</intent-filter>
</receiver>
```

- Cung cấp quyền truy cập nhận SMS tin nhắn

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

- Ví dụ tạo Broadcast Receiver để nhận tin nhắn

```
public class MySMSMonitor extends BroadcastReceiver
{
    private static final String ACTION =
        "android.provider.Telephony.SMS_RECEIVED";
    @Override
    public void onReceive(Context context, Intent intent)
    {
        if(intent != null && intent.getAction() != null &&
            ACTION.compareToIgnoreCase(intent.getAction()) == 0)
        {
            Object[] pduArray = (Object[]) intent.getExtras().get("pdu");
            SmsMessage[] messages = new SmsMessage[pduArray.length];
            for( int i = 0; i < pduArray.length; i++)
            {
                Messages[i] = SmsMessage.createFromPdu((byte[])pduArray[i]);
                Log.d("MySMSMonitor", "From: " + messages[i].
                    getOriginatingAddress());
                Log.d("MySMSMonitor", "Msg: " + messages[i].getMessageBody());
                Log.d("MySMSMonitor", "SMS Message Received");
            }
        }
    }
}
```

- Làm việc với các thư mục tin nhắn, cần đăng ký quyền trong AndroidManifest.xml

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

- Thực hiện truy vấn để lấy ra các thông tin về tin nhắn trong các thư mục với URI tương ứng:
 - All: content://sms/all

- Inbox : content://sms/inbox
- Sent: content://sms/sent
- Draft: content://sms/draft
- Outbox: content://sms/outbox
- Failed : content://sms/failed
- Queued: content://sms/queued
- Undelivered: content://sms/undelivered
- Conversations: content://sms/conversations

- Ví dụ truy vấn lấy nội dung các tin nhắn trong Inbox.

```
public class SMSInboxDemo extends ListActivity{
    private ListAdapter adapter;
    private static final Uri SMS_INBOX = Uri.parse("content://sms/inbox");
    @Override
    public void onCreate(Bundle bundle)
    {
        super.onCreate(bundle);
        Cursor c = getContentResolver().Query(SMS_INBOX,null,null,null, null);
        startManagingCursor(c);
        String[] columns = new String[] {"body"};
        int[] names = new int[] {R.id.row};
        adapter=new SimpleCursorAdapter(this,R.layout.sms_inbox,c,columns,names);
        setListAdapter(adapter);
    }
}
```

Bài 7

BỘ CẢM BIẾN

1. GIỚI THIỆU SƠ LƯỢC VỀ CẢM BIẾN

- Cảm biến là phần cứng được tích hợp trên các thiết bị, có tác dụng phản hồi lại các hành động ở thế giới thực vào trong môi trường ứng dụng chạy trên thiết bị đó.
- Các hành động trong cảm biến được thực hiện một chiều và chúng chỉ cho phép thực hiện các hành động mặc định sẵn (trừ NFC).
- GPS cũng là một bộ cảm biến nhưng không được tích hợp vào nền tảng cảm biến trong Android.
- Các loại cảm biến có thể có trong một thiết bị:
 - o Light Sensor.
 - o Proximity Sensor.
 - o Temperature Sensor.
 - o Pressure Sensor.
 - o Gyroscope Sensor.
 - o Accelerometer Sensor.
 - o Magnetic Field Sensor.
 - o Orientation Sensor.
 - o Gravity Sensor (Android 2.3).
 - o Linear Accelerometer Sensor (Android 2.3).
 - o Rotation Vector Sensor (Android 2.3).
 - o Near Field Communication Sensor (Android 2.3).
- Có 2 cách để nhận biết thiết bị hỗ trợ những loại cảm biến nào:
 - o Sử dụng đối tượng thuộc lớp `SensorManager` để thực hiện thao tác lấy về danh sách các loại cảm biến trên thiết bị.
 - o Thiết lập trong tập tin `AndroidManifest` để chỉ định các tính năng thiết bị cần có cho ứng dụng.

```
uses-feature android:name="android.hardware.sensor.proximity"/>
```

- Ví dụ lấy thông tin cảm biến trên thiết bị thông qua `SensorManager`:

```
SensorManager mgr = (SensorManager)this.getSystemService(SENSOR_SERVICE);
```

```
List<Sensor> sensors = mgr.getSensorList(Sensor.TYPE_ALL);
StringBuilder message = new StringBuilder(2048);
message.append("The sensors on this device are:\n");
for(Sensor sensor : sensors) {
    message.append(sensor.getName() + "\n");
    message.append("  Type: " + sensorTypes.get(sensor.getType()) + "\n");
    message.append("  Vendor: " + sensor.getVendor() + "\n");
    message.append("  Version: " + sensor.getVersion() + "\n");
    message.append("  Resolution: " + sensor.getResolution() + "\n");
    message.append("  Max Range: " +
        sensor.getMaximumRange() + "\n");
    message.append("  Power: " + sensor.getPower() + " mA\n");
}
text.setText(message);
```



Hình 7.1

2. LẤY THÔNG TIN VÀ ĐIỀU KHIỂN CẢM BIẾN

- Cần đăng ký một bộ lắng nghe (Listener) sự thay đổi của cảm biến để thực hiện lấy các thông tin.

- Ví dụ: tạo lớp thực thi giao diện SensorEventListener

```
public class MainActivity extends Activity implements SensorEventListener{
    private SensorManager mgr;
    private Sensor light;
    private TextView text;
    private StringBuilder msg = new StringBuilder(2048);

    @Override
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mgr = (SensorManager) this.getSystemService(SENSOR_SERVICE);
        light = mgr.getDefaultSensor(Sensor.TYPE_LIGHT);
        text = (TextView) findViewById(R.id.text);
    }
}
```

- Override các onResume và onPause để thực hiện các thiết lập trên cảm biến.
- Ví dụ: đăng ký và hủy đăng ký bộ lắng nghe

```
@Override
protected void onResume(){
    mgr.registerListener(this, light, SensorManager.SENSOR_DELAY_NORMAL);
    super.onResume();
}

@Override
protected void onPause()
{
    mgr.unregisterListener(this, light);
    super.onPause();
}
```

- Trong phương thức registerListener() ta thiết lập thông số để nắm bắt các giá trị thay đổi của cảm biến.
 - o SENSOR_DELAY_NORMAL
 - o SENSOR_DELAY_UI
 - o SENSOR_DELAY_GAME
 - o SENSOR_DELAY_FASTEST

- Theo dõi các hoạt động của cảm biến
- Ví dụ: thực hiện callback 2 phương thức `onAccuracyChanged()` và `onSensorChanged()`.

```
public void onAccuracyChanged(Sensor sensor, int accuracy)
{
    msg.insert(0, sensor.getName() + "accuracy changed: " + accuracy +
        (accuracy + (accuracy==1?" (LOW)" : (accuracy == 2?" (MED)":" (HIGH)"))
        + "\n");
    text.setText(msg);
    text.invalidate();
}
public void onSensorChangeed(SensorEvent event)
{
    msg.insert(0, "Got a sensor event: " + event.values[0] + "SI lux
        units\n");
    text.setText(msg);
    text.invalidate();
}
```

- Một số vấn đề phát sinh khi sử dụng Cảm biến
 - o Phương thức ***onAccuracyChanged()*** sẽ được gọi lại mỗi khi một loại cảm biến được sử dụng và được thiết lập thông số cao nhất.
 - o Khó truy cập trực tiếp đến các giá trị của của cảm biến theo thời gian cho trước trừ khi thực hiện các phương thức API của driver cảm biến và thiết lập lại giao diện (Interface) để sử dụng.
 - o Tốc độ truy xuất dữ liệu của cảm biến không đủ nhanh để đáp ứng trong một số các tác vụ. Có thể tùy chỉnh trong các phương thức có sẵn của thư viện thông qua cơ chế JNI.
 - o Android 2.1 không hỗ trợ duy trì các cảm biến khi màn hình hiển thị tắt.

3. XỬ LÝ THÔNG TIN MỘT SỐ CẢM BIẾN

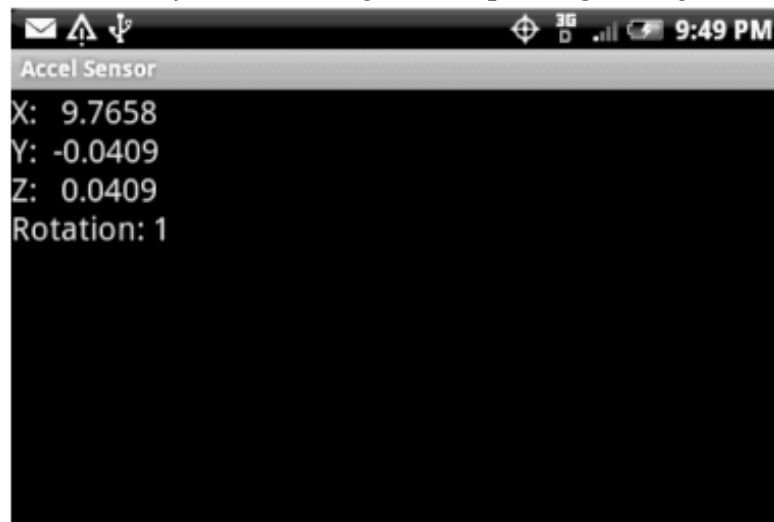
3.1. Accelerometer

- Cảm biến Gia tốc được sử dụng để thu nhận các thay đổi hướng vật lí của thiết bị trong không gian có mối tương quan với trọng lực và các lực tác dụng lên thiết bị.
- Đơn vị tính: m/s²



Hình 7.2

- Các thông số trên các trục x,y,z sẽ được tổng hợp và trả về duy nhất một giá trị Rotation khi thiết bị thay đổi các trạng thái về phương hướng.



Hình 7.3

- Ví dụ sử dụng cảm biến gia tốc

```
SensorManager sm = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
Int sensorType = Sensor.TYPE_ACCELEROMETER;
Sm.registerListener(mySensorEventListener, sm.getDefaultSensor(sensorType),
                    SensorManager.SENSOR_DELAY_NORMAL);
final SensorEventListener mySensorEventListener = new SensorEventListener()
{
    public void onSensorChanged(SensorEvent sensorEvent)
    {
        if(sensorEvent.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
        {
            float xAxis_lateralA = sensorEvent.values[0];
            float yAxis_longitudinalA = sensorEvent.values[1];
            float zAxis_verticalA = sensorEvent.values[2];
        }
    }
};
```

3.2. Near Field Communication Sensor (NFC)

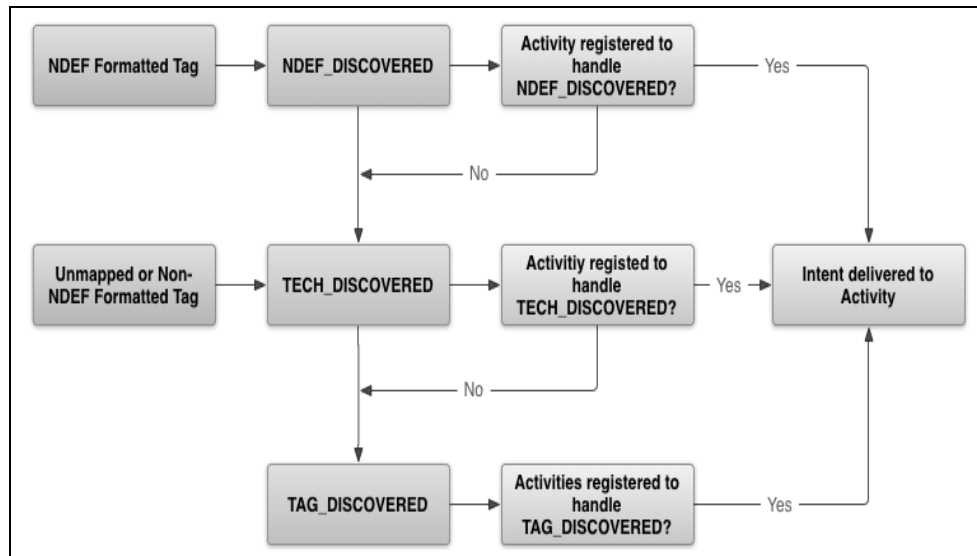
- NFC (Near Field Communication) là một chuẩn giao tiếp giữa thiết bị di động với các thẻ thông tin được gọi là NFC Tags hoặc với các thiết bị khác.
- Có 3 kiểu tiếp nhận dữ liệu bằng NFC:
 - o Tương tác với các thẻ tag cho việc nhận và ghi thông tin trên thiết bị di động.
 - o Thiết bị di động được xem như nơi để các thiết bị khác truy xuất thông tin.
 - o Hai thiết bị cùng trao đổi và tiếp nhận các thông tin của nhau.
- NFC cơ bản làm việc dữ liệu dạng NDEF (NFC Data Exchange Format) và sử dụng hệ thống Tag Dispatch để phân biệt từng loại dữ liệu và khởi động ứng dụng tương ứng.
- Hệ thống Tag Dispatch thực hiện theo cơ chế:
 - o Phân tích dữ liệu NDEF và chuyển dữ liệu sang kiểu MIME hoặc URI.
 - o Đóng gói dữ liệu MIME và URI vào Intent.
 - o Khởi động ứng dụng tương ứng với gói Intent.
- Dữ liệu NDEF được gói trong NdefMessage và chứa dạng các NdefRecord với các trường dữ liệu sau:
 - o 3-bit TNF (Type Name Format): trường chỉ định định dạng dữ liệu.
 - o Variable Length Type: kiểu độ dài dữ liệu.
 - o Variable Length ID: Id chỉ định record
 - o Variable Length Payload: dữ liệu của record.
- Cảm biến NFC được điều khiển bởi NFCAdapter để nhận các gói Intent:

```
NfcManager manager =  
    (NfcManager)context.getSystemService(Context.NFC_SERVICE);  
NfcAdapter adapter = manager.getDefaultAdapter();
```

- Cần thực hiện phương thức **isEnabled()** để kiểm tra trạng thái của cảm biến NFC.
- Không có phương thức cho phép tắt mở NFC.

```
startActivityForResult( new Intent(  
    android.provider.Settings.ACTION_WIRELESS_SETTINGS), 0);
```

- Thứ tự ưu tiên xử lý các gói Intent được gửi tới thiết bị:
 - o ACTION_NDEF_DISCOVERED: intent được sử dụng để khởi chạy một Activity trong ứng dụng có đăng ký xử lý gói intent này.
 - o ACTION_TECH_DISCOVERED: nếu không có Activity nào đăng ký việc xử lý gói intent đến, hệ thống sẽ khởi động ứng dụng thích hợp.
 - o ACTION_TAG_DISCOVERED: hệ thống chuyển sang xử lý kiểu dữ liệu Tag thông thường.



Hình 7.4

- Cần đăng kí trong AndroidManifest để sử dụng NFC

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

- Lưu ý phiên bản của ứng dụng:

```
<uses-sdk android:minSdkVersion= "10" />
```

- Cho phép ứng dụng chỉ cài đặt trên các thiết bị có hỗ trợ NFC:

```
<uses-feature android:name= "android.hardware.nfc" android:required="true"/>
```

Mục lục

Bài 1: Khai thác tài nguyên Internet.....	1
1. Tổng quan tài nguyên Internet	1
2. Sử dụng dịch vụ Download Manager	5
Bài 2: Kết nối các dịch vụ Web thao tác với dữ liệu XML và JSON.....	10
1. Giới thiệu về các dịch vụ Web.....	10
2. Các loại Webservice	11
3. Xây dựng ứng dụng kết nối dịch vụ Web Restful	16
4. Đọc ghi dữ liệu XML.....	19
5. Đọc ghi dữ liệu JSON.....	25
Bài 3: Google Map	29
1. Google Play Service SDK	29
2. Google Maps Android API.....	30
Bài 4: Google Cloud Messaging.....	42
1. Giới thiệu	42
2. Cấu hình cho Google Cloud Messaging	43
Bài 5: Các điều khiển đa truyền thông	56
1. Media Player	56
2. Thu âm thanh và hình ảnh	59
3. Camera	62
Bài 6: Telephony & SMS.....	67
1. Telephone.....	67
2. SMS	71

Bài 7: Bộ cảm biến	77
1. Giới thiệu sơ lược về cảm biến	77
2. Lấy thông tin và điều khiển cảm biến.....	78
3. Xử lý thông tin một số cảm biến.....	80