Location-aware Computing
APIs for Location-based Services

http://www.cs.unibo.it/difelice/

Context-Aware Systems

Prof. Marco Di Felice
Department of Computer Science and Engineering
University of Bologna
APIs for Location-based services

We will still focus on the positioning technologies, but considering available localization APIs.

Source: http://geoawesomeness.com/
Location-aware services

GeoData ACQUISITION → GeoData MODELING → GeoData STORAGE

GeoData Analytics → GeoData VISUALIZATION
Location-aware services

APIs for OUTDOOR LOCALIZATION
(AND GEO-INFORMATION RETRIEVAL)

WEB SERVICES
APIs for Location-based services

- Several geoservices and geolocation APIs are available (at least for outdoor scenarios).

![Geoservices and Geolocation APIs](images)
APIs for Location-based services

- **Geolocation** APIs for location-based services

![Data Formats Used in Geolocation APIs](image1)

![Protocols Used in Geolocation APIs](image2)

APIs for Location-based services

- **Representational State Transfer (REST)** → set of architectural principles for distributed systems.
  1. **Client Server** → Interactions based on a request-response communication pattern.
  2. **Uniform Interfaces** → Unambiguous standard interface for accessing the resources (e.g. the URI).
  3. **Stateless** → client context and state are not stored on the server.
  4. **Cacheable** → data are cached by clients and intermediaries.
  5. **Layered System** → intermediate components can hide what is behind them (e.g. content delivery networks).
Several geoservices and geolocation APIs are available (at least for outdoor scenarios).
APIs for Location-based services

- **W3C Geolocation API** is an effort by the World Wide Web Consortium (W3C) to standardize an interface for retrieving the geographical location information for a client-side (Web) device.

- HTML5 Geolocation API is supported by most of the browser

- **Geolocation data**: latitude, longitude, accuracy, altitude, altitude, altitudeAccuracy, speed, heading, timestamp

- Main methods: getCurrentPosition() and watchPosition()
APIs for Location-based services

HTML5 Geolocation API code example

```java
interface Geolocation {

    void getCurrentPosition(PositionCallback successCallback,
                              optional PositionErrorCallback errorCallback,
                              optional PositionOptions options);

    long watchPosition(PositionCallback successCallback,
                        optional PositionErrorCallback errorCallback,
                        optional PositionOptions options);

    void clearWatch(long watchId);
}
```
APIs for Location-based services

### HTML5 Geolocation API code example

```html
<script>
var x = document.getElementById("demo");
function getLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(showPosition);
    } else {
        x.innerHTML = "Geolocation is not supported by this browser."
    }
}

function showPosition(position) {
    x.innerHTML = "Latitude: " + position.coords.latitude +
        "<br>Longitude: " + position.coords.longitude;
}
</script>
```

APIs for Location-based services

- **HTML5 Geolocation API code example**

<table>
<thead>
<tr>
<th>Property</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>coords.latitude</td>
<td>The latitude as a decimal number (always returned)</td>
</tr>
<tr>
<td>coords.longitude</td>
<td>The longitude as a decimal number (always returned)</td>
</tr>
<tr>
<td>coords.accuracy</td>
<td>The accuracy of position (always returned)</td>
</tr>
<tr>
<td>coords.altitude</td>
<td>The altitude in meters above the mean sea level (returned if available)</td>
</tr>
<tr>
<td>coords.altitudeAccuracy</td>
<td>The altitude accuracy of position (returned if available)</td>
</tr>
<tr>
<td>coords.heading</td>
<td>The heading as degrees clockwise from North (returned if available)</td>
</tr>
<tr>
<td>coords.speed</td>
<td>The speed in meters per second (returned if available)</td>
</tr>
<tr>
<td>timestamp</td>
<td>The date/time of the response (returned if available)</td>
</tr>
</tbody>
</table>

APIs for Location-based services

- Several geoservices and geolocation APIs are available (at least for outdoor scenarios).
APIs for Location-based services

- **Google Maps API**
  - HTTP-based Web service API
  - Web client API (Javascript)
  - Programming language API (Java, Python)
  - Mobile OS API (Android, iOS)

Documentation and source code available at: [https://developers.google.com/maps/web-services/](https://developers.google.com/maps/web-services/)
APIs for Location-based services

Google Maps web services are a collection of **HTTP interfaces** to Google services providing geographic data for **map-related applications**.

- RESTful services; resources are identified by a identifier, and manipulated by using a common set of HTTP methods.
- Outputs are returned in JSON/XML formats.
- A registration key is needed to access the services.

Documentation and source code available at: https://developers.google.com/maps/web-services/

https://console.developers.google.com
APIs for Location-based services

- **Main Google Maps web services**
  - Google Maps Geolocation API
  - Google Maps Elevation API
  - Google Maps Geocoding API
  - Google Maps Directions API
  - Google Maps Roads API
  - Google Maps Time Zone API
  - Google Places API

Documentation and source code available at: https://developers.google.com/maps/web-services/
The Google Maps Geolocation API provides the current location (plus the accuracy radius)

- Geolocation is performed by considering fingerprinting/triangulation techniques, based on cellular towers and WiFi networks the users can detect at their locations (provided as inputs).
- Requests/responses are formatted in JSON, and transmitted using POST methods over HTTPS.
- The API key is needed to identify the application (and the quota)

https://www.googleapis.com/geolocation/v1/geolocate?key=YOUR_API_KEY
APIs for Location-based services

☐ The **request body** must be formatted in **JSON**.

```json
{
  
  "homeMobileCountryCode": 310,
  "homeMobileNetworkCode": 410,
  "radioType": "gsm",
  "carrier": "Vodafone",
  "considerIp": "true",
  "cellTowers": [
    // Cell Tower Objects .
  ],
  "wifiAccessPoints": [
    // WiFi Access Point Objects.
  ]
}
```

- **HOME NETWORK COUNTRY CODE**
- **HOME NETWORK NETWORK CODE**
- **MOBILE RADIO TYPE**: gsm, lte, cdma, wcdma
- Network **provider** name
- Use **IP address** when wireless network info are not available

Documentation and source code available at: [https://developers.google.com/maps/web-services/](https://developers.google.com/maps/web-services/)
APIs for Location-based services

- The **request body** must be formatted in **JSON**.

```json
{
  "cellTowers": [
    {
      "cellId": 42,
      "locationAreaCode": 415,
      "mobileCountryCode": 310,
      "mobileNetworkCode": 410,
      "age": 0,
      "signalStrength": -60,
      "timingAdvance": 15
    }
  ]
}
```

**CELLTOWERS OBJECT EXAMPLE**

- **Unique CELL IDENTIFIER** (mandatory)
- **Area and Network Identifiers** (mandatory)
- **RECEIVED SIGNAL STRENGTH** (in dbm)
- **TIME OF ARRIVAL** from mobile node to the base station
APIs for Location-based services

The request body must be formatted in JSON.

```
"wifiAccessPoints": [
{
  "macAddress": "01:23:45:67:89:AB",
  "signalStrength": -65,
  "age": 0,
  "channel": 11,
  "signalToNoiseRatio": 40
}
]
```

Documentation and source code available at: https://developers.google.com/maps/web-services/
APIs for Location-based services

- The response body must be formatted in JSON.

```json
{
  "location": {
    "lat": 51.0,
    "lng": -0.1
  },
  "accuracy": 1200.4
}
```

- Estimated **latitude** and **longitude of the user** (in degrees)

- **Localization** error, defined as a radius (in meters) of a circle centered at the estimated location returned above.

Documentation and source code available at: [https://developers.google.com/maps/web-services/](https://developers.google.com/maps/web-services/)
APIs for Location-based services

- Main Google Maps web services
  - Google Maps Geolocation API
  - Google Maps Elevation API
  - Google Maps Geocoding API
  - Google Maps Directions API
  - Google Maps Roads API
  - Google Maps Time Zone API
  - Google Places API
The **Google Maps Elevation API** allows determining elevation data for all locations on the surface of the earth, including depth locations on the ocean floor.

- Elevation data are available through measurements or interpolation.
- Requests/responses are formatted in JSON or XML, and transmitted using POST methods over HTTPS.
- Requests can be issued for **discrete locations** or for an **ordered path**.

Documentation and source code available at: [https://developers.google.com/maps/web-services/](https://developers.google.com/maps/web-services/)
Positional requests are indicated through use of the locations parameter, indicating elevation requests for the specific locations (latitude/longitude values).

```json
{ "results" : [ {
    "elevation" : 1608.637939453125,
    "location" : {
      "lat" : 39.7391536,
      "lng" : -104.98470340
    },
    "resolution" : 4.771975994110107
  } ],
  "status" : "OK"
}
```

Minimum distance of interpolation (in meters)
APIs for Location-based services

- **Positional requests** can refer to the elevation data along a path at specified intervals.


- **Path** is a couple of latitude/longitude of the start and end point
- **Sample** is the number of sample points along a path for which to return elevation data

Documentation and source code available at: https://developers.google.com/maps/web-services/
APIs for Location-based services

Positional requests can refer to the elevation data along a path at specified intervals.

```json
{
    "results": [
        {
            "elevation": 4411.941894531250,
            "location": {
                "lat": 36.5785810,
                "lng": -118.2919940
            },
            "resolution": 19.08790397644043
        },
        {
            "elevation": 1381.861694335938,
            "location": {
                "lat": 36.41150289067028,
                "lng": -117.5602607523847
            },
            "resolution": 19.08790397644043
        },
        {
            "elevation": -84.61699676513672,
            "location": {
                "lat": 36.239980,
                "lng": -116.831710
            },
            "resolution": 19.08790397644043
        }
    ],
    "status": "OK"
}
```
APIs for Location-based services

- **Main Google Maps web services**
  - Google Maps Geolocation API
  - Google Maps Elevation API
  - Google Maps **Geocoding** API
  - Google Maps Directions API
  - Google Maps Roads API
  - Google Maps Time Zone API
  - Google Places API

Documentation and source code available at: [https://developers.google.com/maps/web-services/](https://developers.google.com/maps/web-services/)
APIs for Location-based services

- **Geocoding** is the process of converting **addresses** into **geographic** coordinates (latitude/longitude).
  - Reverse geocoding is the process of converting geocoordinates (latitude/longitude) into symbolic addresses (e.g. nation, city, address).

  https://maps.googleapis.com/maps/api/geocode/output?parameters

  - **Output** can be JSON or XML
  - **Parameters** are different for geocoding and reverse geocoding
APIs for Location-based services

- **Geocoding** is the process of converting **addresses** into **geographic** coordinates (latitude/longitude)

**PARAMETERS OF THE GEOCODING SERVICE**

- **Address** → The street address in the format used by the national postal service
- **Components** → A component filter of the geocoding request
  - route matches long or short name of a route.
  - locality matches against both locality and sublocality types.
  - administrative_area matches all the administrative_area levels.
  - postal_code matches postal_code and postal_code_prefix.
  - country matches a country name or a two letter ISO 3166-1 country code.

Documentation and source code available at: https://developers.google.com/maps/web-services/
APIs for Location-based services

https://maps.googleapis.com/maps/api/geocode/json?address=1600+Amphitheatre+Parkway,+Mountain+View,+CA&key=YOUR_API_KEY

```json
{
    "results": [

        {

            "formatted_address": "1600 Amphitheatre Parkway, Mountain View, CA 94043, USA",

            "geometry": {

                "location": {

                    "lat": 37.4224764,
                    "lng": -122.0842499
                },

                "location_type": "ROOFTOP",

                "viewport": {

                    "northeast": {

                        "lat": 37.4238253802915,
                        "lng": -122.0829009197085
                    },

                    "southwest": {

                        "lat": 37.4211274197085,
                        "lng": -122.0855988802915
                    }
                }
            }
        }
    ]
}
```

Documentation and source code available at: https://developers.google.com/maps/web-services/
APIs for Location-based services

- Main Google Maps web services
  - Google Maps Geolocation API
  - Google Maps Elevation API
  - Google Maps Geocoding API
  - Google Maps Directions API
  - Google Maps Roads API
  - Google Maps Time Zone API
  - Google Places API

Documentation and source code available at: https://developers.google.com/maps/web-services/
APIs for Location-based services

- **Google Maps Directions** API allows calculating directions between locations (non-tracking service).
  - Directions may specify origins, destinations and waypoints
  - Result (JSON/XML) contain a list of waypoints
  - Directions can be provided for several modes of transportation, including transit, driving, walking or cycling.
  - Time in traffic can be estimated.
  - Preferences for transit routes (less_walking OR fewer_transfers) can be provided as input.

Documentation and source code available at: https://developers.google.com/maps/web-services/
APIs for Location-based services

- **Google Maps Directions API** allows calculating directions between locations (non-tracking service).

```
https://maps.googleapis.com/maps/api/directions/json?
origin=Toronto&destination=
Montreal&avoid=highways&mode=cycling&key=YOUR_API_KEY
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>origin/destination</td>
<td>address or geocoordinates</td>
</tr>
<tr>
<td>mode</td>
<td>driving, walking, cycling, transit</td>
</tr>
<tr>
<td>waypoints</td>
<td>array of waypoints, expressed as coordinates or addresses</td>
</tr>
<tr>
<td>avoid</td>
<td>tolls, highways, ferries or indoor</td>
</tr>
<tr>
<td>arrival_time/departure_time</td>
<td>used for transit directions</td>
</tr>
<tr>
<td>transit_mode</td>
<td>preferred modes of transit (bus, subway, train, tram, rail)</td>
</tr>
<tr>
<td>units</td>
<td>distance unit to display results</td>
</tr>
</tbody>
</table>

...
APIs for Location-based services

- Main Google Maps web services
  - Google Maps Geolocation API
  - Google Maps Elevation API
  - Google Maps Geocoding API
  - Google Maps Directions API
  - Google Maps Roads API
  - Google Maps Time Zone API
  - Google Places API
APIs for Location-based services

- **Google Maps Roads** API allows mapping GPS coordinates to the geometry of the road, and to determine the speed limit along the road segments.

  ![Google Maps Roads API](https://roads.googleapis.com/v1/speedLimits?parameters&key=YOUR_API_KEY)

- **path** → List of coordinates (latitude/longitude), separated by commas
- **placeId** → place ID of the road segment, returned by the snapToRoads method
- **units** → speed limits in kilometers or miles per hour.

Documentation and source code available at: [https://developers.google.com/maps/web-services/](https://developers.google.com/maps/web-services/)
APIs for Location-based services

https://roads.googleapis.com/v1/speedLimits?placeId=ChIJ1Wi6I2pNFmsRQL9GbW7qABM&placeId=ChIJ58xCoGlNFmsRUEZUbW7qABM&key=YOUR_API_KEY

"speedLimits": [
  {
    "placeId": "ChIJ1Wi6I2pNFmsRQL9GbW7qABM",
    "speedLimit": 60,
    "units": "KPH"
  },
  {
    "placeId": "ChIJ58xCoGlNFmsRUEZUbW7qABM",
    "speedLimit": 60,
    "units": "KPH"
  }
]
APIs for Location-based services

- **Main Google Maps web services**
  - Google Maps Geolocation API
  - Google Maps Elevation API
  - Google Maps Geocoding API
  - Google Maps Directions API
  - Google Maps Roads API
  - Google Maps Time Zone API
  - Google Places API

Documentation and source code available at: [https://developers.google.com/maps/web-services/](https://developers.google.com/maps/web-services/)
APIs for Location-based services

- The **Google Maps Time Zone** API allows requesting the **time zone** for a location on the earth, as well as that **location time offset** from the UTC.


  - location ➔ comma-separated lat,lng tuple (eg. location=-33.86,151.20)
  - timestamp ➔ desired time as seconds since midnight, January 1, 1970 UTC.
APIs for Location-based services

- The **Google Maps Time Zone** API allows requesting the time zone for a location on the earth, as well as that location time offset from the UTC.

```plaintext
https://maps.googleapis.com/maps/api/timezone/json?
location=39.6034810,-119.6822510&
timestamp=1331161200&key=YOUR_API_KEY
```

```json
{
    "dstOffset": 0,
    "rawOffset": -28800,
    "status": "OK",
    "timeZoneId": "America/Los_Angeles",
    "timeZoneName": "Pacific Standard Time"
}
```
APIs for Location-based services

- **Main Google Maps web services**
  - Google Maps Geolocation API
  - Google Maps Elevation API
  - Google Maps Geocoding API
  - Google Maps Directions API
  - Google Maps Roads API
  - Google Maps Time Zone API
  - Google Places API

Documentation and source code available at: https://developers.google.com/maps/web-services/
APIs for Location-based services

The Google Places API returns information about places, defined as establishments, geographic locations, or prominent points of interest.

- **Place Searches** return a list of places based on a user's location or search string.
- **Place Details** requests return information about a specific Place (e.g. reviews)
- **Place Add** allow adding data to the Google Places database
- **Place Photos** gives access to the Place related photos stored in Google's Place database.
- **Place Autocomplete** can be used to automatically fill in the name and/or address of a place
- **Query Autocomplete** can be used to provide a query prediction service for text-based geographic searches, by returning suggested queries as you type.

Documentation and source code available at: https://developers.google.com/maps/web-services/
APIs for Location-based services

- The Google Places API returns information about places, defined as establishments, geographic locations, or prominent points of interest.

Documentation and source code available at: https://developers.google.com/maps/web-services/

- https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=-33.8670522,151.1957362&radius=500&type=restaurant&name=Italia&key=YOUR_API_KEY

- https://maps.googleapis.com/maps/api/place/textsearch/json?query=restaurants+in+Sydney&key=YOUR_API_KEY

- https://maps.googleapis.com/maps/api/place/radarsearch/json?location=51.503186,-0.126446&radius=5000&type=museum&key=YOUR_API_KEY
APIs for Location-based services

- **Google Maps API**
  - HTTP-based Web service API
  - Web client API (Javascript)
  - High-level Programming language API (Java, Python)
  - Mobile OS API (Android, iOS)
Location-aware services

APIs for **OUTDOOR LOCALIZATION**

**MOBILE APPLICATIONS (ANDROID OS)**
Location-aware services

✧ GPS modules might present a serious battery drain problem due to their high computation load.
Location-aware services

✧ GPS modules might present a serious battery drain problem due to their high computation load.

Per-phase consumption

Consumption for different accuracies
GPS modules might present a serious battery drain problem due to their high computation load.

Location-aware services

API vs meter based consumption values

TABLE II. ENERGY CONSUMPTION SENSORS IPHONE4

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Average P</th>
<th>$\Delta P$</th>
<th>Average P</th>
<th>$\Delta P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>102.90mW</td>
<td>---</td>
<td>121.99mW</td>
<td>--</td>
</tr>
<tr>
<td>Gyroscope</td>
<td>153.13mW</td>
<td>50.32mW</td>
<td>175.59mW</td>
<td>53.60mW</td>
</tr>
<tr>
<td>Heading</td>
<td>163.31mW</td>
<td>60.41mW</td>
<td>188.40mW</td>
<td>66.41mW</td>
</tr>
<tr>
<td>Gyroscope and Heading</td>
<td>204.28mW</td>
<td>101.38mW</td>
<td>212.37mW</td>
<td>90.38mW</td>
</tr>
<tr>
<td>Gyroscope Heading &amp; 3G</td>
<td>222.91mW</td>
<td>120.01mW</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

TABLE V. ENERGY CONSUMPTION SENSORS SGIII GT 19300

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Average P</th>
<th>$\Delta P$</th>
<th>Average P</th>
<th>$\Delta P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>219.79mW</td>
<td>--</td>
<td>239.79mW</td>
<td>--</td>
</tr>
<tr>
<td>Gyroscope</td>
<td>270.69mW</td>
<td>50.9mW</td>
<td>273.58mW</td>
<td>44.29mW</td>
</tr>
<tr>
<td>Heading</td>
<td>275.98mW</td>
<td>56.19mW</td>
<td>274.54mW</td>
<td>45.25mW</td>
</tr>
<tr>
<td>Gyroscope and Heading</td>
<td>301.46mW</td>
<td>81.67mW</td>
<td>302.71mW</td>
<td>73.42mW</td>
</tr>
<tr>
<td>Gyroscope Heading &amp; 3G</td>
<td>335.46mW</td>
<td>115.67mW</td>
<td>321.23mW</td>
<td>91.94mW</td>
</tr>
</tbody>
</table>
APIs for Location-based services

HOW to get the current position in Android?

- GPS provider
- Wi-Fi provider
- Cellular provider

LOCATION MANAGER

Location changed!

LOCATION LISTENER

NETWORK PROVIDER
In Android, the battery consumption of Location-aware applications can be managed by properly setting three configuration parameters of the mobile app:

- **Accuracy**: The precision of the location data. In general, the higher the accuracy, the higher the battery drain.
- **Frequency**: How often location is computed. The more frequent location is computed, the more battery is used.
- **Latency**: How quickly location data is delivered. Less latency usually requires more battery.
APIs for Location-based services

- In Android, the battery consumption of Location-aware applications can be managed by properly setting three configuration parameters of the mobile app:

  ✤ **Accuracy**: The precision of the location data. In general, the higher the accuracy, the higher the battery drain.

  ```java
  LocationRequest mLocationRequest = new LocationRequest();
  mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
  ```

  Uses all location providers

  PRIORITY_HIGH_ACCURACY, PRIORITY_BALANCED_POWER_ACCURACY, PRIORITY_LOW_POWER, PRIORITY_NO_POWER

  Passive localization, it exploits localizatio updates from other apps

  [https://developer.android.com/guide/topics/location/battery](https://developer.android.com/guide/topics/location/battery)
In Android, the battery consumption of Location-aware applications can be managed by properly setting three configuration parameters of the mobile app:

- **Frequency**: How often location is computed. The more frequent location is computed, the more battery is used.

```java
LocationRequest mLocationRequest = new LocationRequest();
mLocationRequest.setInterval(10000);
mLocationRequest.setFastestInterval(5000);
```

Interval at which location computed for other apps is delivered to the current app
APIs for Location-based services

In Android, the battery consumption of Location-aware applications can be managed by properly setting three configuration parameters of the mobile app:

- **Latency**: How quickly location data is delivered. Less latency usually requires more battery.

```java
LocationRequest mLocationRequest = new LocationRequest();
mLocationRequest.setInterval(10000);
mLocationRequest.setMaxWaitTime(50000);
```

This value should be considerable larger than the interval.
APIs for Location-based services

- In Android, the battery consumption of Location-aware applications can be managed by properly setting three configuration parameters of the mobile app:

  1. Remove Location Updates if not needed
  2. Set a timeout of when the location update should stop
  3. Use passive location updates and properly set the update parameters

```java
LocationRequest request = new LocationRequest();
request.setInterval(15 * 60 * 1000);
request.setFastestInterval(2 * 60 * 1000);
```

[https://developer.android.com/guide/topics/location/battery](https://developer.android.com/guide/topics/location/battery)
APIs for Location-based services

- **Fused location provider** Android APIs.
  - It merges the localization outputs of different localization providers for better accuracy.
  - It manages/hides the underlying location technology (i.e. GPS, cellular based or WiFi based).
  - It allows the user to specify requirements at a high level, like high accuracy or low power.
  - It supports geofencing and activity recognition.

APIs for Location-based services

_SELECTED TOPIC: APIs for Location-based services

STEP 1 - Request location permissions

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.google.android.gms.location.sample.basiclocationsample">
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
</manifest>

APIs for Location-based services

STEP 2 – Connect to Google Play services

```java
if (mGoogleApiClient == null) {
    mGoogleApiClient = new GoogleApiClient.Builder(this)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .addApi(LocationServices.API)
        .build();
}
...

mGoogleApiClient.connect();
```
APIs for Location-based services

- **STEP 3 – Get the last known location**

```java
public void onConnected(Bundle connectionHint) {
    mLLastLocation = LocationServices.FusedLocationApi.getLastLocation(mGoogleApiClient);
    if (mLastLocation != null) {
        mLLatitudeText.setText(String.valueOf(mLastLocation.getLatitude()));
        mLLongitudeText.setText(String.valueOf(mLastLocation.getLongitude()));
    }
}
```

APIs for Location-based services

⚠️ **STEP 4 – Create a location request**

```java
LocationRequest mLocationRequest = new LocationRequest();
mLocationRequest.setInterval(10000);
mLocationRequest.setFastestInterval(5000);
mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);

LocationSettingsRequest.Builder builder = new LocationSettingsRequest.Builder()
    .addLocationRequest(mLocationRequest);
```

APIs for Location-based services

**STEP 5 – Receive Location updates**

```java
LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient, mLocationRequest, this);
```

```java
public class MainActivity extends ActionBarActivity implements ConnectionCallbacks, OnConnectionFailedListener, LocationListener {
    ...
    @Override
    public void onLocationChanged(Location location) {
        mCurrentLocation = location;
        ...
    }
}
```

APIs for Location-based services

- Geofencing is a technique to define virtual boundaries of a geographic area, and to localize a user over it.

Geofence \( \rightarrow \) point (latitude/longitude) plus a radius

An event is triggered each time the user enters or exits the geofence, or dwells longer than a given interval of time.

Each geofence is associated to an expiration time.

APIs for Location-based services

**STEP 1 – Create a geofence object**

```java
mGeofenceList.add(new Geofence.Builder()
  // Set the request ID of the geofence.
  .setRequestId(entry.getKey())
  .setCircularRegion(
      entry.getValue().latitude,
      entry.getValue().longitude,
      Constants.GEOFENCE_RADIUS_IN_METERS
  )
  .setExpirationDuration( Constants.GEOFENCE_EXPIRATION_IN_MILLISECONDS)
  .setTransitionTypes( Geofence.GEOFENCE_TRANSITION_ENTER | Geofence.GEOFENCE_TRANSITION_EXIT)
  .build());
```

APIs for Location-based services

STEP 2 – Define the initial trigger events

```java
mGeofenceList.add(new Geofence.Builder()
    // Set the request ID of the geofence.
    .setRequestId(entry.getKey())
    .setCircularRegion(
        entry.getValue().latitude,
        entry.getValue().longitude,
        Constants.GEOFENCE_RADIUS_IN_METERS
    )
    .setExpirationDuration(Defaults.GEOFENCE_EXPIRATION_IN_MILLISECONDS)
    .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
        Geofence.GEOFENCE_TRANSITION_EXIT)
    .build());
```

APIs for Location-based services

☐ **STEP 3 – Add** geofences and Intents

```java
LocationServices.GeofencingApi.addGeofences(
    mGoogleApiClient,
    mGeofenceList,
    getGeofencePendingIntent()
)
```

```java
private PendingIntent getGeofencePendingIntent() {
    Intent intent = new Intent(this, GeofenceTransitionsIntentService.class);
    return PendingIntent.getService(this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);
}
```
Location-aware services

APIs for INDOOR LOCALIZATION
(BASED ON WIRELESS TECHNOLOGIES)
APIs for Indoor Localization

- **Indoor Atlas** ([https://www.indooratlas.com](https://www.indooratlas.com))

- Localization by **geomagnetic** technology (i.e. analysis of the Earth’s magnetic field)
- Supports indoor navigation, geo-fencing and way-finding.

SDK available at: [https://www.indooratlas.com](https://www.indooratlas.com)
APIs for Indoor Localization

- **Indoor Atlas** ([https://www.indooratlas.com](https://www.indooratlas.com))

- Client-server architecture, based on **four steps**

  1. Create Location and Add map floor (through a Web application)
  2. Set waypoints (through a mobile app)
  3. Walk through the waypoints, and automatically gather sensor data (e.g. magnetometer)
  4. Import the map, and add location-based features through the mobile application

SDK available at: [https://www.indooratlas.com](https://www.indooratlas.com)
APIs for Indoor Localization

- **Indoor Atlas** (https://www.indooratlas.com)
- **SDK** for Android and IOS operating systems

Location updates are provided by the `IALocationListener` interface and the `onLocationChanged()` callback method, and are expressed as absolute coordinates (LAT; LONG).

```java
private IALocationListener mIALocationListener = new IALocationListener() {
    @Override
    public void onLocationChanged(IALocation location) {
        Log.d(TAG, "Latitude: " + location.getLatitude());
        Log.d(TAG, "Longitude: " + location.getLongitude());
    }
};
```
APIs for Indoor Localization


  - Android/IOS SDK for indoor localization and **Localization based services through Beacons**
  - Define indoor **geofence** regions
  - Monitor the BLE signal
  - Trigger corresponding action based on detected Beacon
APIs for Indoor Localization

- ESTIMOTE (http://www.i-locate.eu/the-virtual-hub/)

Each Beacon is identified by three values:

- **UUID**, commonly represented as a string, e.g. “B9407F30-F5F8-466E-AFF9-25556B57FE6D”
- **major** number, an unsigned short integer, i.e., an integer ranging from 1 to 65535, (0 is a reserved value)
- **minor** number, also an unsigned short integer, like the major number.

Further information available at: http://developer.estimote.com/android/
APIs for Indoor Localization

- **ESTIMOTE** ([http://www.i-locate.eu/the-virtual-hub/](http://www.i-locate.eu/the-virtual-hub/))

**DEFINE THE REGION**

```java
beaconManager = new BeaconManager(getApplicationContext());
// add this below:
beaconManager.connect(new BeaconManager.ServiceReadyCallback() {
    @Override
    public void onServiceReady() {
        beaconManager.startMonitoring(new Region(
            "monitored region",
            UUID.fromString("B9407F30-F5F8-466E-AFF9-25556B57FE6D"),
            22504, 48827));
    }
});
```
APIs for Indoor Localization

- ESTIMOTE (http://www.i-locate.eu/the-virtual-hub/)

```java
beaconManager.setMonitoringListener(new BeaconManager.MonitoringListener() {
    @Override
    public void onEnteredRegion(Region region, List<Beacon> list) {
        DO SOMETHING
    }
    @Override
    public void onExitedRegion(Region region) {
        DO SOMETHING
    }
});
```

Further information available at: http://developer.estimote.com/android/
APIs for Indoor Localization

INDOOR LOCALIZATION DEMO THROUGH BLE BEACONS