Geolocalización: Implementación de ArcGIS Y Leaflet

Katherinne Michelle Pérez Rivas, José Andrés Alonso Oviedo
Dep. Ingeniería informática
Universidad Latinoamericana de Ciencia y Tecnología
San José, Costa Rica
Kperezr614@ulacit.ed.cr
Jalonsoo514@ulacit.ed.cr

***Abstract*—Geolocation has become a key point for the creation of web or mobile applications, as this functionality provides multiple benefits to end users and companies. Therefore, this research proposes an effective solution for the integration of geolocation functionalities and management of service coverage polygons using a Costa Rican geographic information system.**

Keywords—ArcGIS, Leaflet, Geolocation, Coverage polygons.

Resumen. La geolocalización se ha convertido en un punto clave para la creación de aplicaciones web o móviles, ya que esta funcionalidad proporciona múltiples beneficios a los usuarios finales y a las empresas. Por lo que esta investigación propone una solución eficaz para la integración de funcionalidades de geolocalización y manejo de polígonos de cobertura de servicio utilizando un sistema de información geográfica de Costa Rica.

Palabras clave. ArcGIS, Leaflet, Geolocalización, Polígonos de cobertura.

I. INTRODUCCIÓN

"La geolocalización es la capacidad para obtener la ubicación geográfica real de un objeto, como un radar, un teléfono móvil o un ordenador conectado a Internet." [1]. Esta tecnología se ha estado utilizando en los últimos años. Los beneficios de obtener la localización de un usuario son múltiples. Entre ellos se encuentra el conocer si el usuario se encuentra cerca de un punto especifico en el que se encuentra la tienda o local de un negocio.

Empresas como UberEats, Glovo, Rappi, entre otras, utilizan la geoposición para conocer donde la dirección de entrega más rápidamente, o la utilizan para conocer si el punto de entrega de un usuario se encuentra dentro del rango de la cobertura de sus servicios. Es claro que esta funcionalidad se ha propagado rápidamente y que la mayoría de las aplicaciones tanto web como móviles implementan la geolocalización como parte de su funcionamiento general.

II. HISTORIAS DE USUARIO

Para la realización de esta investigación que busca la mejor solución para implementar funcionalidades de geolocalización en el territorio costarricense se tomaron en cuenta las siguientes historias de usuario:

A. HU1

Usuario: Proveedor
Prioridad del negocio: Alta
Descripción: Como proveedor quiero poder obtener las coordenadas exactas (latitud y longitud) por medio de GPS cuando el usuario se encuentre usando un dispositivo móvil con GPS para poder conocer la ubicación de mis usuarios.

B. HU2

Usuario: Proveedor
Prioridad del negocio: Alta
Descripción: Como proveedor quiero poder obtener la siguiente información de mis usuarios:
a) Provincia/Cantón/Distrito/Poblado
b) Coordenadas latitud y longitud
i) Formato simple o decimal (9.7489166, -83.7534256)
ii) Grados decimales (o GD) (9.7489° N 83.7534° O)
iii) Grados, minutos, segundos (GMS) (9°44'56.1" N 83°45'12.3"O)

C. HU3

Usuario: Proveedor
Prioridad del negocio: Alta
Descripción: Como Proveedor quiero poder conocer si mis usuarios se encuentran ubicados dentro de la zona de cobertura para delimitar si se encuentran dentro del rango de servicio.

D. HU4

Usuario: Proveedor
Prioridad del negocio: Alta
Descripción: Como proveedor quiero poder obtener las coordenadas exactas (latitud y longitud) por medio de la dirección IP cuando el usuario se encuentre usando una laptop/desktop para poder conocer la ubicación de mis usuarios.

III. MANUAL DE IMPLEMENTACIÓN

A continuación, se establece el procedimiento a seguir para la implementación completa de funcionalidades de geolocalización y polígonos de cobertura, estas se dividen en las siguientes partes y deben completarse cada una de ellas en el orden presentado en este apartado.

a. Creación de mapa de Costa Rica.

Para comenzar con la aplicación de geolocalización es necesario crear los archivos bases correspondientes; un ".html" y un ".js". Dentro del html incluya las siguientes líneas (CDN) necesarias para la utilización y manipulación de las librerías de Leaflet tanto como de Esri.

```
<!-- Leaftet Draw CDN -->
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/leaflet.draw/0.4.2/leaflet.draw.css"/>

<!-- Bootstrap CSS only -->
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
    integrity="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk"
    crossorigin="anonymous">

<!-- Leaflet's .js and .css from CDN-->
<link rel="stylesheet"
    href="https://unpkg.com/leaflet@1.6.0/dist/leaflet.css"
    integrity="sha512-xwE/Az9zrjBIphAcBb3F6JVqxf46+CDLwfLMHloNu6KEQCAWi6HcDUbeOfBIptF7tcCzusKFjFw2yuvEpDL9wQ=="
    crossorigin=""/>
<script src="https://unpkg.com/leaflet@1.6.0/dist/leaflet.js"
    integrity="sha512-gZwIG9x3wUXg2hdXF6+rVkLF/0Vi9U8D2Ntg4Ga5I5BZpVkVxlJWbSQtXPSiUTtC0TjtGOmxa1AJPuV0CPthew=="
    crossorigin=""></script>
```

Ilustración 1. Importación de CDNs.

Seguidamente, dentro del .js agregar el código base para poder cargar el mapa que se va a desplegar y los datos provenientes de Esri que se mostraran en el mapa.

```
///---------------------------------/
/* BASE MAP AND CONFIG              */
///---------------------------------/
let latitud = 9.933;
let longitud = -84.087;

export default map = L.map('map') // Base map image
            .setView([latitud, longitud], 8); // Map view first loaded

/* OpenStreetMaps base map */
const osmBaseMap = L.tileLayer("https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png",
                    {attribution: '&copy; <a href="http://' +
                    'www.openstreetmap.org/copyright">OpenStreetMap</a>'});

osmBaseMap.addTo(map);

/* ArcGIS CR Data */
// URL ArcGIS Service
const arcgisURL = `https://services.arcgis.com/LjCtRQt1uf8M6LGR/arcgis/rest/services/Distritos_CR/FeatureServer/0`;

// Change map layer style (IF necessary)
let colorizeLayerStyle = (clr, fClr, wgt) => {
    return {color: clr, fillColor: fClr, weight: wgt}
};

// Costa Rican map with information
const locationCR =
    L.esri.featureLayer({
        url: arcgisURL,
        style: () => {return colorizeLayerStyle('#363636', 'rgba(255, 255, 255, 0.5)', 0.1)},
        crossOrigin: null
    });

// Adds CR layer to base map
locationCR.addTo(map);
```

Ilustración 2. Código base geolocalización.

Por último, ejecutamos la aplicación la cual nos presentara el mapa.

b. Creación y configuración de marcador de ubicación.

Tanto Leaflet como Esri ArcGIS poseen en su librería métodos de creación de marcadores. En el caso siguiente se utilizará el de Leaflet.

```
let marker;
function generateMarker(lat, lng, msg) {
    marker = new L.marker([lat, lng]) // Marker position
                    .bindPopup(msg) // Marker information popup
                    .togglePopup()
                    .addTo(map);

    return marker;
};
```

Ilustración 3. Creación de marcador.

c. Creación de formatos de coordenadas.

La funcionalidad de despliegue de coordenadas en sus diferentes formatos DD, DDM y DMS se implementa mediante el conocimiento de sus fórmulas matemáticas para su conversión.

Empezando por crear el elemento en el cuerpo del .html el cual se va a componer de 3 botones referentes a cada formato con la finalidad de oprimir en el botón que se deseen las coordenadas.

```
///---------------------------------/
/* DISPLAY MOUSE COORDINATES        */
///---------------------------------/
map.addEventListener('mousemove', (el) => {
    els.coordinatesDisplay.innerHTML = getFormatedCoord(getFormatBtnValue, el.latlng.lat, el.latlng.lng);
});
```

Ilustración 4. Event listener de formatos.

Posteriormente, la lógica de la conversión de coordenadas se compone de 3 métodos uno para cada formato equivalente a su fórmula matemática. El parámetro que recibe cada método corresponde al grado (latitud o longitud) y a partir del retorna como resultado las coordenadas en el formato respectivo.

Nota: El código puede incorporarse dentro del archivo .js creado al principio, pero para mayor organización y mejor lectura es preferible crear un módulo que después se importará en el .js principal.

```
// Decimal Degree Format
const displayDDCoordinates = (lat, lng) => {
    return `${lat.toFixed(2)}° N ${lng.toFixed(2)}° O`
};

// Decimal Degree Minute Format
const displayDMCoordinates = (lat, lng) => {
    return `${Math.trunc(lat)}° ${minutes(lat).toFixed(1)}' N ${Math.trunc(lng)}° ${minutes(lng).toFixed(1)}' O`;
};

// Degree Minute Second Format
const displayDMSCoordinates = (lat, lng) => {
    return `${Math.trunc(lat)}° ${Math.trunc(minutes(lat))}' ${seconds(lat).toFixed(1)}" N ${Math.trunc(lng)}° ${Math.trunc(minutes(lng))}' ${seconds(lng).toFixed(1)}" O`;
};
```

Ilustración 5. Conversores de formato.

Finalmente, adicionar la función para consumir de manera fácil los formatos y así por medio del selector en él .html cambiar de formato en formato.

```
// Button selected function
let lastActive = els.btnChecked[0];
function isActive(btn) {
    if(!btn.classList.contains('active')){
        lastActive.classList.remove('active');
        lastActive = btn;
        btn.classList.add('active');
    }else{
        lastActive = btn;
    };
    return btn.textContent;
};

// Button value saver
let getFormatBtnValue = 'DD';

els.btnChecked.forEach(btn => {
    btn.addEventListener('click', () => {
        getFormatBtnValue = isActive(btn);
    });
});

// Exported format coordinates function
function getFormatedCoord (formatValue, lat, lng) {
    let coord;
    switch(formatValue){
        case 'DD':
            coord = displayDDCordinates(lat, lng);
            break;
        case 'DDM':
            coord = displayDDMCordinates(lat, lng);
            break;
        case 'DMS':
            coord = displayDMSCordinates(lat, lng);
            break;
    };
    return coord;
};

export {getFormatBtnValue, getFormatedCoord}
```

Ilustración 6. Función coordenadas.

d. Creación de función para la obtención de coordenadas de un usuario.

Para crear una funcionalidad que pueda obtener las coordenadas de un usuario por medio del GPS del móvil o de la dirección IP de una laptop/desktop se debe seguir el procedimiento a continuación:

Primeramente, se debe crear un botón en el cuerpo del archivo **.html** como se muestra en la siguiente ilustración:

```
<body>
    <div id="map"></div>
    <div id="info-pane-container" class="leaflet-bar">
        <div id="info-display"></div>
        <div id="coordinates-display"></div>
        <div class="btn-group">
            <div class="btn btn-secondary formatBtn active">DD</div>
            <div class="btn btn-secondary formatBtn">DDM</div>
            <div class="btn btn-secondary formatBtn">DMS</div>
        </div>
        <hr>
        <div class="btn btn-secondary " id="myLocBtn">Localizarme</div>
```

Ilustración 7. Agregar botón “Localizarme”.

Posteriormente, se debe agregar las siguientes funciones de localización al .js. La función “getCurrentLocation” utiliza el método “latlng” de Leaflet para obtener las coordenadas por medio de GPS o dirección IP.

Al obtener la latitud y longitud de configura el “marker” y el “setCircle” para que este señale en el mapa la ubicación del usuario.

La segunda función, “onGetLocationError” obtiene el error en caso de que la función “getCurrentLocation” falle o no reciba respuesta.

```
// Gets current location

function getCurrentLocation(el, map) {
    const radius = el.accuracy,
          lat = el.latlng.lat,
          lng = el.latlng.lng;

    setMarker(lat, lng, map);

    setCircle(el.latlng, radius, map);
};

function onGetLocationError(el) {
    alert(el.message);
};
```

Ilustración 8. Función de localización.

Por último, se debe agregar un event listener para que al hacer click en el botón agregado, la función de “getCurrentLocation” se ejecute.

```
///----------------------------------/
/* GET CURRENT LOCATION            */
///----------------------------------/
els.getMyLoc.addEventListener('click', () => {
    map.locate({setView: true, maxZoom: 16}); // add "watch: true" for auto update location
    map.on('locationfound', (el) => getCurrentLocation(el, map));
    map.on('locationerror', onGetLocationError);
});
```

Ilustración 9. EventListener para botón de “Localizarme”

Al ejecutar la aplicación y presionar el botón” Localizarme” se obtendrá un resultado similar como el que se muestra a continuación:

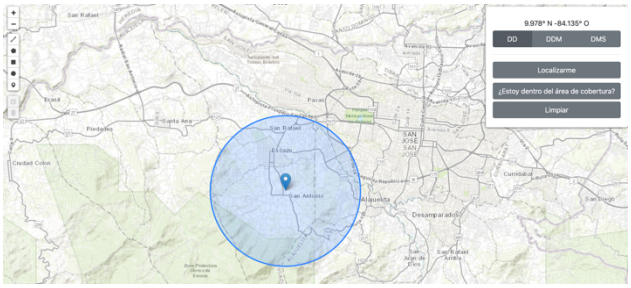

Ilustración 10. Resultado de función de obtener localización.

e. Creación de herramientas para dibujos de polígonos.

Para crear la herramienta para dibujar polígonos de cobertura, se deben agregar el siguiente link y script al archivo .html, este permitirá usar las funcionalidades de dibujo de Leaflet:

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/leaflet.draw/0.4.2/leaflet.draw.css"/>

<script src="https://cdnjs.cloudflare.com/ajax/libs/leaflet.draw/0.4.2/leaflet.draw.js"></script>

Posteriormente se debe agregar el siguiente código en el archivo .js para configurar las funcionalidades de dibujo.

```
var poly;

 var drawnItems = new L.FeatureGroup();
        map.addLayer(drawnItems);

        var drawControl = new L.Control.Draw({
            edit: {
                featureGroup: drawnItems
            }
        });
        map.addControl(drawControl);

        map.on('draw:created', function (e) {
            var type = e.layerType,
                layer = e.layer;
            drawnItems.addLayer(layer);
        });

map.on(L.Draw.Event.CREATED, function (e) {
    var layer = e.layer;
    drawnItems.addLayer(layer);
    console.log(layer.getLatLngs());
    poly = layer;
});
```

Ilustración 11. Función de dibujo de polígonos.

Como se puede observar en la figura, primero se declara una variable que posteriormente se usará para guardar las coordenadas del polígono.

```
var poly;

 var drawnItems = new L.FeatureGroup();
        map.addLayer(drawnItems);
```

Ilustración 12. Variable de coordenadas.

Luego, se declara otra variable y se le asignará un nuevo "FeatureGroup" con el layer del mapa que se representa con "L" en Leaflet. Esto para asignarle dicha variable a un nuevo layer al mapa y que así pueda ser visible el polígono dibujado.

```
var poly;

 var drawnItems = new L.FeatureGroup();
        map.addLayer(drawnItems);
```

Ilustración 13. FeatureGroup.

Posteriormente, se agrega una nueva variable que contendrá los controles de dibujo. Luego de su configuración, se agrega la variable al mapa para que los controles puedan ser visibles.

```
        var drawControl = new L.Control.Draw({
            edit: {
                featureGroup: drawnItems
            }
        });
        map.addControl(drawControl);
```

Ilustración 14. Controles de dibujo.

Además, se debe agregar la siguiente función que establecerá que al seleccionar un tipo de polígono este tome dicha forma.

```
        map.on('draw:created', function (e) {
            var type = e.layerType,
                layer = e.layer;
            drawnItems.addLayer(layer);
        });
```

Ilustración 15. Tipos de polígonos.

Por ultimo se debe crear el evento que obtenga las coordenadas del polígono dibujado y que las guarde en la variable que se ha creado al inicio.

```
map.on(L.Draw.Event.CREATED, function (e) {
    var layer = e.layer;
    drawnItems.addLayer(layer);
    console.log(layer.getLatLngs());
    poly = layer;
});
```

Ilustración 16. Evento de dibujo de coordenadas.

Como resultado la vista tiene los controles de dibujo en la esquina superior derecha, y al dibujar un polígono de cobertura, se visualiza de la siguiente manera:

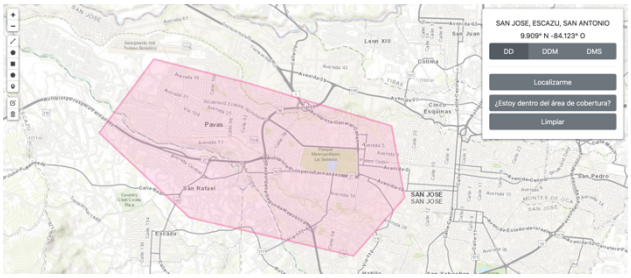

Ilustración 17. Resultado funcionalidad dibujo de polígonos.

f. Creación de función para zonas de cobertura.

Para crear la función que permita conocer si la localización de un usuario se encuentra dentro de un polígono se debe primero crear un botón en el archivo .html para que al finalizar pueda probarse la funcionalidad.

```
<div class="btn btn-secondary " id="areaBtn" >¿Estoy dentro del área de cobertura?</div>
```

Ilustración 18. Botón de cobertura.

Seguidamente se debe agregar la siguiente función en el archivo .js que recibe como parámetros la variable donde se guardaron las coordenadas del polígono y el marcador de la ubicación del usuario. Esta función recorre el array de coordenadas y por medio de ciclos determina si las coordenadas del usuario se encuentran dentro o fuera del polígono.

Si el usuario se encuentra dentro del polígono la variable "inside" será "true", de lo contrario se mantendrá como "false". Para poder visualizarnos, la respuesta se despliega en un "alert" en este caso.

```
function isMarkerInsidePolygon(marker, poly) {
        console.log("entró al inside");
                var inside = false;
                var x = marker.getLatLng().lat, y = marker.getLatLng().lng;
                for (var ii=0;ii<poly.getLatLngs().length;ii++){
                    var polyPoints = poly.getLatLngs()[ii];
                    console.log("entró al primer for");
                    for (var i = 0, j = polyPoints.length - 1; i < polyPoints.length; j = i++) {
                        var xi = polyPoints[i].lat, yi = polyPoints[i].lng;
                        var xj = polyPoints[j].lat, yj = polyPoints[j].lng;
                         console.log("entró al segundo for");
                        var intersect = ((yi > y) != (yj > y))
                            && (x < (xj - xi) * (y - yi) / (yj - yi) + xi);
                        if (intersect) inside = !inside;
                    }
                }
                if (inside == true){
                 alert("SI ESTOY EN EL ÁREA DE COBERTURA");
                }else{
                 alert("NO ESTOY EN EL ÁREA DE COBERTURA");
                }

                return inside;
};
```

Ilustración 19. Función de verificar polígono.

Finalmente, se debe crear un EventListener del botón creado al inicio de este apartado para que ejecute la función creada anteriormente.

```
els.btnArea.addEventListener('click', () => {
  isMarkerInsidePolygon(marker, poly);
});
```

Ilustración 20. EventListener botón cobertura.

El resultado de esta funcionalidad de muestra a continuación, en este caso el usuario se encuentra dentro del polígono dibujado.

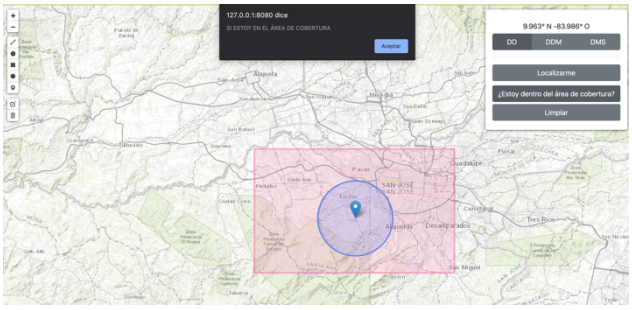

Ilustración 21. Funcionalidad de cobertura.

IV. REFERENCIAS

[1] Tokio School. (s.f). Qué es la geolocalización y su uso en aplicaciones. Recuperado de: https://www.tokioschool.com/noticias/que-es-geolocalizacion-uso-aplicaciones/