****

“AÑO DE LA RECUPERACIÓN Y CONSOLIDACIÓN DE LA ECONOMÍA PERUANA”

# **FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMATICA**

| **SPRINT 3** |
| --- |

**Docente:** Estrada Sanchez Americo

**Curso:** Taller de Proyectos I

**Alumno:**

* Rafaele Huaman Luis Cristian
* Castillo Ccanto Frank
* Espetia Mamani Jhon Cristian

Cusco - 2025

SPRINT 3

| **#** | **HU** | **Actividad** | **Tiempo(h)** |
| --- | --- | --- | --- |
| 30 | HU-08 | Implementar leyenda | 0.50 |
| 31 | HU-09 | Configurar capas inseguras | 1.75 |
| 32 | HU-09 | Colorear mapa por riesgo | 2.00 |
| 33 | HU-09 | Asociar datos con reportes | 2.75 |
| 34 | HU-31 | Implementar selector de idioma | 2.15 |
| 35 | HU-31 | Mostrar lugares turísticos | 2.00 |
| 36 | HU-31 | Filtrar info relevante | 1.50 |
| 37 | HU-10 | Configurar API de rutas | 1.75 |
| 38 | HU-10 | Calcular tiempo | 1.50 |
| **Total** |  |  | **14.40** |

# **Documentación Parte I**

## **Objetivo**

Mostrar en el mapa las zonas seguras e inseguras, colorear según nivel de riesgo y agregar una leyenda para que el usuario entienda los colores.

## **1. Implementación**

### **a) Colorear mapa según riesgo**

Se dibujan círculos sobre el mapa con colores distintos según el nivel de riesgo:

private void dibujarZonasPorRiesgo() {

if(mMap == null) return;

LatLng[] zonas = {zonaSegura, zonaPeligrosa};

int[] riesgos = {1, 3}; // 1 = bajo, 3 = alto

for (int i = 0; i < zonas.length; i++) {

int color;

switch (riesgos[i]) {

case 1: color = Color.parseColor("#5533FF33"); break; // Verde

case 2: color = Color.parseColor("#55FFA500"); break; // Naranja

default: color = Color.parseColor("#55FF3333"); break; // Rojo

}

mMap.addCircle(new CircleOptions()

.center(zonas[i])

.radius(100)

.strokeColor(color)

.fillColor(color));

}

}

### **b) Configurar capas inseguras**

Se agregan marcadores y se permite reportar nuevas zonas inseguras al tocar el mapa:

mMap.setOnMapClickListener(latLng -> {

Reporte reporte = new Reporte(latLng.latitude, latLng.longitude, "Reportado por usuario");

db.collection("reportes").add(reporte);

mMap.addMarker(new MarkerOptions().position(latLng).title("Reporte enviado"));

NotificacionHelper.mostrar(requireContext(),

"Has reportado una zona: " + latLng.latitude + ", " + latLng.longitude);

});

### **c) Implementar leyenda**

Se agrega una vista flotante con leyenda sobre el mapa:

FrameLayout mapContainer = getView().findViewById(R.id.mapContainer);

View legendView = LayoutInflater.from(getContext())

.inflate(R.layout.legend\_layout, mapContainer, false);

mapContainer.addView(legendView);

## **2. Observaciones**

● Verde = zona segura, rojo = zona de alto riesgo, naranja = riesgo medio.

● Los reportes de zonas inseguras se almacenan en Firestore.

● La leyenda permite al usuario interpretar fácilmente los colores del mapa.

# **Documentación Parte II**

**ACTIVIDADES A REALIZAR**

| Asociar datos con reportes |
| --- |
| Implementar selector de idioma |
| Mostrar lugares turísticos |

1. Asociar datos con reportes:  
    CLASE REPORTE:  
    package com.example.myapplicationf.Models;
3. public class Reporte {
4. private double lat;
5. private double lng;
6. private String descripcion;
7. private String nombreLugar; // 🔹 nuevo campo para el lugar
9. // Constructor vacío necesario para Firestore
10. public Reporte() {}
12. // Constructor completo
13. public Reporte(double lat, double lng, String descripcion, String nombreLugar) {
14. this.lat = lat;
15. this.lng = lng;
16. this.descripcion = descripcion;
17. this.nombreLugar = nombreLugar;
18. }
20. // Getters y Setters
21. public double getLat() { return lat; }
22. public void setLat(double lat) { this.lat = lat; }
24. public double getLng() { return lng; }
25. public void setLng(double lng) { this.lng = lng; }
27. public String getDescripcion() { return descripcion; }
28. public void setDescripcion(String descripcion) { this.descripcion = descripcion; }
30. public String getNombreLugar() { return nombreLugar; }
31. public void setNombreLugar(String nombreLugar) { this.nombreLugar = nombreLugar; }
32. }

CLASE DONDE SE CONFIGURA TODO PARA EL REPORTE(galleryfragment):

Aquí se capturan los datos que llegaron del HomeFragment para mostrarlos y asociarlos al nuevo reporte.  
 package com.example.myapplicationf.ui.gallery;

import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import androidx.annotation.NonNull;

import androidx.fragment.app.Fragment;

import com.example.myapplicationf.Models.Reporte;

import com.example.myapplicationf.databinding.FragmentGalleryBinding;

import com.google.firebase.firestore.FirebaseFirestore;

public class GalleryFragment extends Fragment {

private FragmentGalleryBinding binding;

private FirebaseFirestore db;

private double lat, lng;

private String nombreLugar;

@Override

public View onCreateView(@NonNull LayoutInflater inflater,

ViewGroup container, Bundle savedInstanceState) {

binding = FragmentGalleryBinding.*inflate*(inflater, container, false);

View root = binding.getRoot();

db = FirebaseFirestore.*getInstance*();

// 🔹 Recibir lat, lng y nombreLugar desde el HomeFragment

Bundle args = getArguments();

if (args != null) {

lat = args.getDouble("lat", 0);

lng = args.getDouble("lng", 0);

nombreLugar = args.getString("nombreLugar", "Sin nombre");

binding.textGallery.setText("Nuevo reporte:\n" +

"📍 Lugar: " + nombreLugar +

"\nLat: " + lat +

", Lng: " + lng);

} else {

binding.textGallery.setText("Lista de reportes:");

cargarReportes(); // si no viene de un click en el mapa, cargamos todos

}

// 🔹 Guardar reporte cuando se pulse el botón

binding.btnGuardarReporte.setOnClickListener(v -> {

String descripcion = binding.edtDescripcion.getText().toString();

if (descripcion.isEmpty()) {

binding.textGallery.setText("⚠️ Por favor ingresa una descripción.");

return;

}

// Usamos los valores pasados por el HomeFragment

Reporte reporte = new Reporte(lat, lng, descripcion, nombreLugar);

db.collection("reportes").add(reporte)

.addOnSuccessListener(docRef -> {

binding.textGallery.setText("✅ Reporte guardado con éxito\n" +

"📍 Lugar: " + nombreLugar +

"\nLat: " + lat + ", Lng: " + lng);

binding.edtDescripcion.setText(""); // limpiar campo

})

.addOnFailureListener(e -> {

binding.textGallery.setText("❌ Error al guardar reporte: " + e.getMessage());

});

});

return root;

}

// 🔹 Mé*todo para listar reportes cuando no viene desde HomeFragment*

private void cargarReportes() {

db.collection("reportes")

.get()

.addOnCompleteListener(task -> {

if(task.isSuccessful()){

StringBuilder sb = new StringBuilder();

task.getResult().forEach(doc -> {

double lat = doc.getDouble("lat");

double lng = doc.getDouble("lng");

String descripcion = doc.getString("descripcion");

String nombreLugar = doc.getString("nombreLugar");

sb.append("Reporte: ").append(descripcion)

.append("\n📍 Lugar: ").append(nombreLugar)

.append("\nLat: ").append(lat)

.append(", Lng: ").append(lng)

.append("\n\n");

});

binding.textGallery.setText(sb.toString());

} else {

binding.textGallery.setText("No se pudieron cargar los reportes.");

}

});

}

@Override

public void onDestroyView() {

super.onDestroyView();

binding = null;

}

}

Clase HomeFragement:  
 **Pasar datos de HomeFragment a GalleryFragment**

Esto permite que cuando el usuario haga click en el mapa, la latitud, longitud y el nombre del lugar se envíen al fragmento de reportes.

mMap.setOnMapClickListener(latLng -> {

Bundle bundle = new Bundle();

bundle.putDouble("lat", latLng.latitude);

bundle.putDouble("lng", latLng.longitude);

// 🔹 Obtener nombre del lugar con Geocoder

String nombreLugar = "Ubicación seleccionada";

try {

Geocoder geocoder = new Geocoder(requireContext(), Locale.*getDefault*());

List<Address> direcciones = geocoder.getFromLocation(latLng.latitude, latLng.longitude, 1);

if (direcciones != null && !direcciones.isEmpty()) {

Address dir = direcciones.get(0);

// ✅ Forzar dirección legible (sin plus code)

if (dir.getThoroughfare() != null) {

nombreLugar = dir.getThoroughfare(); // Ejemplo: "Av. Jorge Chavez"

if (dir.getSubThoroughfare() != null) {

nombreLugar += " " + dir.getSubThoroughfare(); // Ejemplo: "Av. Jorge Chavez 123"

}

} else if (dir.getAddressLine(0) != null) {

nombreLugar = dir.getAddressLine(0); // Dirección completa

}

}

} catch (Exception e) {

e.printStackTrace();

}

bundle.putString("nombreLugar", nombreLugar);

NavController navController = Navigation.*findNavController*(requireActivity(),

R.id.*nav\_host\_fragment\_content\_contenido\_general*);

navController.navigate(R.id.*nav\_gallery*, bundle);

});

# **Documentación Parte III**

| Asociar datos con reportes |
| --- |
| Implementar selector de idioma |
| Mostrar lugares turísticos |

1. Implementar selector de idioma

**HOME FRAGMENT:**

**// Spinner para seleccionar idioma**

**Spinner spinner = root.findViewById(R.id.*spinnerIdiomas*);**

**String[] idiomas = {"Español", "Inglés"};**

**ArrayAdapter<String> adapter = new ArrayAdapter<>(requireContext(), android.R.layout.*simple\_spinner\_item*, idiomas);**

**adapter.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*);**

**spinner.setAdapter(adapter);**

**spinner.setOnItemSelectedListener(new android.widget.AdapterView.OnItemSelectedListener() {**

**@Override**

**public void onItemSelected(android.widget.AdapterView<?> parent, View view, int position, long id) {**

**switch (position) {**

**case 0: idiomaSeleccionado = "es"; break;**

**case 1: idiomaSeleccionado = "en"; break;**

**}**

**}**

**@Override**

**public void onNothingSelected(android.widget.AdapterView<?> parent) {}**

**});**

**return root;**

**}**

**// 🔹 Aquí aplicamos traducción automática**

**traducirTexto(mensaje, idiomaSeleccionado, textoTraducido -> {**

**NotificacionHelper.*mostrar*(requireContext(), textoTraducido);**

**db.collection("alertas").add(new Alertas(textoTraducido, System.*currentTimeMillis*()));**

**});**

**// 🔹 Mé*todo de traducción usando Google Translation API***

**private void traducirTexto(String texto, String idiomaDestino, TranslationCallback callback) {**

**try {**

**// Aquí simulamos traducción. En producción deberías llamar a tu backend**

**// que use Google Cloud Translation API con la API Key.**

**// Ejemplo de llamada real con Retrofit/Volley.**

**callback.onTranslated(texto); // por ahora devuelve igual**

**} catch (Exception e) {**

**e.printStackTrace();**

**callback.onTranslated(texto);**

**}**

**}**

1. Mostrar lugares turísticos

// 🔹 Mostrar lugares turísticos cercanos

private void mostrarLugaresTuristicos(LatLng location) {

String url = "https://maps.googleapis.com/maps/api/place/nearbysearch/json?" +

"location=" + location.latitude + "," + location.longitude +

"&radius=2000&type=tourist\_attraction&key=" + apiKey;

RequestQueue queue = Volley.*newRequestQueue*(requireContext());

JsonObjectRequest request = new JsonObjectRequest(Request.Method.*GET*, url, null,

response -> {

try {

JSONArray results = response.getJSONArray("results");

for (int i = 0; i < results.length(); i++) {

JSONObject place = results.getJSONObject(i);

String name = place.getString("name");

JSONObject geometry = place.getJSONObject("geometry").getJSONObject("location");

double lat = geometry.getDouble("lat");

double lng = geometry.getDouble("lng");

LatLng pos = new LatLng(lat, lng);

mMap.addMarker(new MarkerOptions().position(pos).title(name));

}

} catch (Exception e) {

e.printStackTrace();

}

},

error -> error.printStackTrace()

);

queue.add(request);

}

**Documentación Parte IV**

1️⃣ **Filtrar información relevante**  Se implementaron filtros en el mapa para mostrar zonas de riesgo, lugares turísticos o ambos combinados. Se utilizó un Spinner (spinnerFiltro) que llama a los métodos dibujarZonasDesdeFirestore() y mostrarLugaresTuristicos(LatLng location) para actualizar el mapa según la selección del usuario.

**Fragmento de código:**

**// Spinner Filtro**

**String[] filtros = {"Ninguno", "Zonas de riesgo", "Lugares turísticos", "Ambos"};**

**ArrayAdapter<String> adapterFiltro = new ArrayAdapter<>(requireContext(),**

**android.R.layout.*simple\_spinner\_item*, filtros);**

**adapterFiltro.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*);**

**spinnerFiltro.setAdapter(adapterFiltro);**

**spinnerFiltro.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {**

**@Override**

**public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {**

**if (mMap == null || ultimaUbicacion == null) return;**

**limpiarRuta();**

**switch (position) {**

**case 0:**

**dibujarZonasDesdeFirestore();**

**break;**

**case 1:**

**mostrarLugaresTuristicos(new LatLng(ultimaUbicacion.getLatitude(), ultimaUbicacion.getLongitude()));**

**break;**

**case 2:**

**dibujarZonasDesdeFirestore();**

**mostrarLugaresTuristicos(new LatLng(ultimaUbicacion.getLatitude(), ultimaUbicacion.getLongitude()));**

**break;**

**}**

**}**

**// Spinner Modo de transporte**

**Spinner spinnerModo = root.findViewById(R.id.*spinnerModo*);**

**String[] modos = {"A pie", "Bicicleta", "Carro"};**

**ArrayAdapter<String> adapterModo = new ArrayAdapter<>(requireContext(),**

**android.R.layout.*simple\_spinner\_item*, modos);**

**adapterModo.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*);**

**spinnerModo.setAdapter(adapterModo);**

**spinnerModo.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {**

**@Override**

**public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {**

**switch (position) {**

**case 0: modoTransporte = "walking"; break; //**

**case 1: modoTransporte = "bicycling"; break; //**

**case 2: modoTransporte = "driving"; break; //**

**}**

**}**

**@Override**

**public void onNothingSelected(AdapterView<?> parent) { }**

**});**

**2️⃣ Configurar API de rutas  
 Se configuró la Google Directions API para calcular rutas entre un origen y un destino, utilizando Places API para autocompletado. La URL de la API se construye dinámicamente según el modo de transporte y el idioma seleccionado, y la ruta se dibuja en el mapa con Polyline.**

**Fragmento de código:**

**private void calcularRuta(LatLng origen, LatLng destino) {**

**// Limpiar ruta anterior**

**if (rutaActual != null) {**

**rutaActual.remove();**

**rutaActual = null;**

**}**

**// Limpiar marcadores anteriores**

**if (marcadorOrigen != null) marcadorOrigen.remove();**

**if (marcadorDestino != null) marcadorDestino.remove();**

**// Agregar nuevos marcadores**

**marcadorOrigen = mMap.addMarker(new MarkerOptions().position(origen).title("Origen"));**

**marcadorDestino = mMap.addMarker(new MarkerOptions().position(destino).title("Destino"));**

**// Construir URL según modo de transporte**

**String url = construirUrlDirecciones(origen, destino, modoTransporte);**

**RequestQueue queue = Volley.*newRequestQueue*(requireContext());**

**JsonObjectRequest request = new JsonObjectRequest(Request.Method.*GET*, url, null,**

**response -> {**

**try {**

**String status = response.getString("status");**

**Log.*d*("DebugRuta", "Directions API status: " + status);**

**if (!status.equals("OK")) {**

**// Si no hay ruta y el modo era driving, intentar walking**

**if (modoTransporte.equals("driving")) {**

**Log.*d*("DebugRuta", "No se encontró ruta en driving, intentando walking...");**

**String walkingUrl = construirUrlDirecciones(origen, destino, "walking");**

**JsonObjectRequest walkingRequest = new JsonObjectRequest(Request.Method.*GET*, walkingUrl, null,**

**walkingResponse -> procesarRespuestaRuta(walkingResponse, origen, destino),**

**error -> {**

**error.printStackTrace();**

**calcularTiempoAproximado(origen, destino); // fallback si walking también falla**

**});**

**queue.add(walkingRequest);**

**} else {**

**calcularTiempoAproximado(origen, destino); // mostrar tiempo aproximado**

**}**

**return;**

**}**

**// Procesar la respuesta normalmente**

**procesarRespuestaRuta(response, origen, destino);**

**} catch (Exception e) {**

**e.printStackTrace();**

**calcularTiempoAproximado(origen, destino);**

**}**

**},**

**error -> {**

**error.printStackTrace();**

**calcularTiempoAproximado(origen, destino);**

**}**

**);**

**queue.add(request);**

**}**

**3️⃣ Calcular tiempo aproximado  
 Se implementó un cálculo del tiempo estimado de traslado según el modo de transporte (walking, bicycling, driving). Se consideran velocidad promedio y factores de ajuste para simular condiciones reales y se muestra en un TextView.**

**Fragmento de código:**

**private void calcularTiempoAproximado(LatLng origen, LatLng destino) {**

**float[] distancia = new float[1];**

**Location.*distanceBetween*(**

**origen.latitude, origen.longitude,**

**destino.latitude, destino.longitude,**

**distancia**

**);**

**double velocidadMps;**

**switch (modoTransporte) {**

**case "driving":**

**velocidadMps = 4.17; // ~15 km/h promedio urbano realista**

**break;**

**case "bicycling":**

**velocidadMps = 2.78; // ~10 km/h**

**break;**

**default:**

**velocidadMps = 1.11; // ~4 km/h caminando**

**break;**

**}**

**// Factor por curvas, semáforos, tráfico**

**double factorCurvas = 1.10 + Math.*random*() \* 0.15; // +10% a +25% en carro**

**double factorAleatorio = 0.95 + Math.*random*() \* 0.10; // ±5%**

**velocidadMps \*= factorAleatorio / factorCurvas;**

**int tiempoSegundos = (int)(distancia[0] / velocidadMps);**

**int horas = tiempoSegundos / 3600;**

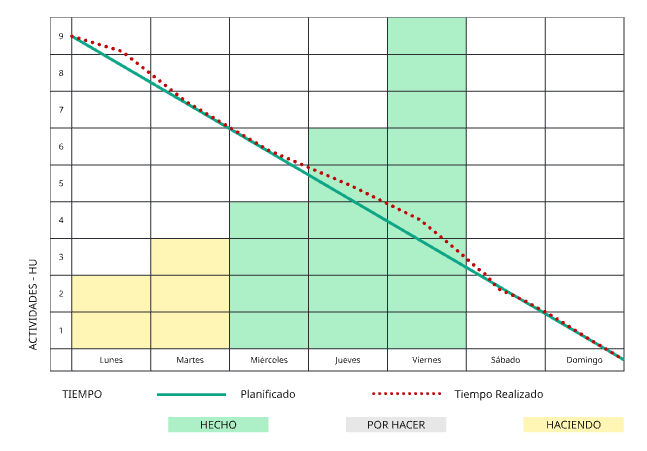
**int minutos = (tiempoSegundos % 3600) / 60;**

**String tiempoTexto = (horas > 0) ? (horas + " h " + minutos + " min") : (minutos + " min");**

**tvTiempo.setText("Tiempo estimado aprox.: " + tiempoTexto);**

**}**

BURNDOWN CHART



| N° | Responsable | Actividades - HU | Nombre | Por hacer | | Haciendo | | Hecho | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Luis Cristian | HU-08 | Implementar leyenda |  | |  | | Implementar leyenda | |
| 2 | Luis Cristian | HU-09 | Configurar capas inseguras |  | |  | | Configurar capas inseguras | |
| 3 | Luis Cristian | HU-09 | Colorear mapa por riesgo |  | |  | | Colorear mapa por riesgo | |
| 4 | Frank | HU-09 | Asociar datos con reportes |  | |  | | Asociar datos con reportes | |
| 5 | Frank | HU-31 | Implementar selector de idioma |  | |  | | Implementar selector de idioma | |
| 6 | Frank | HU-31 | Mostrar lugares turísticos |  | |  | | Mostrar lugares turísticos | |
| 7 | Jhon Cristian | HU-31 | Filtrar info relevante |  | |  | | Filtrar info relevante | |
| 8 | Jhon Cristian | HU-10 | Configurar API de rutas |  | |  | | Configurar API de rutas | |
| 9 | Jhon Cristian | HU-10 | Calcular tiempo |  | |  | | Calcular tiempo | |