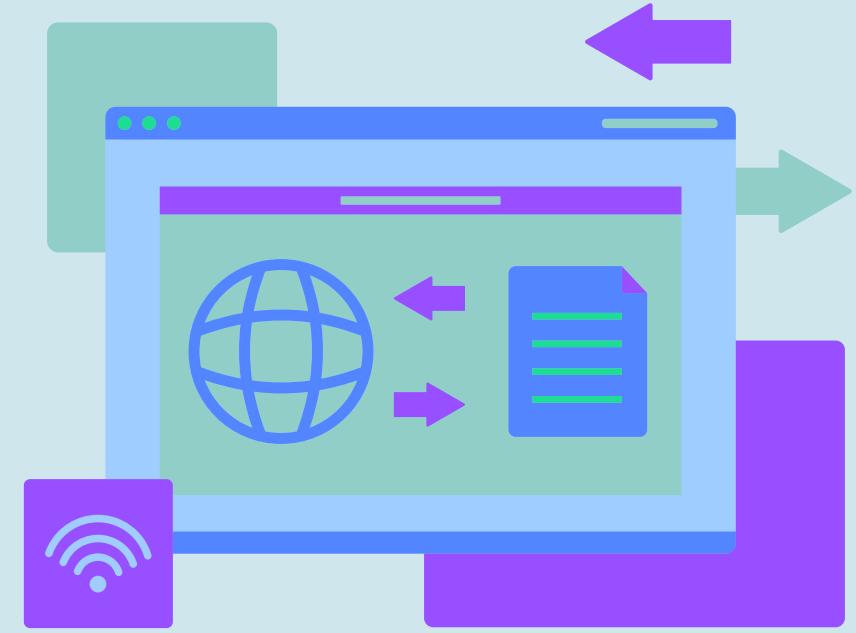


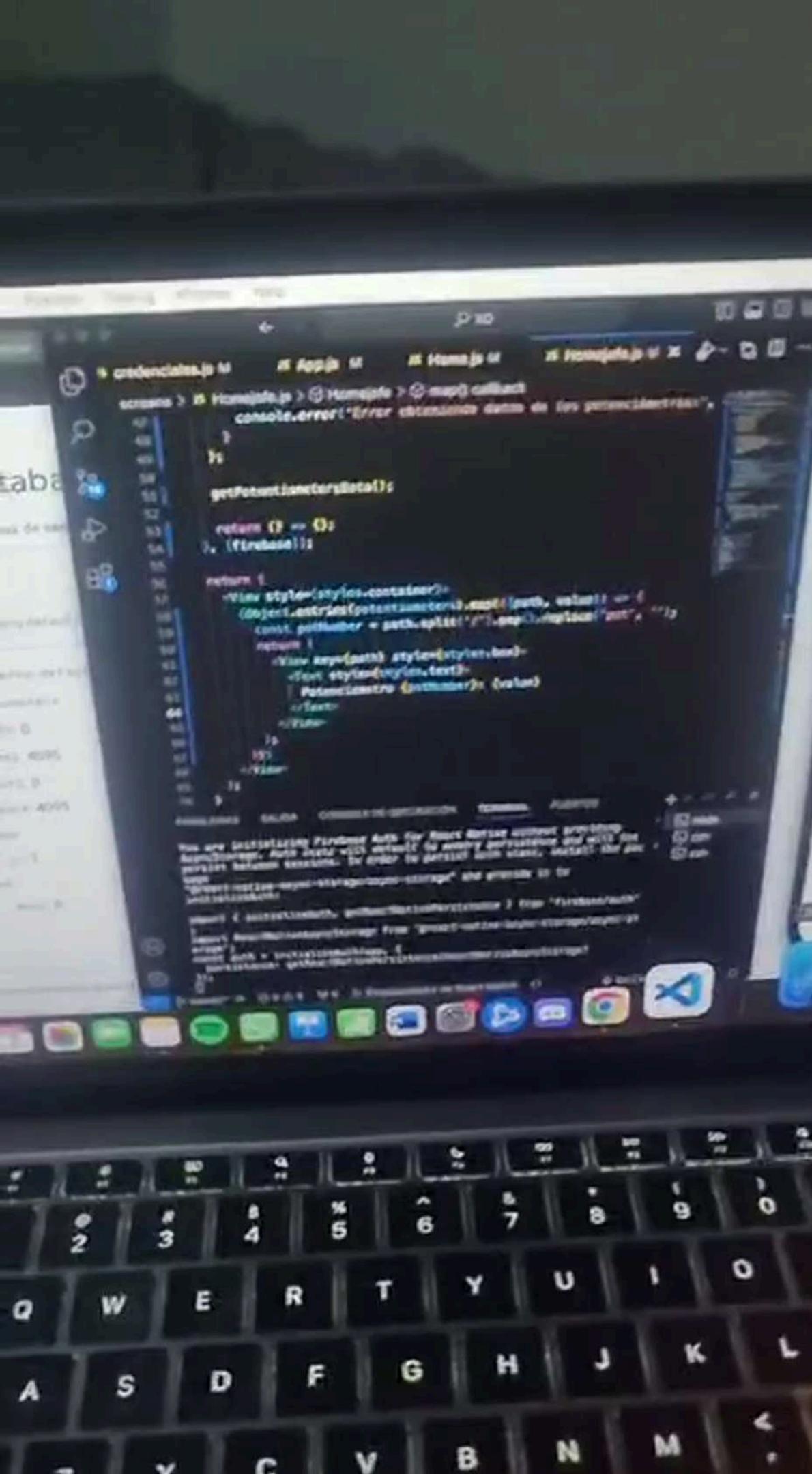
ENTregABe

REPORTE DE PRUEBAS, RETOS Y
LIMITACIONES

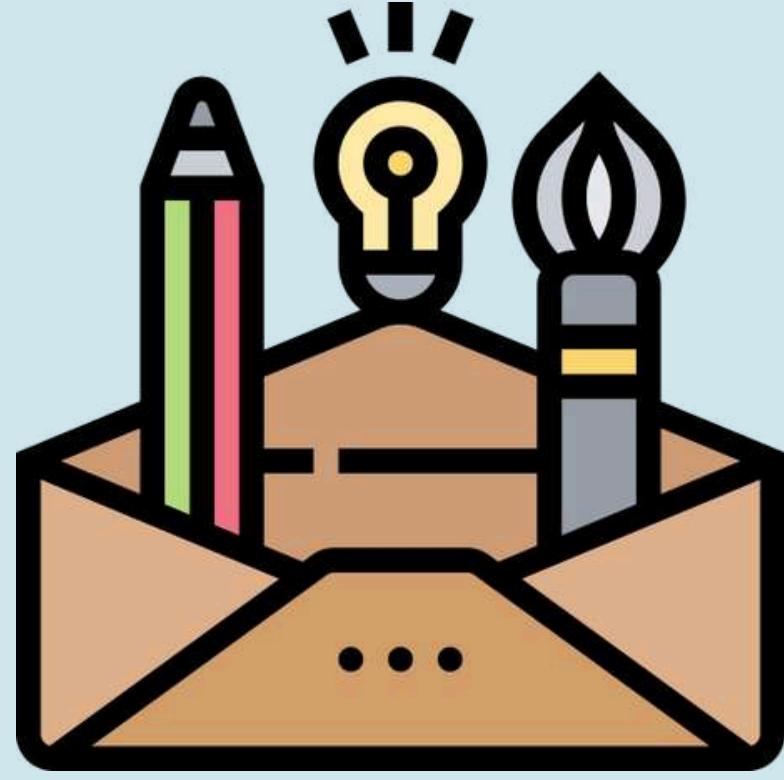
8



DOCUMENTACIÓN: FUNCIONAMIENTO DEL SOFTWARE



**Pruebas de la conexión entre
el software y los
componentes físicos.
Observamos que sí capta la
señal emitida por los
potenciómetros con un poco
de lentitud.**



DOCUMENTACIÓN: REPRESENTACIÓN DE LA MAQUETA

PRETOTIPO

1



Tacho hecho a base de papel con un potenciometro en su interior. Primera perspectiva en 3D del tacho.

- cableado expuesto

2

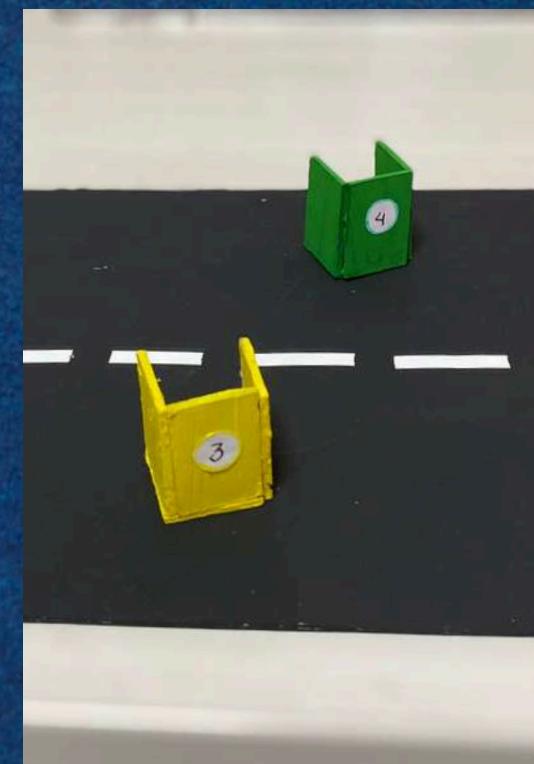


Tacho hecho a base de plastilina con un potenciometro en su interior.

- Poca resistencia.
- cableado expuesto.
- función nula del eje (no gira)

PROTOTIPO

3



Tacho hecho a base de madera con un potenciometro en su interior.

- resistencia.
- cableado expuesto.

4



Tacho hecho con una tapa pegada al potenciometro

- Mayor resistencia.
- Cumple con su función
- No estético

5



Tacho hecho con una tapa pegada al potenciometro

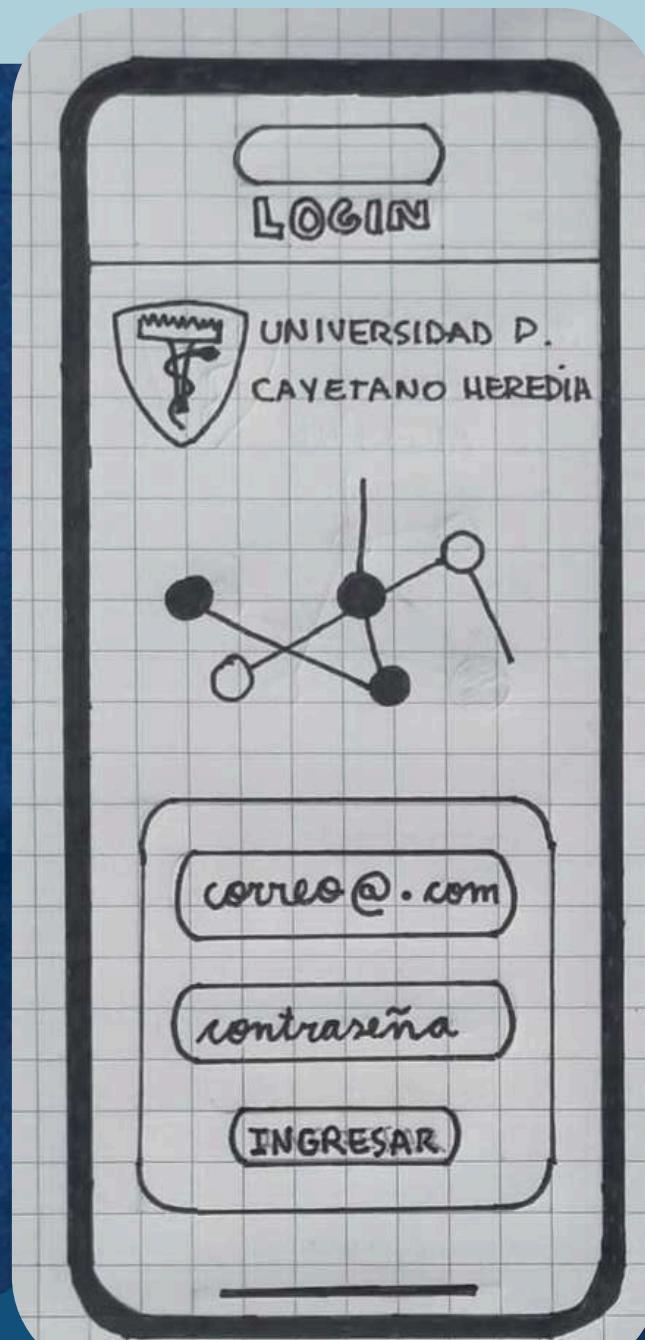
- Mayor resistencia.
- cableado no expuesto.
- Mayor estabilidad en la función del eje (gira).



DOCUMENTACIÓN: VISUALIZACIÓN DEL APLICATIVO

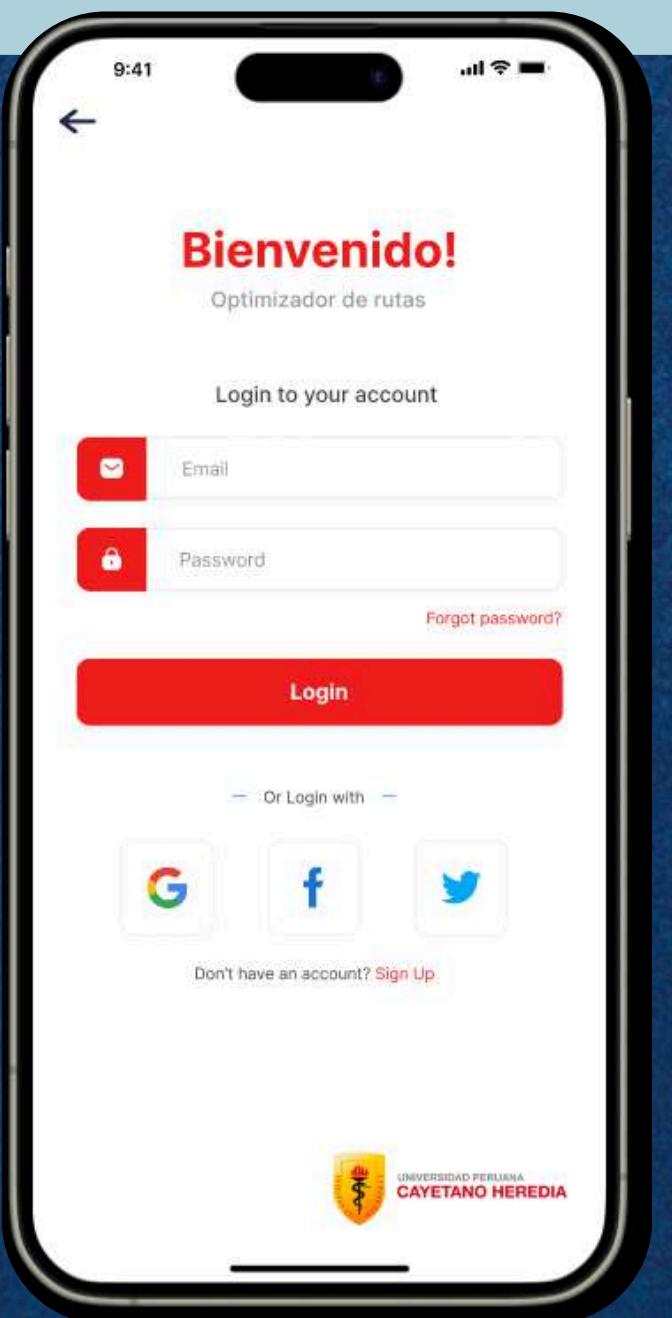
PRETOTIPO DE LA APP

Boceto del interfaz en la aplicación.



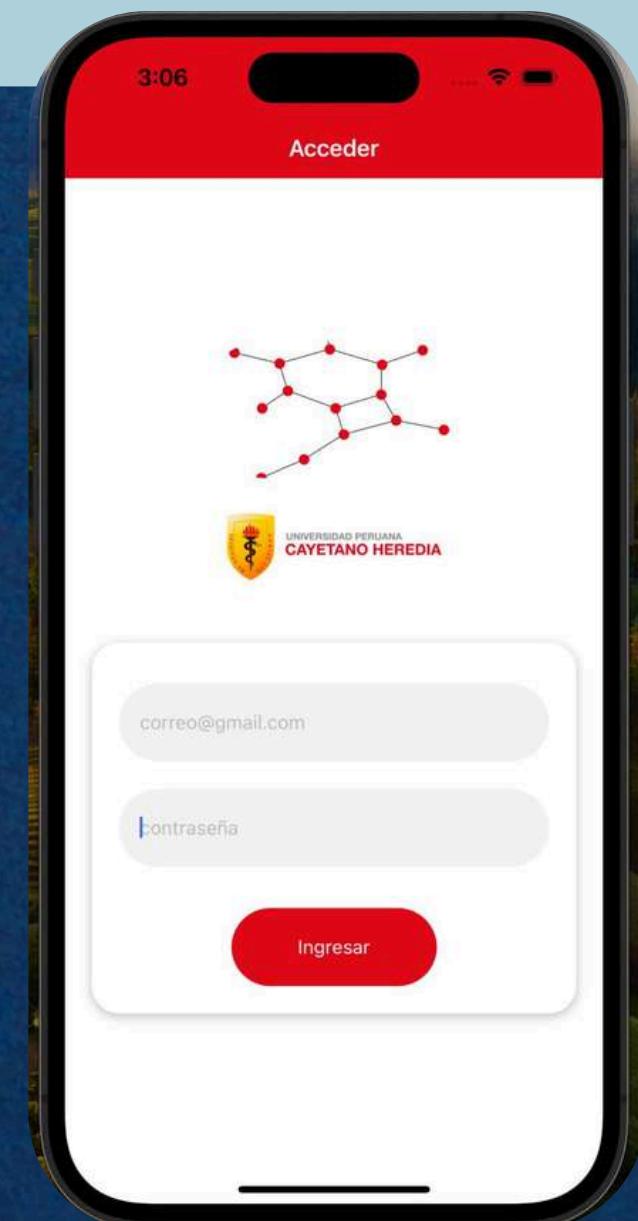
APLICACIÓN CLICKABLE

Creada con Figma para simular la interacción con el usuario.



APLICACIÓN

Creada con la librería React del lenguaje de programación de java



APLICACIÓN

Constructor de la app

```
1 import { StatusBar } from 'expo-status-bar';
2 import { StyleSheet, Text, View } from 'react-native';
3 import 'react-native-gesture-handler';
4 import { createStackNavigator } from '@react-navigation/stack'
5 import { NavigationContainer } from '@react-navigation/native';
6 import Login from './screens/Login';
7 import Home from './screens/Home';
8 import Homejefe from './screens/Homejefe';
9
10 export default function App() {
11   const Stack = createStackNavigator();
12   function MyStack() {
13     return (
14       <Stack.Navigator>
15         <Stack.Screen name="Login" component={Login}
16           options={{
17             title: "Acceder",
18             headerTintColor: "white",
19             headerTitleAlign: "center",
20             headerStyle: {backgroundColor: "#df0818"}
21           } />
22         <Stack.Screen name="Home" component={Home}
23           options={{
24             title: "HOME",
25             headerTintColor: "white",
26             headerTitleAlign: "center",
```

```
27               headerStyle: {backgroundColor: "#df0818"}
28             }}/>
29           <Stack.Screen name="Homejefe" component={Homejefe}
30             options={{
31               title: "ENTORNO LIDER",
32               headerTintColor: "white",
33               headerTitleAlign: "center",
34               headerStyle: {backgroundColor: "#df0818"}
35             }}/>
36         </Stack.Navigator>
37     );
38   }
39
40   return (
41     <NavigationContainer>
42       <MyStack/>
43     </NavigationContainer>
44   );
45 }
46
47 const styles = StyleSheet.create({
48   container: {
49     flex: 1,
50     backgroundColor: '#fff',
51     alignItems: 'center',
52     justifyContent: 'center',
```

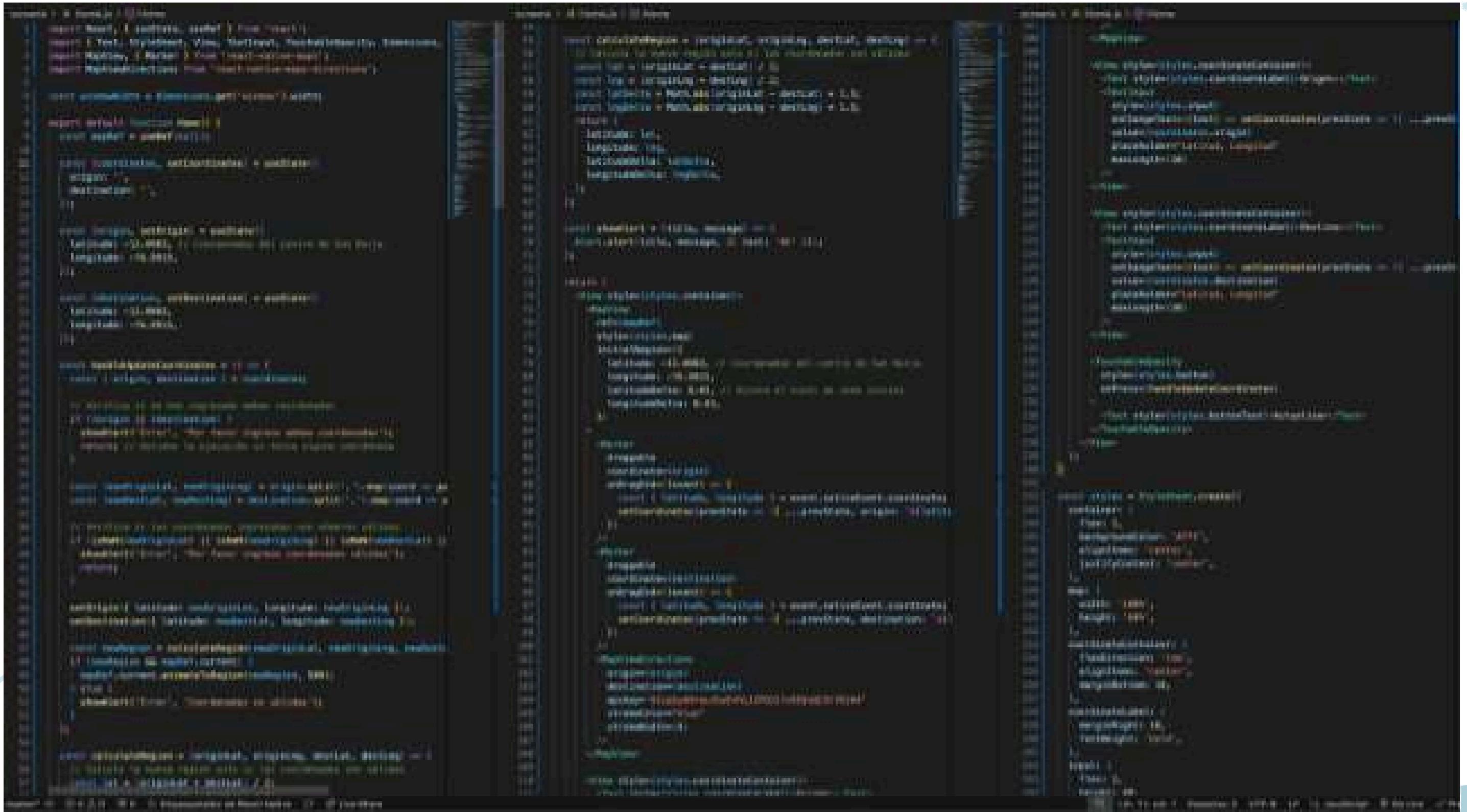
APLICACIÓN

Inicio de sesión

```
55
56      </View>
57    </View>
58
59  );
60}
61
62 const styles = StyleSheet.create({
63   padre: {
64     flex: 1,
65     justifyContent: 'center',
66     alignItems: 'center',
67     backgroundColor: 'white'
68   },
69   profile: {
70     width: 200,
71     height: 100,
72     borderRadius: 100,
73     borderColor: 'white'
74   },
75   tarjeta: {
76     margin: 20,
77     backgroundColor: 'white',
78     borderRadius: 20,
79     width: '90%',
80     padding: 20,
81     shadowColor: '#000',
82     shadowOffset: {
83       width: 0,
84       height: 2
85     },
86     shadowOpacity: 0.25,
87     shadowRadius: 4,
88     elevation: 5,
89   },
90   cajatexto: {
91     paddingVertical: 20,
92     backgroundColor: '#cccccc40',
93     borderRadius: 30,
94     marginVertical: 10
95   },
96   padreboton: {
97     alignItems: 'center',
98   },
99   cajaboton: {
100    backgroundColor: '#df0818',
101    borderRadius: 30,
102    paddingVertical: 20,
103    width: 150,
104    marginTop: 20
105  },
106  textaboton: {
107    textAlign: 'center',
108    color: 'white'
109  }
110});
```

APLICACIÓN

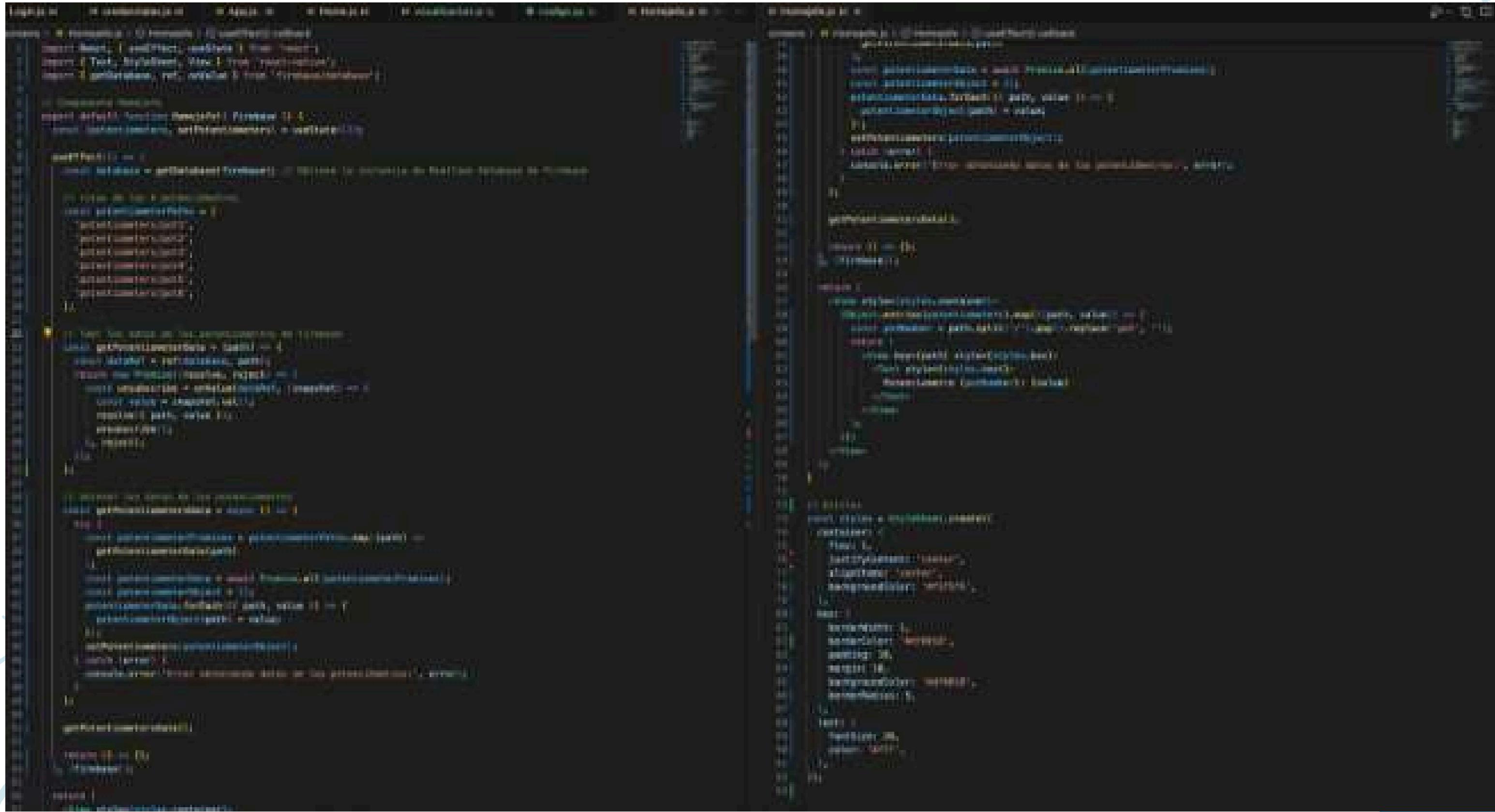
Genera el mapa que variará de acuerdo a la ruta generada



The image shows a terminal window with three separate code editors side-by-side, each displaying a different file from a GitHub repository. The files appear to be related to a project named "mapa". The code is written in a programming language that uses curly braces {}, likely C or C++, and includes various functions, variables, and comments. The terminal window has a dark background with light-colored text.

APLICACIÓN

Guarda los datos en tiempo real de los potenciómetros.



The image shows a screenshot of a computer desktop environment with a dark blue background. In the center, there is a large window titled "IDE" which contains two tabs: "IDE" and "Terminal". The "IDE" tab displays a Java code editor with several files open, including "Main.java", "POTENTIOMETRO.java", and "SERIALPORT.java". The "Terminal" tab shows a command-line interface with several lines of text, likely logs or output from a serial port. To the left of the main window, there is a vertical stack of smaller windows, also titled "IDE" and "Terminal", suggesting a multi-instance or tabbed application. The overall theme is technical and development-oriented.

```
IDE
Main.java
POTENTIOMETRO.java
SERIALPORT.java

Terminal
[Output text from terminal]
```

RESULTADO EJECUCIÓN DEL ALGORITMO EN PYTHON

Iniciando en el nodo X con capacidad 0. Capacidad restante: 6000

Considerando nodo A con capacidad 30 y distancia 4289.39 metros. Prioridad: 0.68

Considerando nodo B con capacidad 20 y distancia 2887.56 metros. Prioridad: 0.45

Considerando nodo C con capacidad 40 y distancia 3349.81 metros. Prioridad: 0.77

Considerando nodo D con capacidad 25 y distancia 2044.60 metros. Prioridad: 0.47

Considerando nodo E con capacidad 35 y distancia 2439.43 metros. Prioridad: 0.64

Considerando nodo F con capacidad 45 y distancia 873.80 metros. Prioridad: 0.68

Considerando nodo G con capacidad 15 y distancia 1388.11 metros. Prioridad: 0.28

Considerando nodo H con capacidad 10 y distancia 678.69 metros. Prioridad: 0.16

Considerando nodo I con capacidad 5 y distancia 547.35 metros. Prioridad: 0.08

Seleccionado nodo C con prioridad 0.77, capacidad 40 y distancia 3349.81 metros.

Moviéndose al nodo C con capacidad 40. Capacidad restante: 5960. Distancia total recorrida: 3349.81 metros.

Considerando nodo A con capacidad 30 y distancia 1524.33 metros. Prioridad: 0.51

Considerando nodo B con capacidad 20 y distancia 1458.13 metros. Prioridad: 0.36

Considerando nodo D con capacidad 25 y distancia 2537.72 metros. Prioridad: 0.50

Considerando nodo E con capacidad 35 y distancia 2352.63 metros. Prioridad: 0.63

Considerando nodo F con capacidad 45 y distancia 2481.91 metros. Prioridad: 0.78

Considerando nodo G con capacidad 15 y distancia 4288.07 metros. Prioridad: 0.47

Considerando nodo H con capacidad 10 y distancia 3565.80 metros. Prioridad: 0.35

Considerando nodo I con capacidad 5 y distancia 3379.36 metros. Prioridad: 0.26

Seleccionado nodo F con prioridad 0.78, capacidad 45 y distancia 2481.91 metros

Recorrido completo: ['X', 'C', 'F', 'E', 'A', 'D', 'B', 'G', 'H', 'I', 'X']

Distancia total recorrida: 23517.31 metros

Recorrido completo: ['X', 'C', 'F', 'E', 'A', 'D', 'B', 'G', 'H', 'I', 'X']

Distancia total recorrida: 23517.31 metros

Capacidad restante del camión: 5775



RETOS, LIMITACIONES Y POSIBLES SOLUCIONES

ELECTRÓNICA

RETOS Y LIMITACIONES

1. Elección de fuente de alimentación: uso de pilas o power bank.
2. Cálculo de la cantidad de voltaje en el circuito.
3. Conexión por Wifi entre el ESP32 y una base de datos en la nube para el envío de valores de la capacidad de los tachos
4. Cableado del ESP32 a los 9 potenciómetros (tachos de basura).

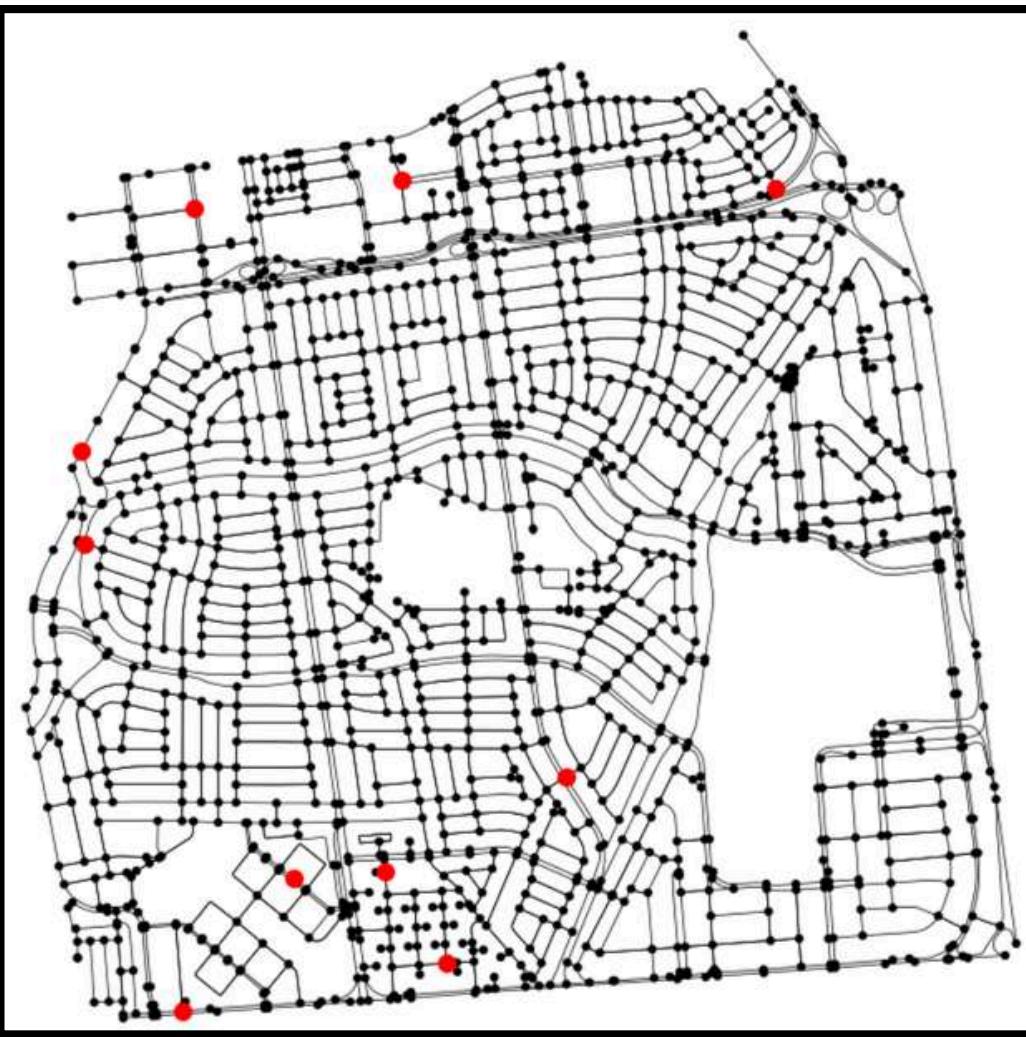
SOLUCIONES

1. Uso de cargador de celular para regular y convertir la cantidad de voltaje en el circuito.
2. Revisión manual para confirmar que los circuitos sean conectados de manera correcta
3. Cargamos con el IDE Arduino el código para que se envíen los valores a Firebase y este sea usado en la app.
4. Crear un esquema de cableado en la parte inferior de la maqueta para evitar que estén expuestos.

SOFTWARE (Algoritmo)

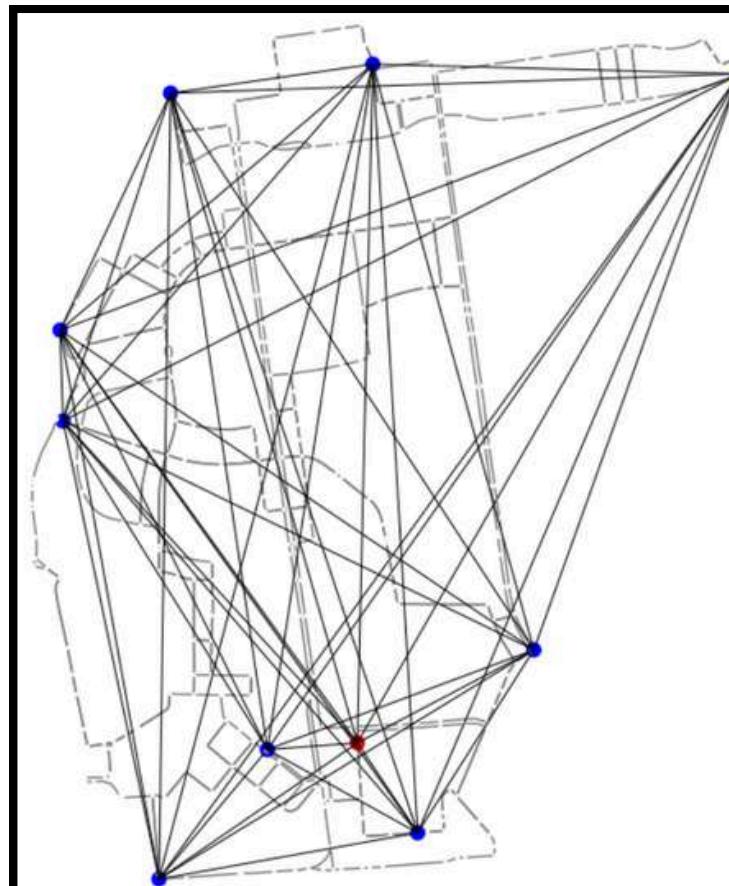
RETOS Y LIMITACIONES

1. La zona delimitada de San Borja cuenta con 1402 nodos y 3266 aristas (Grafo Completo).



SOLUCIONES

1. Reducir las cantidades de nodos y las aristas, considerando solo por donde haría el traslado (camino) o el recojo de residuos convirtiéndolo en un grafo filtrado.



SOFTWARE (Algoritmo)

RETOS Y LIMITACIONES

Código del (Grafo Completo).

```
1 import osmnx as ox
2 import networkx as nx
3 import matplotlib as plt
4
5 nombre_ciudad = "San Borja, Peru"
6 grafo_completo = ox.graph_from_place(nombre_ciudad, network_type = 'drive')
7 fig, ax = ox.plot_graph(grafo_completo)
```



SOLUCIONES

Código del Grafo Filtrado. con solo las rutas posibles que iria el camion compactador, estas rutas fueron obtenidas con el algoritmo de dijkstra

```
1 import osmnx as ox
2 import networkx as nx
3 import itertools
4
5 # Lista de nodos
6 nodos = [3628169514, 1641248425, 1047200484, 791160018, 412554058, 392153976, 392152310, 2329670857, 4549916939, 417252571]
7
8 # Crear pares de nodos (combinaciones)
9 pares = list(itertools.combinations(nodos, 2))
10
11 # Lista para almacenar las rutas
12 rutas = []
13
14 # Iterar sobre cada par de nodos y encontrar la ruta más corta usando el algoritmo de Dijkstra
15 for (inicio, fin) in pares:
16     try:
17         ruta = nx.shortest_path(grafo, source=inicio, target=fin, weight='length', method='dijkstra')
18         rutas.append(ruta)
19     except nx.NetworkXNoPath:
20         # Si no hay ruta entre los nodos, agregar una lista vacía
21         rutas.append([])
22
23 # Conjunto para almacenar todos los nodos sin duplicados
24 todos_nodos = set()
25
26 # Agregar todos los nodos de cada ruta al conjunto
27 for ruta in rutas:
28     todos_nodos.update(ruta)
29
30 # Convertir el conjunto a una lista
31 lista_nodos_unicos = list(todos_nodos)
32
33 # Mostrar la lista de nodos únicos
34 print("Lista de nodos únicos:", lista_nodos_unicos)
35
36 # Opcional: Guardar la lista de nodos únicos en un archivo de texto
37 with open('nodos_unicos.txt', 'w') as f:
38     f.write("Lista de nodos únicos:\n")
39     for nodo in lista_nodos_unicos:
40         f.write(f"{nodo}\n")
```

SOFTWARE (Algoritmo)

SOLUCIONES

Luego de obtener las rutas unicas, las usaremos para armar el grafo filtrado.

Datos Necesarios:

1. Distancia de punto a punto para usarlo en nuestra base de datos

```
1 import networkx as nx
2
3 nodos_calculo = [3628169514, 1641248425, 1047200484, 791160018, 412554058, 392153976, 392152310, 2329670857, 4549916939, 417252571]
4
5 # Calcular la distancia de nodo a nodo
6 def calcular_distancia_nodos(grafico, nodos):
7     distancias_nodo_a_nodo = {}
8     for i in range(len(nodos)):
9         for j in range(len(nodos)):
10            if i != j:
11                nodo_inicio = nodos[i]
12                nodo_destino = nodos[j]
13                try:
14                    # Calcular la ruta más corta usando el algoritmo de Dijkstra
15                    ruta_mas_corta = nx.shortest_path(grafico, source=nodo_inicio, target=nodo_destino, weight='length')
16                    # Calcular la longitud de la ruta encontrada
17                    longitud_total = sum(grafico[u][v][0]['length'] for u, v in zip(ruta_mas_corta[:-1], ruta_mas_corta[1:]))
18                    # Guardar la distancia nodo a nodo
19                    distancias_nodo_a_nodo[(nodo_inicio, nodo_destino)] = longitud_total
20                except nx.NetworkXNoPath:
21                    print(f"No hay camino entre {nodo_inicio} y {nodo_destino}.")
22                    distancias_nodo_a_nodo[(nodo_inicio, nodo_destino)] = float('inf') # Indicar que no hay camino
23    return distancias_nodo_a_nodo
24
25 # Recibe a grafo filtrado como grafo original
26 distancias_nodo_a_nodo = calcular_distancia_nodos(grafico_filtrado, nodos_calculo)
27
28 # Imprimir resultados
29 for nodo, distancia in distancias_nodo_a_nodo.items():
30     nodo_inicio, nodo_destino = nodo
31     print(f"Distancia entre nodo {nodo_inicio} y nodo {nodo_destino}: {distancia:.2f} metros."
```

Código te genera la distancia de punto a punto manualmente tenemos que darle los nombres a los nodos que resaltaremos

```
1 import heapq
2 capacidad_maxima = 6000
3 capacidades_nodos = {'X': 0, 'A': 30, 'B': 20, 'C': 40, 'D': 25, 'E': 35, 'F': 45, 'G': 15, 'H': 10, 'I': 5}
4
5 distancias_renombradas = {
6     'X': {'A': 4289.39, 'B': 2887.56, 'C': 3349.81, 'D': 2044.60, 'E': 2439.43, 'F': 873.80, 'G': 1388.11, 'H': 678.69, 'I': 547.35},
7     'A': {'X': 4033.17, 'B': 2417.92, 'C': 1586.21, 'D': 3467.37, 'E': 3284.98, 'F': 3182.96, 'G': 5159.82, 'H': 4437.55, 'I': 4080.41},
8     'B': {'X': 3225.05, 'A': 3178.66, 'C': 1654.33, 'D': 1978.83, 'E': 1793.74, 'F': 3033.58, 'G': 3729.18, 'H': 3006.91, 'I': 3352.13},
9     'C': {'X': 3332.11, 'A': 1524.33, 'B': 1458.13, 'D': 2537.72, 'E': 2352.63, 'F': 2481.91, 'G': 3565.80, 'H': 3379.36, 'I': 3379.36},
10    'D': {'X': 2449.06, 'A': 3685.67, 'B': 2035.39, 'C': 2558.68, 'E': 394.82, 'F': 2261.25, 'G': 2234.64, 'H': 1985.24, 'I': 2576.14},
11    'E': {'X': 3197.52, 'A': 3522.90, 'B': 1872.62, 'C': 2395.83, 'D': 1409.83, 'F': 3009.71, 'G': 3418.46, 'H': 2818.92, 'I': 3324.60},
12    'F': {'X': 882.09, 'A': 3415.59, 'B': 3097.73, 'C': 2476.01, 'D': 2257.29, 'E': 2652.11, 'G': 2151.32, 'H': 1560.78, 'I': 929.33},
13    'G': {'X': 1572.22, 'A': 5861.61, 'B': 4459.78, 'C': 4922.03, 'D': 3616.82, 'E': 4011.65, 'F': 2446.02, 'H': 920.03, 'I': 1699.30},
14    'H': {'X': 684.18, 'A': 4973.57, 'B': 3571.74, 'C': 4033.99, 'D': 2728.79, 'E': 3123.61, 'F': 1557.98, 'G': 1188.32, 'I': 811.26},
15    'I': {'X': 547.35, 'A': 4344.92, 'B': 3434.91, 'C': 3405.34, 'D': 2591.96, 'E': 2986.78, 'F': 929.33, 'G': 1515.19, 'H': 805.77}
16
17 max_capacidad = max(capacidades_nodos.values())
18 min_capacidad = min(capacidades_nodos.values())
19 max_distancia = max(max(d.values()) for d in distancias_renombradas.values())
20 min_distancia = min(min(d.values()) for d in distancias_renombradas.values())
21
22 def normalizar(valor, max_valor, min_valor):
23     return (valor - min_valor) / (max_valor - min_valor)
24
25 def seleccionar_siguiente_nodo(nodo_actual, distancias, capacidades, visitados, capacidad_restante):
26     max_heap = []
27
28     for nodo_destino, distancia in distancias[nodo_actual].items():
29         if nodo_destino not in visitados:
30             distancia_normalizada = normalizar(distancia, max_distancia, min_distancia)
31             capacidad_normalizada = normalizar(capacidades[nodo_destino], max_capacidad, min_capacidad)
32             prioridad = (0.35 * distancia_normalizada + 0.65 * capacidad_normalizada)
33             if capacidades[nodo_destino] <= capacidad_restante:
34                 heapq.heappush(max_heap, (-prioridad, nodo_destino, distancia))
35
36     if max_heap:
37         mejor_opcion = heapq.heappop(max_heap)
38         return mejor_opcion[1], capacidades[mejor_opcion[1]], mejor_opcion[2]
39     return None, 0, 0
40
41 def recorrido_priorizado(nodo_inicial, distancias, capacidades, capacidad_maxima):
42     nodo_actual = nodo_inicial
43     visitados = set([nodo_actual])
44     recorrido = [nodo_actual]
45     capacidad_restante = capacidad_maxima - capacidades[nodo_actual]
46     distancia_total = 0
47
48     while True:
49         siguiente_nodo, capacidad_nodo, distancia = seleccionar_siguiente_nodo(nodo_actual, distancias, capacidades, capacidad_restante)
50         if siguiente_nodo is None:
51             break
52         visitados.add(siguiente_nodo)
53         recorrido.append(siguiente_nodo)
54         capacidad_restante -= capacidad_nodo
55         distancia_total += distancia
56         nodo_actual = siguiente_nodo
57         if capacidad_restante <= 0:
58             break
59     if nodo_actual != nodo_inicial:
60         distancia_de_Regreso = distancias[nodo_actual][nodo_inicial]
61         distancia_total += distancia_de_Regreso
62         recorrido.append(nodo_inicial)
63
64     return recorrido, distancia_total, capacidad_restante
```

Código final que recibe todos los datos y obtienes una respuesta

SOFTWARE (Algoritmo)

RETOS Y LIMITACIONES

1. Determinar la función costo (representación matemática) para calcular la prioridad dentro del algoritmo y que sea escalable en el futuro.

Distribución de los pesos de prioridad para la distancia mínima a recorrer y capacidad de los tachos.

Variables Utilizadas en el Algoritmo

1. C : Conjunto de contenedores de basura.
2. $d(i, j)$: Distancia entre el contenedor i y el contenedor j .
3. $l(i)$ Nivel de llenado del contenedor i .
4. p Contenedor de partida (ubicación inicial del camión recolector).
5. Q Capacidad del camión recolector.
6. q Carga actual del camión.
7. w_d Peso asignado a la distancia.
8. w_l Peso asignado al nivel de llenado.

Función de Costo

La función de costo $C(i)$ para cada contenedor i se define como:

$$C(i) = \frac{1}{w_d \times d(i,j) + w_l \times l(i)}$$

La función de costo T por toda la ruta se define como:

$$T = \sum \frac{1}{d(i,j) + l(i)}$$

SOLUCIONES

1. Establecemos y decidimos que 0.8 a la capacidad de los tachos y 0.2 a la distancia por el impacto ambiental que representan los tachos.

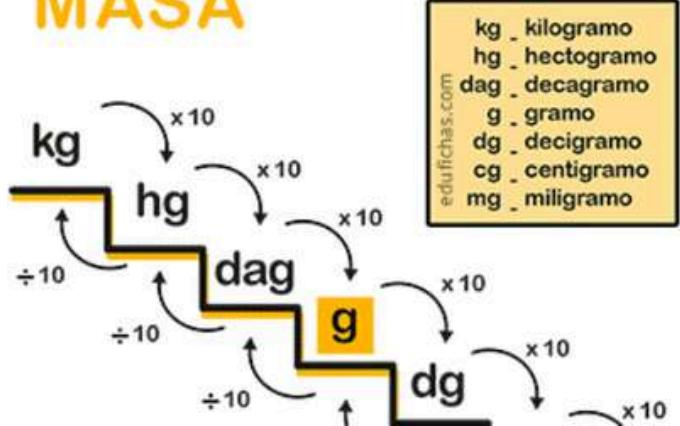
```
prioridad = (0.35 * distancia_normalizada + 0.65 * capacidad_normalizada)
```

SOFTWARE (Algoritmo)

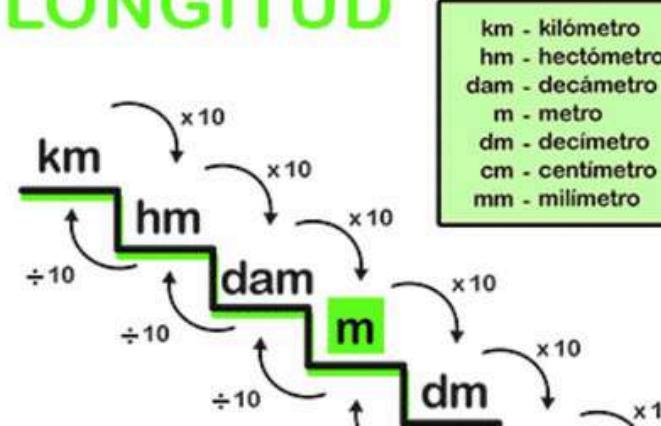
RETOS Y LIMITACIONES

1. Establecer en el mismo tipo la distancia en Km y la capacidad de los tachos en Kg.
2. Tener que iniciar nuevamente sesión en el aplicativo para actualizar la información en tiempo real.

MASA



LONGITUD



SOLUCIONES

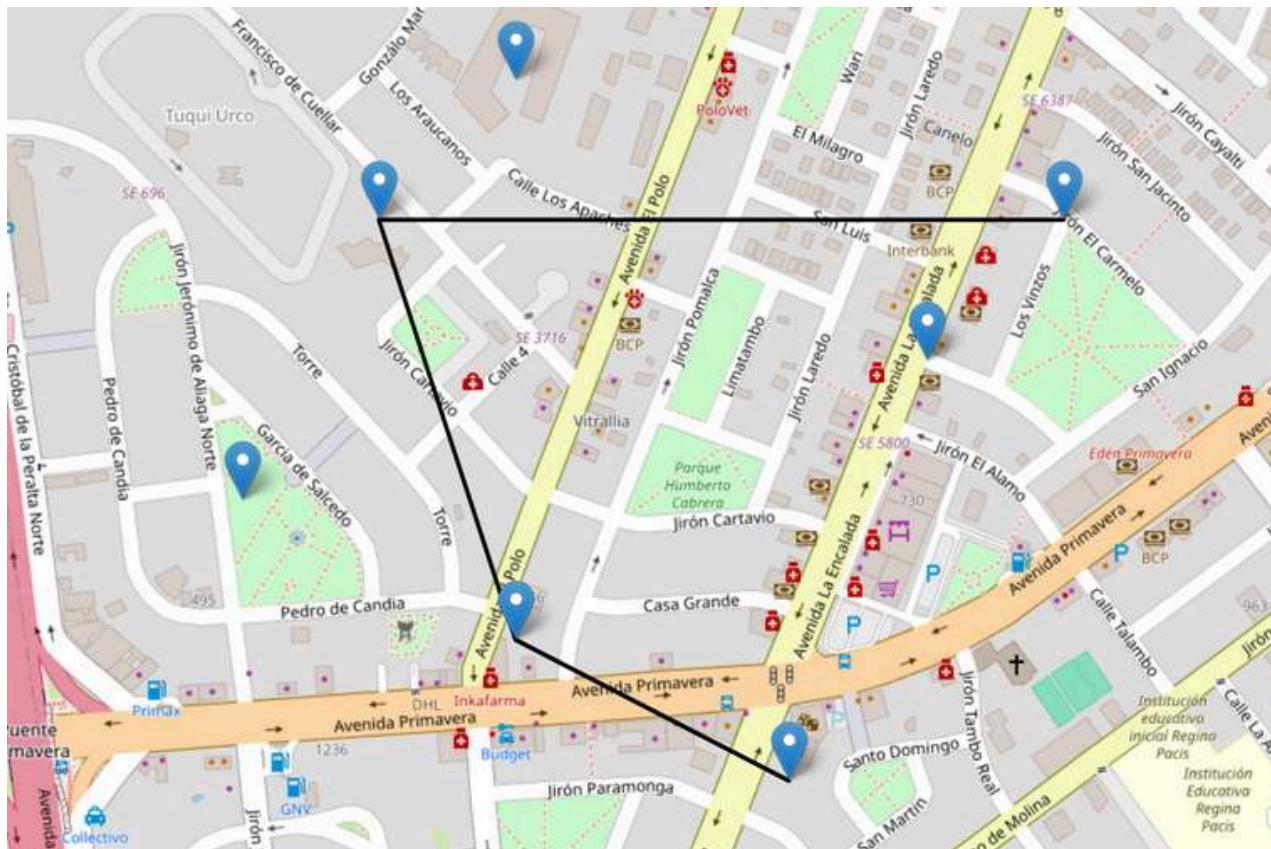
1. Empleo de la técnica de normalización MinMax, realizando el cálculo dentro del algoritmo.

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

SOFTWARE (Algoritmo)

RETOS Y LIMITACIONES

No traza correctamente la ruta óptima(resultado del algoritmo) por caminos o calles reales en la visualización dentro del mapa de OpenStreetMap.



SOLUCIONES

Emplear la API de Google Maps o continuar con OpenStreetMap pero asignandole solo los caminos correctos por los que debe trazar la ruta.

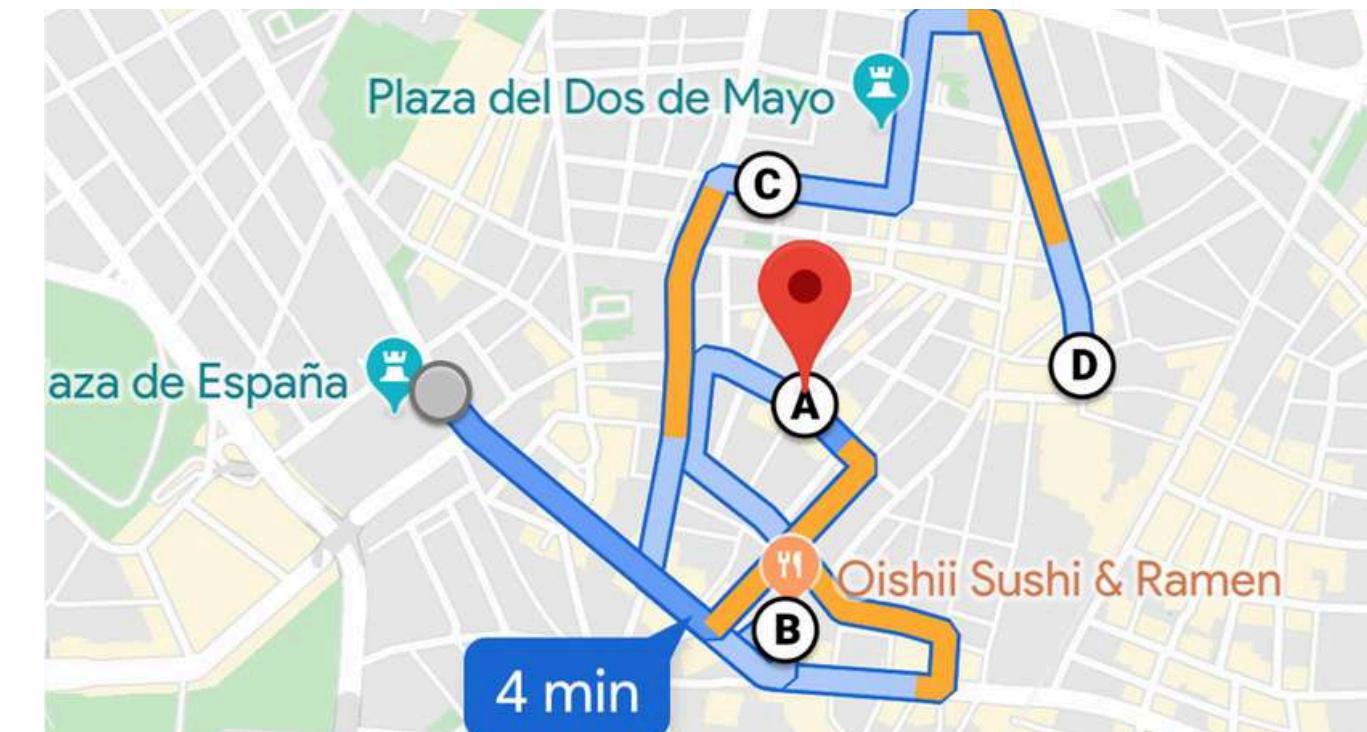


Imagen Referencial de Google Maps

MANUFACTURA DIGITAL

RETOS Y LIMITACIONES

- Representación del funcionamiento del software en una maqueta a escala
- Elección del mapa de nodos en la maqueta
- Introducción de los potenciómetros
- Escalar las distancias de los nodos en unidades métricas (m o km)
- Observación en tiempo real de las modificaciones en los valores del potenciómetro en la app
- Distribución de los componentes físicos en el proyecto según el tamaño de los materiales
- El largo de las conexiones respecto a la cantidad de alambre presupuestado
- Representación visual del cambio de la ruta óptima para el usuario

SOLUCIONES

- Implementar una leyenda de mapa que permita al usuario visualizar mejor las distancias a través de una referencia de escala
- Realizar una interfaz en la aplicación que le muestre al usuario las actualizaciones recientes de los valores del potenciómetro como de las rutas
- Emplear una gestión y orden en el cableado que reduzca el desperdicio y optimice el uso de nuestros materiales.