

Módulo 5: Personalización de Mapas y Estilos

Instructor: Eduardo Barron

Agenda del Módulo 5

- Conceptos de estilo de mapa
- Estilos básicos: claro/oscuro, contraste y legibilidad
- Estilos temáticos por caso de uso (retail, movilidad, logística)
- Capas visuales y controles (tráfico/transporte/terreno)
- Buenas prácticas, rendimiento y límites
- Ejemplos sugeridos (JS y Flutter)
- Cierre y preguntas

**Conceptos de
estilo: ¿qué se
puede
personalizar?**

Colores y visibilidad de
elementos: agua, parques,
carreteras, edificios.

Etiquetas y puntos de
interés (densidad, visibilidad
por zoom).

Controles y gestos del mapa
(zoom, tipo de mapa,
interacción).

Los estilos se definen como arrays JSON con reglas de `featureType`, `elementType` y `stylers`.

JavaScript (Google Maps JS API)

<> html

```
<script async defer src="https://maps.googleapis.com/maps/api/js?key=TU_API_KEY&libraries=maps&language=es-419"></script>
```

js javascript

// Aplicar un estilo JSON personalizado al mapa

```
const customStyle = [
  {
    featureType: "all",
    elementType: "geometry.fill",
    stylers: [{ color: "#f5f5f5" }]
  },
  {
    featureType: "water",
    elementType: "geometry",
    stylers: [{ color: "#c9c9c9" }]
  }
];
```

// Inicializar mapa con estilo personalizado

```
function initMap() {
  const map = new google.maps.Map(document.getElementById("map"), {
    center: { lat: -34.397, lng: 150.644 },
    zoom: 8,
    styles: customStyle // Aplicar el estilo JSON
  });
}
```

En Flutter, el estilo se pasa como string JSON al método `setMapStyle()`.

Flutter (google_maps_flutter)

```
dart

// Aplicar estilo personalizado usando MapStyleOptions
class MapStyleController {
  final GoogleMapController? _controller;

  MapStyleController(this._controller);

  // Estilo JSON como string
  static const String customStyle = '''
[
  {
    "featureType": "all",
    "elementType": "geometry.fill",
    "stylers": [{"color": "#f5f5f5"}]
  },
  {
    "featureType": "water",
    "elementType": "geometry",
    "stylers": [{"color": "#c9c9c9"}]
  }
]
''';

  Future<void> applyCustomStyle() async {
    await _controller?.setMapStyle(customStyle);
  }
}
```

Conceptos Clave

featureType: Tipo de elemento del mapa (all, water, road, poi, etc.)

elementType: Parte específica del elemento (geometry, labels, icons, etc.)

stylers: Array de propiedades de estilo (color, visibility, weight, etc.)

Aplicación: Los estilos se aplican al mapa usando `map.setOptions({styles: style})` en JS o `controller.setMapStyle(style)` en Flutter

Estilos base: claro/oscuro y alto contraste

01

MODO CLARO:
MEJOR PARA
AMBIENTES
LUMINOSOS Y
LEGIBILIDAD DE
DATOS.

02

MODO OSCURO:
IDEAL PARA UIS
NOCTURNAS, REDUCE
FATIGA VISUAL.

03

ALTO CONTRASTE:
RESALTAR
RUTAS/DATOS SOBRE
EL MAPA
(ACCESIBILIDAD).

JS javascript

// Definir estilos para modo claro y oscuro

```
const lightStyle = [
  {
    featureType: "all",
    elementType: "geometry.fill",
    stylers: [{ color: "#ffffff" }]
  },
  {
    featureType: "water",
    elementType: "geometry",
    stylers: [{ color: "#c9c9c9" }]
  }
];
```


```
const darkStyle = [
  {
    featureType: "all",
    elementType: "geometry.fill",
    stylers: [{ color: "#212121" }]
  },
  {
    featureType: "water",
    elementType: "geometry",
    stylers: [{ color: "#000000" }]
  }
];
```

// Función para alternar entre temas

```
function toggleMapTheme(isDark) {
  map.setOptions({
    styles: isDark ? darkStyle : lightStyle
  });
}
```

// Toggle con botón

```
document.getElementById('themeToggle').addEventListener('click', function() {
  const isDark = this.checked;
  toggleMapTheme(isDark);
});
```


 dart



```
// Alternar entre estilos claro y oscuro
class ThemeMapController {
  final GoogleMapController? _controller;
  bool _isDarkMode = false;

  ThemeMapController(this._controller);

  static const String lightStyle = '''
[{"featureType": "all", "elementType": "geometry.fill", "stylers": [{"color": "#ffffff"}]}]
''';

  static const String darkStyle = '''
[{"featureType": "all", "elementType": "geometry.fill", "stylers": [{"color": "#212121"}]}]
''';

  Future<void> toggleTheme() async {
    _isDarkMode = !_isDarkMode;
    final style = _isDarkMode ? darkStyle : lightStyle;
    await _controller?.setMapStyle(style);
  }

  // Usar con Theme.of(context).brightness
  Future<void> applyThemeBasedOnSystem(Brightness brightness) async {
    final style = brightness == Brightness.dark ? darkStyle : lightStyle;
    await _controller?.setMapStyle(style);
  }
}
```

Conceptos Clave

Modo Claro: Colores claros de fondo con texto oscuro para mejor legibilidad diurna

Modo Oscuro: Colores oscuros de fondo con texto claro para mejor experiencia nocturna

Alto Contraste: Colores muy contrastantes para mejorar accesibilidad

Integración con Sistema: En Flutter se puede integrar con `Theme.of(context).brightness` para detección automática

Toggle Dinámico: Cambio de tema en tiempo real sin recargar la página

Estilos temáticos por caso de uso

- Retail: destacar áreas comerciales y POIs relevantes.

- Movilidad: enfatizar red vial, tráfico y restricciones viales.

- Logística: zonas de cobertura, bodegas, estaciones de carga.

// Estilo temático para retail – resaltar comercios y ocultar POIs irrelevantes

```
const retailStyle = [
  {
    featureType: "poi.business",
    elementType: "labels",
    stylers: [{ visibility: "on" }]
  },
  {
    featureType: "poi.attraction",
    elementType: "labels",
    stylers: [{ visibility: "off" }]
  },
  {
    featureType: "road.highway",
    elementType: "geometry",
    stylers: [{ color: "#ff6b6b" }]
  },
  {
    featureType: "transit.station",
    elementType: "labels",
    stylers: [{ visibility: "off" }]
  }
];
```

// Estilo para logística – enfocar en vías principales

```
const logisticsStyle = [
  {
    featureType: "road.highway",
    elementType: "geometry",
    stylers: [{ color: "#2ecc71" }, { weight: 2.0 }]
  },
  {
    featureType: "poi",
    elementType: "labels",
    stylers: [{ visibility: "off" }]
  }
];
```

// Aplicar estilo según caso de uso

```
function applyThematicStyle(useCase) {
  const style = useCase === 'retail' ? retailStyle : logisticsStyle;
  map.setOptions({ styles: style });
}
```

```
// Estilos temáticos para diferentes casos de uso
class ThematicMapStyles {
  // Estilo para retail - resaltar comercios
  static const String retailStyle = '''
  [
    {
      "featureType": "poi.business",
      "elementType": "labels",
      "stylers": [{"visibility": "on"}]
    },
    {
      "featureType": "poi.attraction",
      "elementType": "labels",
      "stylers": [{"visibility": "off"}]
    }
  ]
  ''';

  // Estilo para logística - enfocar en vías
  static const String logisticsStyle = '''
  [
    {
      "featureType": "road.highway",
      "elementType": "geometry",
      "stylers": [{"color": "#2ecc71"}, {"weight": 2.0}]
    }
  ]
  ''';

  static Future<void> applyStyle(GoogleMapController controller, String useCase) async {
    final style = useCase == 'retail' ? retailStyle : logisticsStyle;
    await controller.setMapStyle(style);
  }
}
```

Conceptos Clave

Los estilos temáticos mejoran la experiencia según el contexto de uso.

Considera la legibilidad de etiquetas al ocultar/mostrar POIs.

Ocultación Selectiva: Usar visibility: "off" para ocultar POIs no relevantes

Resaltado: Usar colores llamativos y visibility: "on" para elementos importantes

Contexto de Uso: Cada estilo está optimizado para un caso de uso específico

Capas y controles visuales

- Capas: tráfico en tiempo real, transporte público, terreno/satélite.

- Controles: tipo de mapa, fullscreen, zoom, mi ubicación.

- Recomendación: activar solo lo necesario para no saturar la UI.


JS javascript

```
// Activar capas y controles visuales
function initializeMapWithLayers() {
  const map = new google.maps.Map(document.getElementById("map"), {
    center: { lat: -34.397, lng: 150.644 },
    zoom: 10,
    // Controles del mapa
    mapTypeControl: true,
    fullscreenControl: true,
    zoomControl: true,
    streetViewControl: false,
    scaleControl: true
  });

  // Activar capa de tráfico
  const trafficLayer = new google.maps.TrafficLayer();
  trafficLayer.setMap(map);

  // Activar capa de transporte público
  const transitLayer = new google.maps.TransitLayer();
  transitLayer.setMap(map);

  // Controlar visibilidad de capas
  document.getElementById('toggleTraffic').addEventListener('click', function() {
    trafficLayer.setMap(trafficLayer.getMap() ? null : map);
  });
}
```


 dart



```
// Configurar capas y controles en Flutter
class MapLayersController {
  final GoogleMapController? _controller;

  MapLayersController(this._controller);

  // Configuración del mapa con controles
  static const GoogleMapOptions mapOptions = GoogleMapOptions(
    mapType: MapType.normal,
    myLocationEnabled: true,
    myLocationButtonEnabled: true,
    zoomControlsEnabled: true,
    compassEnabled: true,
    mapToolbarEnabled: true,
  );

  // Activar/desactivar ubicación del usuario
  Future<void> toggleMyLocation(bool enabled) async {
    // Nota: En Flutter, esto se controla desde el widget GoogleMap
    // myLocationEnabled: enabled
  }

  // Cambiar tipo de mapa
  Future<void> setMapType(MapType mapType) async {
    // Se controla desde el widget GoogleMap con mapType: mapType
  }
}
```

Conceptos Clave

Capas de Información: Tráfico, transporte público, ciclovías

Controles del Mapa: Selector de tipo, pantalla completa, zoom, Street View, escala

Mi Ubicación: Mostrar y centrar en la ubicación del usuario

Gestión de Visibilidad: Activar/desactivar capas dinámicamente

Costo Adicional: Las capas de tráfico y transporte requieren facturación adicional

Limitaciones en Flutter: Algunas capas no están disponibles directamente como en JavaScript

Buenas prácticas, rendimiento y límites

- Reducir estilos excesivos; probar legibilidad en diferentes zooms.

- Cargar estilos desde JSON (JS) o recursos (Flutter) para reutilización.

- Considerar cuotas y costos de capas (tráfico/places) y sobrecarga visual.

- Verificar compatibilidad y cambios de API con el tiempo.

JS javascript

```
// Buenas prácticas para rendimiento y gestión de estilos
class OptimizedMapManager {
  constructor() {
    this.map = null;
    this.styles = new Map(); // Cache de estilos
    this.isAnimating = false;
  }

  // Cargar estilos desde archivo JSON
  async loadStyleFromFile(stylePath) {
    if (this.styles.has(stylePath)) {
      return this.styles.get(stylePath);
    }

    try {
      const response = await fetch(stylePath);
      const style = await response.json();
      this.styles.set(stylePath, style);
      return style;
    } catch (error) {
      console.error('Error cargando estilo:', error);
      return [];
    }
  }

  // Aplicar estilo optimizado
  async applyStyle(stylePath) {
    const style = await this.loadStyleFromFile(stylePath);

    // Desactivar animaciones para mejor rendimiento
    this.map.setOptions({
      styles: style,
      gestureHandling: 'cooperative', // Requiere scroll + zoom
      disableDefaultUI: false,
      zoomControl: true
    });
  }

  // Limpiar recursos
  destroy() {
    this.styles.clear();
    this.map = null;
  }
}
```

```
// Buenas prácticas para Flutter
class OptimizedMapWidget extends StatefulWidget {
  @override
  _OptimizedMapWidgetState createState() => _OptimizedMapWidgetState();
}

class _OptimizedMapWidgetState extends State<OptimizedMapWidget> {
  GoogleMapController? _controller;
  String? _currentStyle;

  // Pre-cargar estilos desde assets
  Future<String> _loadStyleFromAssets(String assetPath) async {
    return await rootBundle.loadString(assetPath);
  }

  // Aplicar estilo sin rebuilds innecesarios
  Future<void> _applyStyle(String styleJson) async {
    if (_currentStyle == styleJson) return; // Evitar aplicaciones duplicadas

    await _controller?.setMapStyle(styleJson);
    _currentStyle = styleJson;
  }

  // Controlar markers para mejor rendimiento
  Set<Marker> _markers = {};

  void _addMarker(LatLng position) {
    setState(() {
      _markers.add(Marker(
        markerId: MarkerId(position.toString()),
        position: position,
      ));
    });
  }

  @override
  Widget build(BuildContext context) {
    return GoogleMap(
      onMapCreated: (GoogleMapController controller) {
        _controller = controller;
      },
      markers: _markers,
      // Configuración optimizada
      mapType: MapType.normal,
      myLocationEnabled: true,
      zoomControlsEnabled: true,
    );
  }
}
```

Conceptos Clave

Cache de Estilos: Almacenar estilos en memoria para evitar cargas repetidas

Pre-carga: Cargar estilos comunes al inicio de la aplicación

Gestión de Memoria: Limpiar recursos cuando no se necesiten

Límites de API: 50,000 requests/día para estilos personalizados

Optimización de Markers: Implementar clustering para muchos markers

Monitoreo de Uso: Rastrear el uso de API para evitar límites

Fallbacks: Implementar estilos por defecto cuando se alcancen los límites

Rendimiento: Desactivar animaciones innecesarias y optimizar configuraciones



Ejemplos propuestos (JS y Flutter)



Tema claro/oscuro con switch +
estilo JSON (toggle en vivo).



Mapa minimalista para dashboards
(ocultar POIs y etiquetas
específicas).



Tema movilidad: resaltar tránsito,
rutas y límites de velocidad.



Tema retail: enfatizar zonas
comerciales y horarios de apertura.



Tema logística: zonas de cobertura y
puntos de carga.

Cierre y preguntas



- Personalizar ≠ saturar: prioriza legibilidad y casos de uso.



- Centraliza estilos y documenta cambios.



- ¿Qué estilo aplicaría mejor en tu proyecto?