Google Maps API Integration Guide

Table of Contents

Google Maps API Overview

What is Google Maps JavaScript API?

The **Google Maps JavaScript API** is a powerful service provided by Google that allows developers to embed interactive, customizable maps into web applications. It provides a rich set of features for displaying geographic data, adding markers, drawing shapes, calculating routes, and much more.

Key Features

1. **Interactive Maps**: Pan, zoom, rotate, and tilt maps
2. **Markers**: Place custom markers at specific locations
3. **Info Windows**: Display information popups
4. **Overlays**: Add shapes, polylines, polygons, and images
5. **Street View**: Integrate Google Street View
6. **Directions**: Calculate and display routes
7. **Geocoding**: Convert addresses to coordinates and vice versa
8. **Places**: Search for businesses and points of interest
9. **Drawing Tools**: Let users draw on the map
10. **Heatmaps**: Visualize data density

Why Use Google Maps API?

Advantages:

- **Familiar Interface**: Users already know how to interact with Google Maps
- **Rich Features**: Extensive functionality out of the box
- **Accurate Data**: Google's comprehensive mapping data
- **Mobile Support**: Works seamlessly on mobile devices
- **Regular Updates**: Google continuously improves the API
- **Integration**: Easy to combine with other Google services

Common Use Cases:

- Store locators
- Delivery tracking
- Real estate listings
- Event location displays
- Travel planning applications
- Fleet management systems
- Social location sharing
- Geographic data visualization

Setting Up Google Maps

Step 1: Get an API Key

Before using Google Maps API, you need an API key from Google Cloud Platform.

Process:

1. Go to Google Cloud Console
2. Create a new project (or select existing one)
3. Enable "Maps JavaScript API"
4. Go to Credentials section
5. Create credentials → API Key
6. (Optional) Restrict your API key for security

API Key Restrictions (Recommended for Production):

- **Application restrictions**: Limit to specific websites
- **API restrictions**: Limit to specific Google APIs
- **Usage quotas**: Set daily usage limits

Important Security Note:
The API key in your code (`AIzaSyB41DRUbKWJHPxaFjMAwdrzWzbVKartNGg`) is exposed in the HTML. For production applications:

- Restrict the key to your domain
- Monitor usage regularly
- Never commit API keys to public repositories
- Consider using environment variables or backend proxy

Step 2: Include Google Maps Script

Add the Google Maps JavaScript API to your HTML file using a script tag.

Basic Syntax:

```
<script
  src="https://maps.googleapis.com/maps/api/js?
key=YOUR_API_KEY&callback=initMap&v=weekly"
  defer>
</script>
```

URL Parameters Explained:

1. **key=YOUR_API_KEY**
   - Your unique API key from Google Cloud Console
   - Required for authentication and billing
   - Example: `key=AIzaSyB41DRUbKWJHPxaFjMAwdrzWzbVKartNGg`
2. **callback=initMap**
   - Function to call when API loads successfully
   - Must be a globally accessible function
   - Example: `window.initMap = initMap;`
3. **v=weekly** (Version)
   - Specifies which version of the API to load
   - Options:
     - `v=weekly` : Latest weekly release (recommended for development)
     - `v=quarterly` : Latest quarterly release (more stable)
     - `v=3.45` : Specific version number (maximum stability)
   - Using `weekly` ensures you get latest features but may have breaking changes
4. **libraries** (Optional, not in your code)
   - Load additional Google Maps libraries

- Example: `libraries=places,drawing,geometry`
- Available libraries:
  - `places` : Places API (search, autocomplete)
  - `drawing` : Drawing tools
  - `geometry` : Geometric calculations
  - `visualization` : Heatmaps and data visualization
  - `marker` : Advanced marker customization

Complete Example with All Parameters:

```
<script
  src="https://maps.googleapis.com/maps/api/js?
key=YOUR_KEY&callback=initMap&v=weekly&libraries=places,drawing"
  defer>
</script>
```

Step 3: Prepare the HTML Structure

Create a container element where the map will be displayed.

```
<!DOCTYPE html>
<html>
<head>
    <title>Simple Markers</title>
    <script src="map.js" defer></script>

    <style>
        /* Map container MUST have explicit height */
        #map {
            height: 100%;
        }

        /* Make page fill window */
        html, body {
            height: 100%;
            margin: 0;
            padding: 0;
        }
    </style>
</head>
<body>
    <div id="map"></div>
```

```
    <script
        src="https://maps.googleapis.com/maps/api/js?
key=YOUR_API_KEY&callback=initMap&v=weekly"
        defer>
    </script>
</body>
</html>
```

Critical CSS Requirements:

The map container **MUST** have an explicit height defined. Without it, the map won't display.

```
/* REQUIRED - Map container height */
#map {
    height: 100%;  /* Or specific value like 500px */
}

/* Make container fill entire viewport */
html, body {
    height: 100%;
    margin: 0;
    padding: 0;
}
```

Why Height is Critical:

- Google Maps calculates tile positions based on container dimensions
- Without explicit height, container has 0px height
- Map renders but is invisible (0px tall)
- Common mistake for beginners

Alternative Height Specifications:

```
/* Fixed height */
#map {
    height: 600px;
}

/* Viewport height */
#map {
    height: 100vh;
}

/* Calculated height */
```

```css
#map {
    height: calc(100vh - 60px);  /* Full height minus header */
}

/* Percentage with parent having height */
.container {
    height: 800px;
}
#map {
    height: 50%;  /* 400px */
}
```

Step 4: defer Attribute

What is `defer`?

```html
<script src="map.js" defer></script>
```

The `defer` attribute tells the browser to:

1. Download the script in parallel with HTML parsing (non-blocking)
2. Execute the script only after HTML is fully parsed
3. Maintain script execution order

defer vs async vs Normal:

Attribute	Download	Execution	Use Case
Normal	Blocking	Immediately	Legacy code
async	Non-blocking	As soon as downloaded	Independent scripts
defer	Non-blocking	After HTML parsed	Scripts needing DOM

Why Use defer for Google Maps:

```html
<script src="map.js" defer></script>
<script src="https://maps.googleapis.com/...&callback=initMap" defer></script>
```

1. Both scripts download in parallel (fast page load)
2. Scripts execute after DOM is ready (`#map` element exists)
3. Execution order is maintained (map.js before Google Maps API)
4. Prevents "element not found" errors

Without defer (Problems):

```
<!-- BAD: Blocks HTML parsing -->
<script src="map.js"></script>

<!-- BAD: May execute before DOM ready -->
<script async src="map.js"></script>
```

Map Initialization

Basic Map Creation

The `initMap()` function creates and displays the map.

Basic Structure:

```
function initMap(latitude, longitude) {
    // 1. Define center position
    const myLatLng = { lat: latitude, lng: longitude };

    // 2. Create map instance
    const map = new google.maps.Map(document.getElementById("map"), {
        zoom: 15,
        center: myLatLng,
    });

    // 3. Add marker
    new google.maps.Marker({
        position: myLatLng,
        map: map,
        title: "Hello World!",
    });
}

// 4. Make function globally accessible
window.initMap = initMap;
```

Understanding Each Component

1. Latitude and Longitude Object

```
const myLatLng = { lat: myLat, lng: myLon };
```

Latitude and Longitude Explained:

- **Latitude**: North-South position
  - Range: -90° (South Pole) to +90° (North Pole)
  - 0° = Equator
  - Example: Cairo, Egypt = 30.0444° N
- **Longitude**: East-West position
  - Range: -180° (West) to +180° (East)
  - 0° = Prime Meridian (Greenwich, London)
  - Example: Cairo, Egypt = 31.2357° E

Object Format:

```
// Google Maps format (required)
{ lat: 40.7128, lng: -74.0060 }  // New York City

// Alternative formats (also work)
new google.maps.LatLng(40.7128, -74.0060)

// Array format (for some methods)
[40.7128, -74.0060]
```

Important Notes:

- Property names must be `lat` and `lng` (not `latitude` / `longitude`)
- Values are numbers, not strings
- Positive latitude = North, negative = South
- Positive longitude = East, negative = West

2. Creating the Map Object

```
const map = new google.maps.Map(container, options);
```

Constructor Parameters:

Parameter 1 - container: DOM element or element ID

```
// Using element directly
const mapDiv = document.getElementById("map");
```

```
const map = new google.maps.Map(mapDiv, options);

// Using ID (shorthand)
const map = new google.maps.Map(document.getElementById("map"), options);
```

Parameter 2 - options: Configuration object

Essential Options:

```
{
    center: { lat: 40.7128, lng: -74.0060 },  // Required: Initial center
    zoom: 15                                   // Required: Zoom level
}
```

3. Zoom Levels

The `zoom` property controls how close the map is to the earth's surface.

Zoom Level Scale:

```
0  = Entire world
1  = Continents
5  = Large countries/states
10 = City
15 = Streets (your code uses this)
20 = Buildings
21 = Maximum zoom (varies by location)
```

Visual Examples:

```
zoom: 1   // See entire continents
zoom: 5   // See country
zoom: 10  // See city
zoom: 15  // See neighborhood/streets
zoom: 20  // See buildings/parcels
```

Zoom Best Practices:

Use Case	Recommended Zoom
World map	1-3
Country view	4-6
City view	10-12

Use Case	Recommended Zoom
Neighborhood	13-15
Street level	16-18
Building details	19-21

Dynamic Zoom:

```
// Set zoom programmatically
map.setZoom(12);

// Get current zoom
const currentZoom = map.getZoom();

// Zoom in/out
map.setZoom(map.getZoom() + 1);  // Zoom in
map.setZoom(map.getZoom() - 1);  // Zoom out
```

4. Making initMap Globally Accessible

```
window.initMap = initMap;
```

Why This is Necessary:

The Google Maps API script calls the callback function specified in the URL:

```
<script src="...&callback=initMap" defer></script>
```

When the API loads, it looks for `window.initMap` and calls it. Without this line:

```
// Without window.initMap (DOESN'T WORK)
function initMap() {
    // This is in module scope or local scope
    // Google Maps can't find it
}

// With window.initMap (WORKS)
function initMap() {
    // Code here
}
window.initMap = initMap;  // Makes it globally accessible
```

Alternative Approaches:

Method 1: Direct window property

```
window.initMap = function(myLat, myLon) {
    // Map initialization code
};
```

Method 2: Arrow function

```
window.initMap = (myLat, myLon) => {
    const myLatLng = { lat: myLat, lng: myLon };
    // Rest of code
};
```

Method 3: Traditional (no modules)

```
// If not using modules, function is automatically global
function initMap(myLat, myLon) {
    // No need for window.initMap assignment
}
```

Geolocation API Integration

Your code combines HTML5 Geolocation API with Google Maps to display the user's current location.

Complete Flow Explanation

```
navigator.geolocation.getCurrentPosition(
    function(pos) {
        // Success callback
        initMap(pos.coords.latitude, pos.coords.longitude);
    },
    function(e) {
        // Error callback
        // Currently empty - should handle errors!
    }
);
```

Step-by-Step Process

Step 1: Request User's Location

```
navigator.geolocation.getCurrentPosition(successCallback, errorCallback);
```

When this code runs:

1. Browser prompts user for location permission
2. User clicks "Allow" or "Block"
3. If allowed: Success callback executes
4. If blocked or error: Error callback executes

Step 2: Success Callback

```
function(pos) {
    initMap(pos.coords.latitude, pos.coords.longitude);
}
```

The `pos` (position) object contains:

```
{
    coords: {
        latitude: 40.7128,      // User's latitude
        longitude: -74.0060,    // User's longitude
        accuracy: 20,           // Accuracy in meters
        altitude: null,         // Height (meters, may be null)
        altitudeAccuracy: null, // Altitude accuracy
        heading: null,          // Direction of travel (degrees)
        speed: null             // Speed (meters/second)
    },
    timestamp: 1641234567890    // When position was acquired
}
```

Accessing Coordinates:

```
const lat = pos.coords.latitude;
const lng = pos.coords.longitude;
const accuracy = pos.coords.accuracy;  // How accurate (in meters)

console.log(`Location: ${lat}, ${lng}`);
console.log(`Accuracy: ±${accuracy} meters`);
```

Step 3: Initialize Map with User's Location

```
initMap(pos.coords.latitude, pos.coords.longitude);
```

This calls the `initMap` function with the user's actual coordinates, centering the map on their location.

Improving Error Handling

Your current error callback is empty. Here's a robust implementation:

```
navigator.geolocation.getCurrentPosition(
    function(pos) {
        // Success: Show map at user's location
        initMap(pos.coords.latitude, pos.coords.longitude);
    },
    function(error) {
        // Error: Handle gracefully
        console.error('Geolocation error:', error);

        switch(error.code) {
            case error.PERMISSION_DENIED:
                alert("Location access denied. Showing default location.");
                // Show default location (e.g., city center)
                initMap(40.7128, -74.0060);  // New York
                break;

            case error.POSITION_UNAVAILABLE:
                alert("Location information unavailable.");
                initMap(40.7128, -74.0060);
                break;

            case error.TIMEOUT:
                alert("Location request timed out.");
                initMap(40.7128, -74.0060);
                break;

            default:
                alert("An unknown error occurred.");
                initMap(40.7128, -74.0060);
                break;
        }
    },
    // Optional: Configuration options
    {
```

```
        enableHighAccuracy: true,  // Use GPS if available
        timeout: 5000,             // Max wait time: 5 seconds
        maximumAge: 0              // Don't use cached position
    }
);
```

Geolocation Options (Third Parameter)

```
navigator.geolocation.getCurrentPosition(success, error, options);
```

Available Options:

1. enableHighAccuracy

```
enableHighAccuracy: true  // Use GPS (more accurate, slower, more battery)
enableHighAccuracy: false // Use WiFi/cell towers (less accurate, faster)
```

Comparison:

- **true**: Accuracy ~5-10 meters, takes longer, drains battery
- **false**: Accuracy ~100-500 meters, faster, saves battery

When to Use:

- **true**: Navigation apps, delivery tracking, precise location services
- **false**: Store locators, weather apps, general location features

2. timeout

```
timeout: 5000  // Wait maximum 5 seconds for location
```

Prevents indefinite waiting if location can't be determined.

Recommended Values:

- **5000-10000ms**: Standard applications
- **30000ms**: Background location updates
- **Infinity**: No timeout (not recommended)

3. maximumAge

```
maximumAge: 0  // Must get fresh position (your code)
```

```
maximumAge: 60000  // Can use position cached within last 60 seconds
```

Controls whether to use cached location data.

Use Cases:

- **0**: Real-time tracking, navigation
- **30000-300000**: Weather, store locators (less critical precision)

Complete Example with Best Practices:

```
function getUserLocationAndShowMap() {
    // Check if geolocation is supported
    if (!navigator.geolocation) {
        alert("Geolocation is not supported by your browser");
        initMap(40.7128, -74.0060);  // Default location
        return;
    }

    // Show loading indicator
    showLoadingIndicator();

    navigator.geolocation.getCurrentPosition(
        // Success
        function(position) {
            hideLoadingIndicator();
            const lat = position.coords.latitude;
            const lng = position.coords.longitude;

            console.log('Location acquired:', lat, lng);
            console.log('Accuracy:', position.coords.accuracy, 'meters');

            initMap(lat, lng);
        },

        // Error
        function(error) {
            hideLoadingIndicator();

            let errorMsg = '';
            switch(error.code) {
                case error.PERMISSION_DENIED:
                    errorMsg = "You denied location access. Using default
location.";
                    break;
                case error.POSITION_UNAVAILABLE:
```

```
                    errorMsg = "Location unavailable. Using default
location.";
                    break;
                case error.TIMEOUT:
                    errorMsg = "Location request timed out. Using default
location.";
                    break;
            }

            console.error(errorMsg);
            alert(errorMsg);

            // Fallback to default location
            initMap(40.7128, -74.0060);
        },

        // Options
        {
            enableHighAccuracy: true,
            timeout: 10000,
            maximumAge: 5000
        }
    );
}

// Call when page loads
getUserLocationAndShowMap();
```

Markers and Overlays

Creating Markers

Markers are the most common way to identify locations on a map.

Basic Marker Creation:

```
new google.maps.Marker({
    position: myLatLng,
    map: map,
    title: "Hello World!",
});
```

Marker Options Explained:

1. position (Required)

```
position: { lat: 40.7128, lng: -74.0060 }
```

The geographic location where the marker appears.

2. map (Required)

```
map: map  // The map instance to display marker on
```

Associates marker with specific map. Can be `null` to hide marker initially.

3. title (Optional)

```
title: "Hello World!"
```

Tooltip text shown on hover. Similar to HTML `title` attribute.

Advanced Marker Options

Complete Marker Configuration:

```
const marker = new google.maps.Marker({
    position: { lat: 40.7128, lng: -74.0060 },
    map: map,
    title: "New York City",

    // Visual customization
    label: "NYC",                    // Single character label
    icon: "custom-icon.png",         // Custom icon image
    opacity: 0.8,                    // 0.0 (invisible) to 1.0 (opaque)

    // Behavior
    draggable: true,                 // Can user drag marker?
    clickable: true,                 // Can marker be clicked?
    animation: google.maps.Animation.DROP,  // Drop animation

    // Advanced
    zIndex: 100,                     // Stacking order
    optimized: true,                 // Performance optimization
    crossOnDrag: false               // Show crosshair when dragging
});
```

Marker Animations

Available Animations:

1. DROP Animation

```
animation: google.maps.Animation.DROP
```

Marker drops from top of map to position. Great for showing new locations.

2. BOUNCE Animation

```
animation: google.maps.Animation.BOUNCE
```

Marker bounces continuously. Use to highlight important locations.

Example:

```
// Marker drops in
const marker = new google.maps.Marker({
    position: myLatLng,
    map: map,
    animation: google.maps.Animation.DROP
});

// Make marker bounce on click
marker.addListener('click', function() {
    if (marker.getAnimation() !== null) {
        marker.setAnimation(null);  // Stop bouncing
    } else {
        marker.setAnimation(google.maps.Animation.BOUNCE);  // Start bouncing
    }
});

// Stop bouncing after 3 seconds
setTimeout(function() {
    marker.setAnimation(null);
}, 3000);
```

Custom Marker Icons

Using Image URL:

```
const marker = new google.maps.Marker({
    position: myLatLng,
    map: map,
    icon: 'https://example.com/custom-icon.png'
});
```

Custom Icon with Size:

```
const icon = {
    url: 'custom-marker.png',
    scaledSize: new google.maps.Size(50, 50),  // Resize to 50x50px
    origin: new google.maps.Point(0, 0),
    anchor: new google.maps.Point(25, 50)      // Anchor at bottom-center
};

const marker = new google.maps.Marker({
    position: myLatLng,
    map: map,
    icon: icon
});
```

SVG Markers:

```
const svgMarker = {
    path: google.maps.SymbolPath.CIRCLE,
    fillColor: "#FF0000",
    fillOpacity: 0.8,
    strokeColor: "#FFFFFF",
    strokeWeight: 2,
    scale: 10
};

const marker = new google.maps.Marker({
    position: myLatLng,
    map: map,
    icon: svgMarker
});
```

Built-in Symbol Paths:

- `google.maps.SymbolPath.CIRCLE`
- `google.maps.SymbolPath.BACKWARD_CLOSED_ARROW`
- `google.maps.SymbolPath.FORWARD_CLOSED_ARROW`

- `google.maps.SymbolPath.BACKWARD_OPEN_ARROW`
- `google.maps.SymbolPath.FORWARD_OPEN_ARROW`

Marker Labels

Simple Text Label:

```
const marker = new google.maps.Marker({
    position: myLatLng,
    map: map,
    label: "A"  // Single character
});
```

Custom Label Styling:

```
const marker = new google.maps.Marker({
    position: myLatLng,
    map: map,
    label: {
        text: "STORE",
        color: "#FFFFFF",
        fontSize: "14px",
        fontWeight: "bold"
    }
});
```

Info Windows

Info windows display content in a popup above a marker.

Basic Info Window:

```
const infoWindow = new google.maps.InfoWindow({
    content: "<h3>Location Name</h3><p>Address details here</p>"
});

// Show on marker click
marker.addListener('click', function() {
    infoWindow.open(map, marker);
});
```

Complex Info Window:

```
const contentString = `
    <div style="padding: 10px;">
        <h2 style="margin: 0 0 10px 0;">Empire State Building</h2>
        <img src="empire-state.jpg" style="width: 200px; height: 150px;">
        <p><strong>Address:</strong> 20 W 34th St, New York, NY 10001</p>
        <p><strong>Phone:</strong> (212) 736-3100</p>
        <a href="https://www.esbnyc.com/" target="_blank">Visit Website</a>
    </div>
`;

const infoWindow = new google.maps.InfoWindow({
    content: contentString,
    maxWidth: 300
});

marker.addListener('click', function() {
    infoWindow.open(map, marker);
});
```

Managing Multiple Markers

Creating Multiple Markers:

```
const locations = [
    { lat: 40.7128, lng: -74.0060, title: "Location 1" },
    { lat: 40.7580, lng: -73.9855, title: "Location 2" },
    { lat: 40.7489, lng: -73.9680, title: "Location 3" }
];

const markers = locations.map(function(location) {
    return new google.maps.Marker({
        position: { lat: location.lat, lng: location.lng },
        map: map,
        title: location.title
    });
});
```

One Info Window for Multiple Markers:

```
const infoWindow = new google.maps.InfoWindow();

locations.forEach(function(location, index) {
    const marker = new google.maps.Marker({
        position: { lat: location.lat, lng: location.lng },
```

```
        map: map
    });

    marker.addListener('click', function() {
        infoWindow.setContent(`
            <h3>${location.title}</h3>
            <p>Latitude: ${location.lat}</p>
            <p>Longitude: ${location.lng}</p>
        `);
        infoWindow.open(map, marker);
    });
});
```

Removing Markers

Hide Marker:

```
marker.setMap(null);  // Remove from map
```

Show Marker Again:

```
marker.setMap(map);  // Add back to map
```

Delete All Markers:

```
markers.forEach(function(marker) {
    marker.setMap(null);
});
markers = [];  // Clear array
```

Marker Clustering

For many markers, use clustering to improve performance and readability.

Using MarkerClusterer Library:

```
<script src="https://unpkg.com/@googlemaps/markerclusterer/dist/index.min.js">
</script>
```

```
// Create many markers
const markers = locations.map(location => {
    return new google.maps.Marker({
```

```
        position: location,
        map: map
    });
});

// Create cluster
const markerCluster = new markerClusterer.MarkerClusterer({
    map,
    markers
});
```

Map Configuration Options

Complete Map Options

```
const map = new google.maps.Map(document.getElementById("map"), {
    // Position and zoom
    center: { lat: 40.7128, lng: -74.0060 },
    zoom: 15,

    // Map type
    mapTypeId: google.maps.MapTypeId.ROADMAP,

    // UI controls
    zoomControl: true,
    zoomControlOptions: {
        position: google.maps.ControlPosition.RIGHT_CENTER
    },
    mapTypeControl: true,
    scaleControl: true,
    streetViewControl: true,
    rotateControl: true,
    fullscreenControl: true,

    // Interaction
    draggable: true,
    scrollwheel: true,
    disableDoubleClickZoom: false,

    // Styling
    styles: customMapStyles,  // Custom styling
    backgroundColor: '#e5e3df',
```

```
    // Constraints
    minZoom: 5,
    maxZoom: 20,
    restriction: {
        latLngBounds: bounds,  // Restrict panning
        strictBounds: false
    },

    // Advanced
    gestureHandling: 'auto',  // or 'greedy', 'cooperative', 'none'
    clickableIcons: true,
    tilt: 0  // 0 or 45 degrees (3D view)
});
```

Map Types

Available Map Types:

1. ROADMAP (Default)

```
mapTypeId: google.maps.MapTypeId.ROADMAP
```

Standard road map view. Shows streets, labels, and political boundaries.

2. SATELLITE

```
mapTypeId: google.maps.MapTypeId.SATELLITE
```

Satellite imagery only, no labels.

3. HYBRID

```
mapTypeId: google.maps.MapTypeId.HYBRID
```

Satellite imagery with road labels overlaid.

4. TERRAIN

```
mapTypeId: google.maps.MapTypeId.TERRAIN
```

Topographic information showing elevation and water features.

Changing Map Type Dynamically:

```
// Change map type
map.setMapTypeId(google.maps.MapTypeId.SATELLITE);

// Get current map type
const currentType = map.getMapTypeId();
```

Control Positions

Controls can be positioned at different locations:

```
// Available positions
google.maps.ControlPosition.TOP_LEFT
google.maps.ControlPosition.TOP_CENTER
google.maps.ControlPosition.TOP_RIGHT
google.maps.ControlPosition.LEFT_TOP
google.maps.ControlPosition.LEFT_CENTER
google.maps.ControlPosition.LEFT_BOTTOM
google.maps.ControlPosition.RIGHT_TOP
google.maps.ControlPosition.RIGHT_CENTER
google.maps.ControlPosition.RIGHT_BOTTOM
google.maps.ControlPosition.BOTTOM_LEFT
google.maps.ControlPosition.BOTTOM_CENTER
google.maps.ControlPosition.BOTTOM_RIGHT
```

Example:

```
zoomControlOptions: {
    position: google.maps.ControlPosition.LEFT_CENTER
},
mapTypeControlOptions: {
    position: google.maps.ControlPosition.TOP_RIGHT,
    style: google.maps.MapTypeControlStyle.DROPDOWN_MENU
}
```

Custom Map Styling

Create custom color schemes for your map.

Example - Dark Mode:

```
const darkModeStyles = [
    {
        elementType: "geometry",
```

```
        stylers: [{ color: "#242f3e" }]
    },
    {
        elementType: "labels.text.stroke",
        stylers: [{ color: "#242f3e" }]
    },
    {
        elementType: "labels.text.fill",
        stylers: [{ color: "#746855" }]
    },
    {
        featureType: "water",
        elementType: "geometry",
        stylers: [{ color: "#17263c" }]
    }
];

const map = new google.maps.Map(document.getElementById("map"), {
    center: myLatLng,
    zoom: 15,
    styles: darkModeStyles
});
```

Using Snazzy Maps:
Visit Snazzy Maps for pre-made style templates.

Gesture Handling

Controls how users interact with the map.

```
gestureHandling: 'auto'  // Default behavior
gestureHandling: 'greedy'  // Single-finger panning/zooming
gestureHandling: 'cooperative'  // Two-finger panning, prevents accidental
scrolling
gestureHandling: 'none'  // No interaction
```

When to Use:

- **auto**: Standard websites
- **cooperative**: Mobile-first designs, prevents scroll hijacking
- **greedy**: Full-screen map applications
- **none**: Static map displays

Advanced Features

Bounds and Fitting Content

Auto-zoom to Show All Markers:

```
const bounds = new google.maps.LatLngBounds();

locations.forEach(function(location) {
    const marker = new google.maps.Marker({
        position: location,
        map: map
    });

    bounds.extend(marker.getPosition());
});

// Fit map to show all markers
map.fitBounds(bounds);
```

Get Map Bounds:

```
const currentBounds = map.getBounds();
const ne = currentBounds.getNorthEast();
const sw = currentBounds.getSouthWest();

console.log('Northeast:', ne.lat(), ne.lng());
console.log('Southwest:', sw.lat(), sw.lng());
```

Event Listeners

Map Events:

```
// Click on map
map.addListener('click', function(e) {
    console.log('Clicked at:', e.latLng.lat(), e.latLng.lng());

    // Add marker at clicked location
    new google.maps.Marker({
        position: e.latLng,
        map: map
    });
});
```

```
// Zoom changed
map.addListener('zoom_changed', function() {
    console.log('New zoom:', map.getZoom());
});

// Center changed
map.addListener('center_changed', function() {
    const center = map.getCenter();
    console.log('New center:', center.lat(), center.lng());
});

// Map idle (finished moving)
map.addListener('idle', function() {
    console.log('Map is idle');
    // Good time to load new data
});

// Bounds changed
map.addListener('bounds_changed', function() {
    console.log('Bounds changed');
});
```

Marker Events:

```
marker.addListener('click', function() {
    console.log('Marker clicked');
});

marker.addListener('dblclick', function() {
    console.log('Marker double-clicked');
});

marker.addListener('dragend', function() {
    const position = marker.getPosition();
    console.log('Marker moved to:', position.lat(), position.lng());
});

marker.addListener('mouseover', function() {
    marker.setAnimation(google.maps.Animation.BOUNCE);
});

marker.addListener('mouseout', function() {
    marker.setAnimation(null);
});
```

Directions Service

Display routes between locations.

```
const directionsService = new google.maps.DirectionsService();
const directionsRenderer = new google.maps.DirectionsRenderer();

directionsRenderer.setMap(map);

const request = {
    origin: { lat: 40.7128, lng: -74.0060 },
    destination: { lat: 40.7580, lng: -73.9855 },
    travelMode: google.maps.TravelMode.DRIVING
};

directionsService.route(request, function(result, status) {
    if (status === 'OK') {
        directionsRenderer.setDirections(result);

        // Get route info
        const route = result.routes[0];
        const leg = route.legs[0];
        console.log('Distance:', leg.distance.text);
        console.log('Duration:', leg.duration.text);
    }
});
```

Travel Modes:

- `DRIVING`: Car directions
- `WALKING`: Pedestrian paths
- `BICYCLING`: Bike routes
- `TRANSIT`: Public transportation

Places Autocomplete

Add location search with autocomplete.

HTML:

```
<input id="pac-input" type="text" placeholder="Search location">
```

JavaScript:

```
// Requires places library: &libraries=places

const input = document.getElementById('pac-input');
const autocomplete = new google.maps.places.Autocomplete(input);

// Bind to map bounds
autocomplete.bindTo('bounds', map);

// Listen for place selection
autocomplete.addListener('place_changed', function() {
    const place = autocomplete.getPlace();

    if (!place.geometry) {
        console.log("No details available for input: '" + place.name + "'");
        return;
    }

    // Move map to selected place
    if (place.geometry.viewport) {
        map.fitBounds(place.geometry.viewport);
    } else {
        map.setCenter(place.geometry.location);
        map.setZoom(17);
    }

    // Add marker
    new google.maps.Marker({
        position: place.geometry.location,
        map: map,
        title: place.name
    });
});
```

Geocoding

Convert addresses to coordinates or coordinates to addresses.

Address to Coordinates:

```
const geocoder = new google.maps.Geocoder();

geocoder.geocode({ address: "1600 Amphitheatre Parkway, Mountain View, CA" },
    function(results, status) {
        if (status === 'OK') {
            map.setCenter(results[0].geometry.location);
```

```
            new google.maps.Marker({
                map: map,
                position: results[0].geometry.location
            });

            console.log('Coordinates:',
                results[0].geometry.location.lat(),
                results[0].geometry.location.lng()
            );
        } else {
            console.log('Geocode failed: ' + status);
        }
    }
);
```

Coordinates to Address (Reverse Geocoding):

```
const latlng = { lat: 37.4224, lng: -122.0841 };

geocoder.geocode({ location: latlng }, function(results, status) {
    if (status === 'OK') {
        if (results[0]) {
            console.log('Address:', results[0].formatted_address);
        }
    }
});
```

Best Practices

1. Performance Optimization

Minimize Marker Count:

```
// Use clustering for 100+ markers
if (markers.length > 100) {
    new markerClusterer.MarkerClusterer({ map, markers });
}
```

Lazy Load Maps:

```
// Only load map when needed
let mapLoaded = false;

function loadMap() {
    if (!mapLoaded) {
        const script = document.createElement('script');
        script.src = 'https://maps.googleapis.com/maps/api/js?
key=YOUR_KEY&callback=initMap';
        document.head.appendChild(script);
        mapLoaded = true;
    }
}

// Load on user interaction
document.getElementById('showMapBtn').addEventListener('click', loadMap);
```

Use Viewport-based Loading:

```
map.addListener('idle', function() {
    const bounds = map.getBounds();
    // Load only markers within current viewport
    loadMarkersInBounds(bounds);
});
```

2. Security Best Practices

Restrict API Key:

1. **HTTP referrer restrictions**: Limit to your domains
2. **API restrictions**: Enable only needed APIs
3. **Usage quotas**: Set daily limits

Never Expose Keys in Public Repos:

```
// BAD - Key visible in code
const key = 'AIzaSyB41DRUbKWJHPxaFjMAwdrzWzbVKartNGg';

// GOOD - Use environment variables
const key = process.env.GOOGLE_MAPS_API_KEY;

// BETTER - Proxy through backend
fetch('/api/maps-data')  // Backend makes Google Maps request
```

3. User Experience

Show Loading State:

```
// Show loader while getting location
document.getElementById('map').innerHTML = '<div class="loader">Loading map...
</div>';

navigator.geolocation.getCurrentPosition(
    function(pos) {
        // Remove loader
        document.getElementById('map').innerHTML = '';
        initMap(pos.coords.latitude, pos.coords.longitude);
    }
);
```

Provide Fallback:

```
// If geolocation fails, ask user to enter location
function handleLocationError() {
    const input = prompt("We couldn't detect your location. Please enter your
city:");
    if (input) {
        geocodeAddress(input);
    } else {
        // Show default location
        initMap(40.7128, -74.0060);
    }
}
```

Mobile Optimization:

```
/* Ensure map is touch-friendly */
#map {
    touch-action: none;  /* Prevents browser zoom on map */
}

/* Full-screen on mobile */
@media (max-width: 768px) {
    #map {
        height: 100vh;
        width: 100vw;
    }
}
```

4. Error Handling

Comprehensive Error Handling:

```
function initializeMap() {
    // Check browser support
    if (!navigator.geolocation) {
        showError("Geolocation not supported");
        return;
    }

    // Check Google Maps loaded
    if (typeof google === 'undefined') {
        showError("Google Maps failed to load");
        return;
    }

    // Get location with error handling
    navigator.geolocation.getCurrentPosition(
        successCallback,
        errorCallback,
        { enableHighAccuracy: true, timeout: 10000 }
    );
}

function showError(message) {
    document.getElementById('error-message').textContent = message;
    document.getElementById('error-message').style.display = 'block';
}
```

5. Accessibility

Add ARIA Labels:

```
<div id="map" role="application" aria-label="Interactive map showing your
location"></div>
```

Keyboard Navigation:

```
// Allow keyboard focus on map
document.getElementById('map').setAttribute('tabindex', '0');

// Add keyboard shortcuts
document.addEventListener('keydown', function(e) {
```

```
    if (e.target.id === 'map') {
        switch(e.key) {
            case 'ArrowUp':
                map.panBy(0, -100);
                break;
            case 'ArrowDown':
                map.panBy(0, 100);
                break;
            // etc.
        }
    }
});
```

6. Testing

Test Different Scenarios:

```
// Test with mock location
function testMap() {
    const testLocations = [
        { lat: 40.7128, lng: -74.0060, name: "New York" },
        { lat: 51.5074, lng: -0.1278, name: "London" },
        { lat: 35.6762, lng: 139.6503, name: "Tokyo" }
    ];

    testLocations.forEach(location => {
        console.log(`Testing ${location.name}...`);
        initMap(location.lat, location.lng);
    });
}
```

Monitor Performance:

```
console.time('mapLoad');
initMap(lat, lng);
console.timeEnd('mapLoad');  // Check load time
```

Complete Working Example

Here's a production-ready implementation combining all best practices:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Location Finder</title>

    <style>
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        body {
            font-family: Arial, sans-serif;
        }

        #map {
            height: 100vh;
            width: 100%;
        }

        .loading {
            position: absolute;
            top: 50%;
            left: 50%;
            transform: translate(-50%, -50%);
            background: white;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 2px 10px rgba(0,0,0,0.2);
            z-index: 1000;
        }

        .error {
            position: absolute;
            top: 20px;
            left: 50%;
            transform: translateX(-50%);
            background: #ff4444;
            color: white;
            padding: 15px 25px;
            border-radius: 4px;
            z-index: 1000;
            display: none;
```

```
        }

        .controls {
            position: absolute;
            top: 20px;
            left: 20px;
            background: white;
            padding: 15px;
            border-radius: 4px;
            box-shadow: 0 2px 5px rgba(0,0,0,0.2);
            z-index: 999;
        }

        .controls button {
            display: block;
            width: 100%;
            padding: 10px;
            margin: 5px 0;
            cursor: pointer;
            border: none;
            background: #4285f4;
            color: white;
            border-radius: 3px;
        }

        .controls button:hover {
            background: #357ae8;
        }
    </style>
</head>
<body>
    <div id="error" class="error"></div>
    <div id="loading" class="loading">
        <p>Getting your location...</p>
    </div>

    <div class="controls">
        <button onclick="recenterMap()">📍 My Location</button>
        <button onclick="toggleMapType()">🗺️ Change View</button>
        <button onclick="zoomIn()">➕ Zoom In</button>
        <button onclick="zoomOut()">➖ Zoom Out</button>
    </div>

    <div id="map" role="application" aria-label="Interactive location map">
</div>
```

```
    <script>
        let map;
        let userMarker;
        let userLocation;

        // Initialize when page loads
        window.addEventListener('DOMContentLoaded', function() {
            getUserLocation();
        });

        function getUserLocation() {
            // Check support
            if (!navigator.geolocation) {
                showError("Your browser doesn't support geolocation");
                useDefaultLocation();
                return;
            }

            // Show loading
            document.getElementById('loading').style.display = 'block';

            // Get location
            navigator.geolocation.getCurrentPosition(
                handleLocationSuccess,
                handleLocationError,
                {
                    enableHighAccuracy: true,
                    timeout: 10000,
                    maximumAge: 5000
                }
            );
        }

        function handleLocationSuccess(position) {
            userLocation = {
                lat: position.coords.latitude,
                lng: position.coords.longitude
            };

            console.log('Location acquired:', userLocation);
            console.log('Accuracy:', position.coords.accuracy, 'meters');

            // Hide loading
            document.getElementById('loading').style.display = 'none';

            // Initialize map
```

```
            initMap(userLocation.lat, userLocation.lng);
        }

        function handleLocationError(error) {
            // Hide loading
            document.getElementById('loading').style.display = 'none';

            let errorMessage = '';
            switch(error.code) {
                case error.PERMISSION_DENIED:
                    errorMessage = "Location access denied. Using default
location.";
                    break;
                case error.POSITION_UNAVAILABLE:
                    errorMessage = "Location unavailable. Using default
location.";
                    break;
                case error.TIMEOUT:
                    errorMessage = "Location request timed out. Using default
location.";
                    break;
                default:
                    errorMessage = "Unknown error occurred. Using default
location.";
            }

            showError(errorMessage);
            useDefaultLocation();
        }

        function useDefaultLocation() {
            // Default to New York City
            initMap(40.7128, -74.0060);
        }

        function initMap(lat, lng) {
            const position = { lat: lat, lng: lng };

            // Create map
            map = new google.maps.Map(document.getElementById("map"), {
                center: position,
                zoom: 15,
                mapTypeId: google.maps.MapTypeId.ROADMAP,
                zoomControl: true,
                mapTypeControl: true,
                streetViewControl: true,
```

```
            fullscreenControl: true,
            gestureHandling: 'auto'
        });

        // Add marker
        userMarker = new google.maps.Marker({
            position: position,
            map: map,
            title: "You are here!",
            animation: google.maps.Animation.DROP,
            icon: {
                path: google.maps.SymbolPath.CIRCLE,
                fillColor: "#4285f4",
                fillOpacity: 1,
                strokeColor: "#ffffff",
                strokeWeight: 3,
                scale: 10
            }
        });

        // Add info window
        const infoWindow = new google.maps.InfoWindow({
            content: `
                <div style="padding: 10px;">
                    <h3 style="margin: 0 0 5px 0;">Your Location</h3>
                    <p style="margin: 0;">Lat: ${lat.toFixed(6)}</p>
                    <p style="margin: 0;">Lng: ${lng.toFixed(6)}</p>
                </div>
            `
        });

        userMarker.addListener('click', function() {
            infoWindow.open(map, userMarker);
        });

        // Listen for map clicks to add markers
        map.addListener('click', function(e) {
            addMarker(e.latLng);
        });
    }

    function addMarker(location) {
        new google.maps.Marker({
            position: location,
            map: map,
            animation: google.maps.Animation.DROP
```

```
            });
        }

        function recenterMap() {
            if (userLocation && map) {
                map.setCenter(userLocation);
                map.setZoom(15);
            }
        }

        function toggleMapType() {
            if (map) {
                const currentType = map.getMapTypeId();
                const newType = currentType === 'roadmap' ? 'satellite' :
'roadmap';
                map.setMapTypeId(newType);
            }
        }

        function zoomIn() {
            if (map) {
                map.setZoom(map.getZoom() + 1);
            }
        }

        function zoomOut() {
            if (map) {
                map.setZoom(map.getZoom() - 1);
            }
        }

        function showError(message) {
            const errorDiv = document.getElementById('error');
            errorDiv.textContent = message;
            errorDiv.style.display = 'block';

            setTimeout(function() {
                errorDiv.style.display = 'none';
            }, 5000);
        }

        // Make initMap globally accessible for Google Maps callback
        window.initMap = function() {
            // Callback function for Google Maps script
            getUserLocation();
        };
```

```
    </script>

    <script
        src="https://maps.googleapis.com/maps/api/js?
key=YOUR_API_KEY&callback=initMap&v=weekly"
        defer
        async>
    </script>
</body>
</html>
```

Summary

Key Takeaways:

1. **Google Maps API** provides comprehensive mapping functionality for web applications
2. **API Key** is required and should be secured with restrictions
3. **Geolocation API** integrates seamlessly to show user's location
4. **Markers** identify locations and can be fully customized
5. **Event handling** enables interactive features
6. **Best practices** ensure performance, security, and user experience

The code flow in your example:

1. Page loads → Scripts load with `defer`
2. Geolocation API requests user's location
3. User grants permission
4. `initMap()` called with coordinates
5. Map displays centered on user's location
6. Marker shows user's position

This powerful combination creates location-aware applications that provide personalized, interactive mapping experiences.

Abdullah Ali

Contact : +201012613453