Índice

4. Visualização Geoespacial

A visualização geoespacial é uma etapa fundamental para compreender a distribuição espacial de fenômenos. Ela permite comunicar padrões, identificar clusters e tomar decisões baseadas em localização de forma mais eficiente.

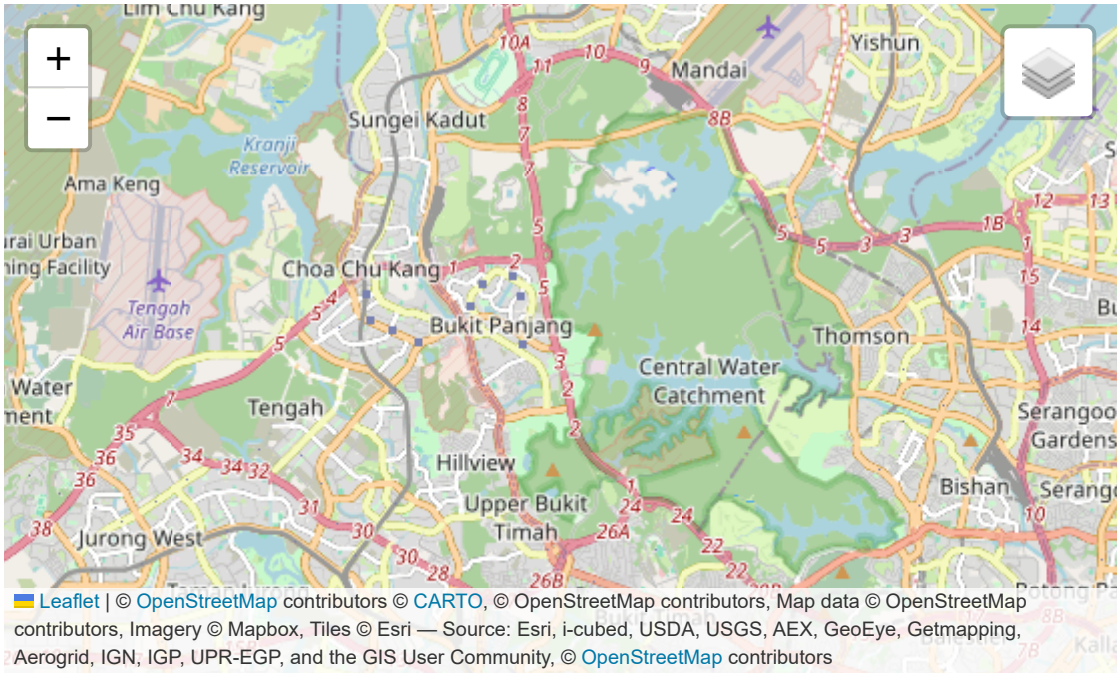
4.1 Camadas de Mapas Base

Folium permite adicionar diferentes estilos de mapas base (tiles), como ruas, satélite e versões claras ou escuras. Essas camadas ajudam a adaptar a visualização ao tipo de dado apresentado.

In [1]:
```
import folium

# Criar o mapa sem tile inicial (usaremos múltiplos)
m = folium.Map(location=[1.3521, 103.8198], zoom_start=12, tiles=None)

# CartoDB (claro e escuro)
folium.TileLayer("CartoDB positron", name="CartoDB Claro").add_to(m)
folium.TileLayer("CartoDB dark_matter", name="CartoDB Escuro").add_to(m)

# Stamen
folium.TileLayer("Stamen Toner", name="Stamen Toner (PB)", attr="© OpenStreetMap
folium.TileLayer("Stamen Terrain", name="Stamen Terreno", attr="© OpenStreetMap 
folium.TileLayer("Stamen Watercolor", name="Stamen Aquarela", attr="© OpenStreet

# Mapbox Streets (personalizado - requer token se premium)
folium.TileLayer(
    tiles='https://api.mapbox.com/styles/v1/mapbox/streets-v11/tiles/{z}/{x}/{y}
    attr='Map data © OpenStreetMap contributors, Imagery © Mapbox',
    name='Mapbox Streets',
    overlay=False,
    control=True
).add_to(m)

# Esri World Imagery (satélite)
folium.TileLayer(
    tiles='https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/Ma
```

```
    attr='Tiles &copy; Esri &mdash; Source: Esri, i-cubed, USDA, USGS, AEX, GeoE
    name='Esri Satélite',
    overlay=False,
    control=True
).add_to(m)

# OpenStreetMap tradicional
folium.TileLayer("OpenStreetMap", name="OpenStreetMap").add_to(m)

# Adicionar controle de camadas
folium.LayerControl().add_to(m)

m
```

Out[1]:



4.2 Mapa de Pontos

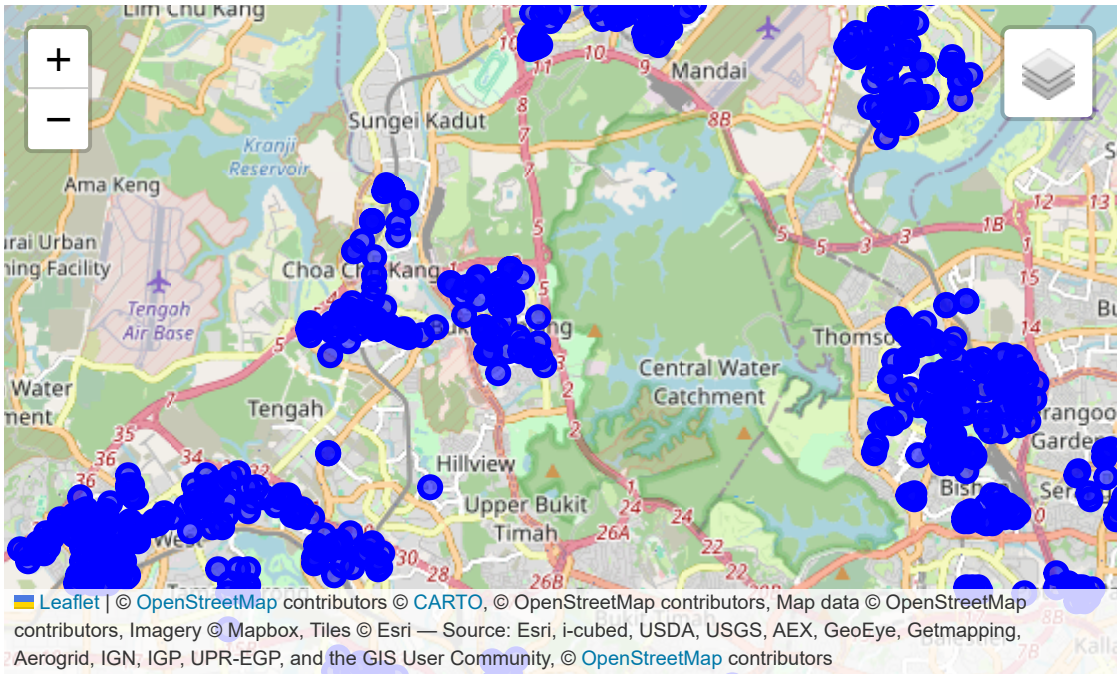
Mapas de pontos são usados para mostrar a localização de eventos ou objetos, como imóveis, escolas ou ocorrências de crimes.

In [2]:

```
import pandas as pd
import folium

# Carregar dados geocodificados de imóveis (já com latitude/longitude)
df = pd.read_csv("datasets/Singapore/geocodificados.csv")
df = df.dropna(subset=["latitude", "longitude"])



# Adicionar marcadores para cada imóvel
for _, row in df.iterrows():
    folium.CircleMarker(
        location=(row["latitude"], row["longitude"]),
        radius=5,
        color="blue",
        fill=True,
        fill_opacity=0.6,
        popup=f"${row['resale_price']:.0f}"
    ).add_to(m)
```

```
m
```

Out[2]:



4.3 Mapa Coropletico

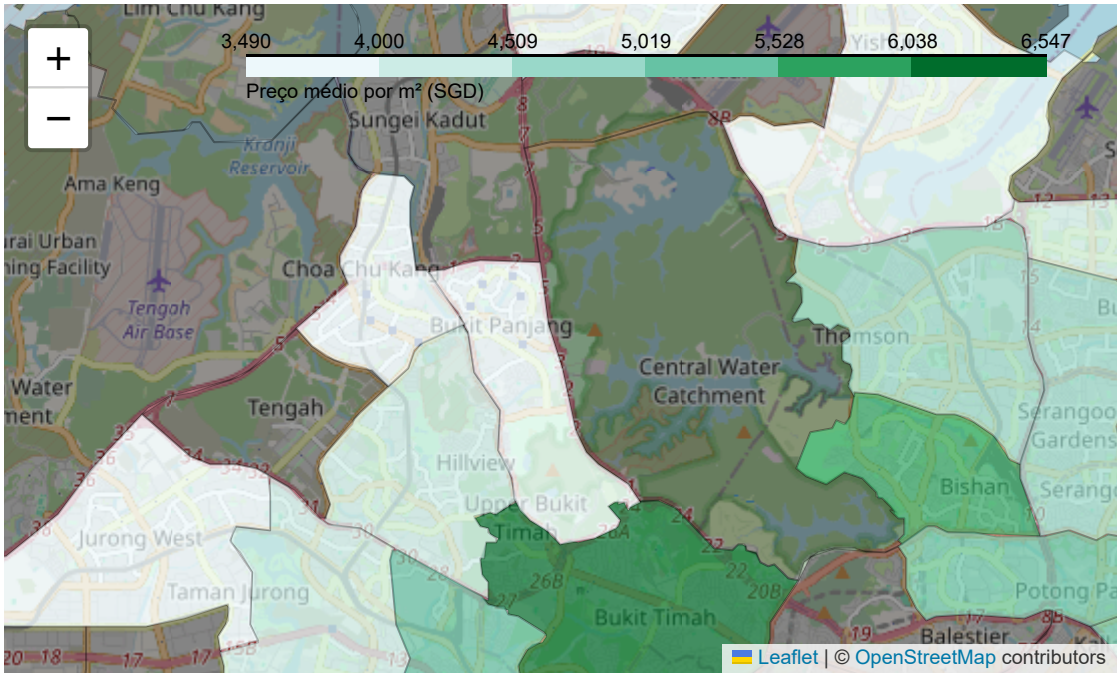
Mapas coropléticos permitem representar atributos agregados por regiões, como preço médio por bairro ou densidade populacional.

In [3]:

```
import geopandas as gpd
import folium
import pandas as pd

# Carregar dados dos imóveis
df = pd.read_csv("datasets/Singapore/geocodificados.csv")
df["preco_m2"] = df["resale_price"] / df["floor_area_sqm"]

# Padronizar nomes dos bairros
df["town_clean"] = df["town"].str.lower().str.strip()

# Calcular média de preço por região padronizada
media_por_town = df.groupby("town_clean")["preco_m2"].mean().reset_index()

# Carregar GeoJSON dos bairros
gdf_towns = gpd.read_file("datasets/Singapore/district_and_planning_area.geojson
gdf_towns["planning_area_clean"] = gdf_towns["planning_area"].str.lower().str.st

# Merge com nomes padronizados
gdf_towns = gdf_towns.merge(media_por_town, left_on="planning_area_clean", right

# Criar mapa coroplético
m_choro = folium.Map(location=[1.3521, 103.8198], zoom_start=12)
folium.Choropleth(
    geo_data="datasets/Singapore/district_and_planning_area.geojson",
    data=gdf_towns,
    columns=["planning_area", "preco_m2"],
    key_on="feature.properties.planning_area",
    fill_color="BuGn",
    fill_opacity=0.7,
```

```
    line_opacity=0.2,
    nan_fill_color="black",  # Cor para regiões sem dados
    nan_fill_opacity=0.4,
    legend_name="Preço médio por m² (SGD)"
).add_to(m_choro)

m_choro
```

Out[3]:



4.4 Mapa de Pontos Classificados por Faixa de Preco

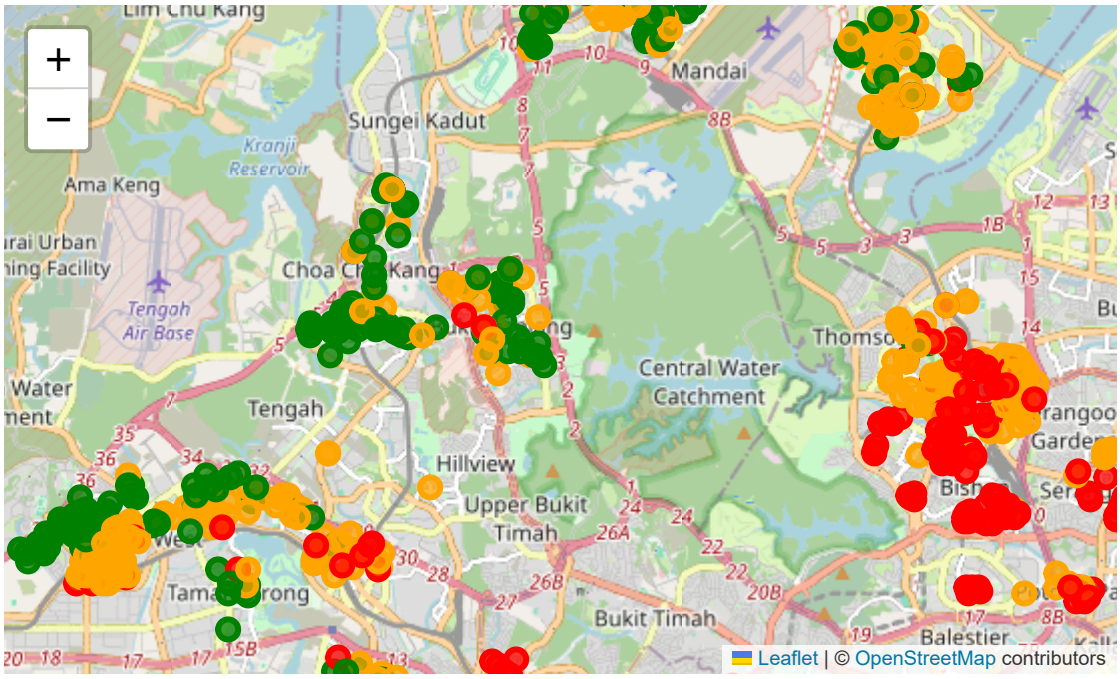
Podemos agrupar visualmente os imóveis com base no preço por metro quadrado, utilizando diferentes cores para faixas de valores.

In [4]:

```
# Calcular preço por metro quadrado
df["preco_m2"] = df["resale_price"] / df["floor_area_sqm"]

# Definir os quartis
q1 = df["preco_m2"].quantile(0.25)
q3 = df["preco_m2"].quantile(0.75)

# Função para classificar
def classificar_preco(valor):
    if valor < q1:
        return "Barato"
    elif valor < q3:
        return "Médio"
    else:
        return "Caro"

# Aplicar classificação e definir cores
df["categoria"] = df["preco_m2"].apply(classificar_preco)
cores = {"Barato": "green", "Médio": "orange", "Caro": "red"}

# Mapa com cores por categoria
m2 = folium.Map(location=[1.3521, 103.8198], zoom_start=12)
df = df.dropna()
```

```
for _, row in df.iterrows():
    folium.CircleMarker(
        location=(row["latitude"], row["longitude"]),
        radius=5,
        color=cores[row["categoria"]],
        fill=True,
        fill_opacity=0.75,
        popup=f"{row['categoria']} - ${row['preco_m2']:.0f}/m²"
    ).add_to(m2)

m2
```

Out[4]:



Podemos agrupar visualmente os imóveis com base no preço por metro quadrado, utilizando diferentes cores para faixas de valores.

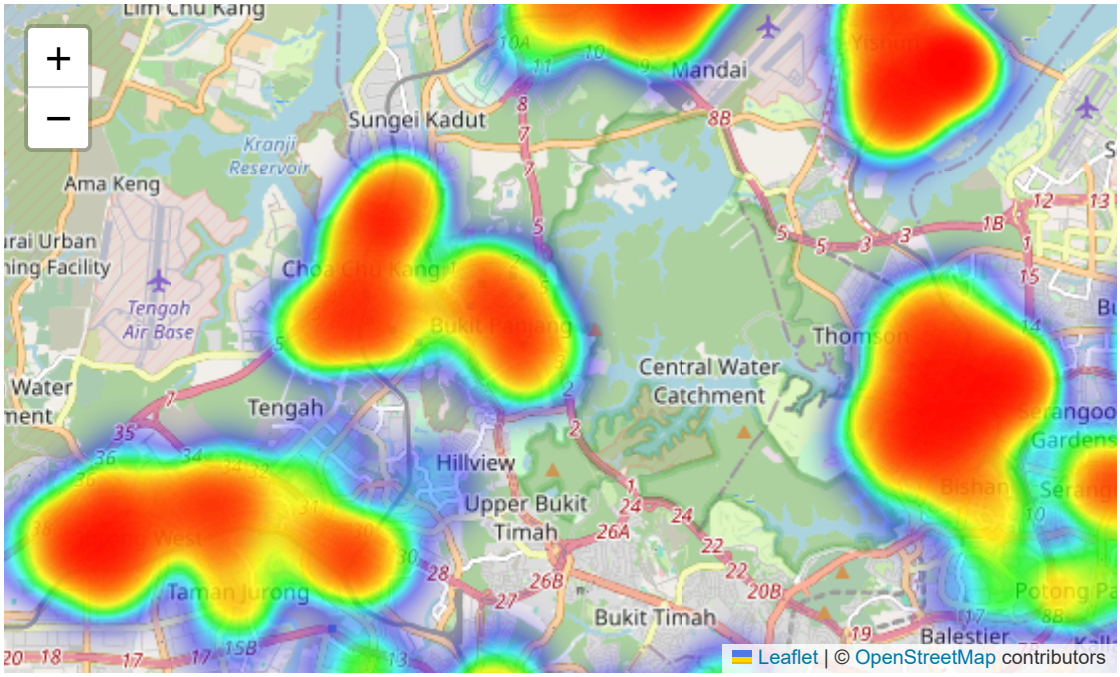
4.5 Heatmap de Densidade de Imoveis

Um heatmap ajuda a identificar as regiões com maior concentração de pontos (ex: imóveis cadastrados).

In [5]:

```
from folium.plugins import HeatMap

# Mapa de calor da densidade de imóveis
m3 = folium.Map(location=[1.3521, 103.8198], zoom_start=12)
HeatMap(data=df[["latitude", "longitude"]]).add_to(m3)
m3
```

Out[5]:



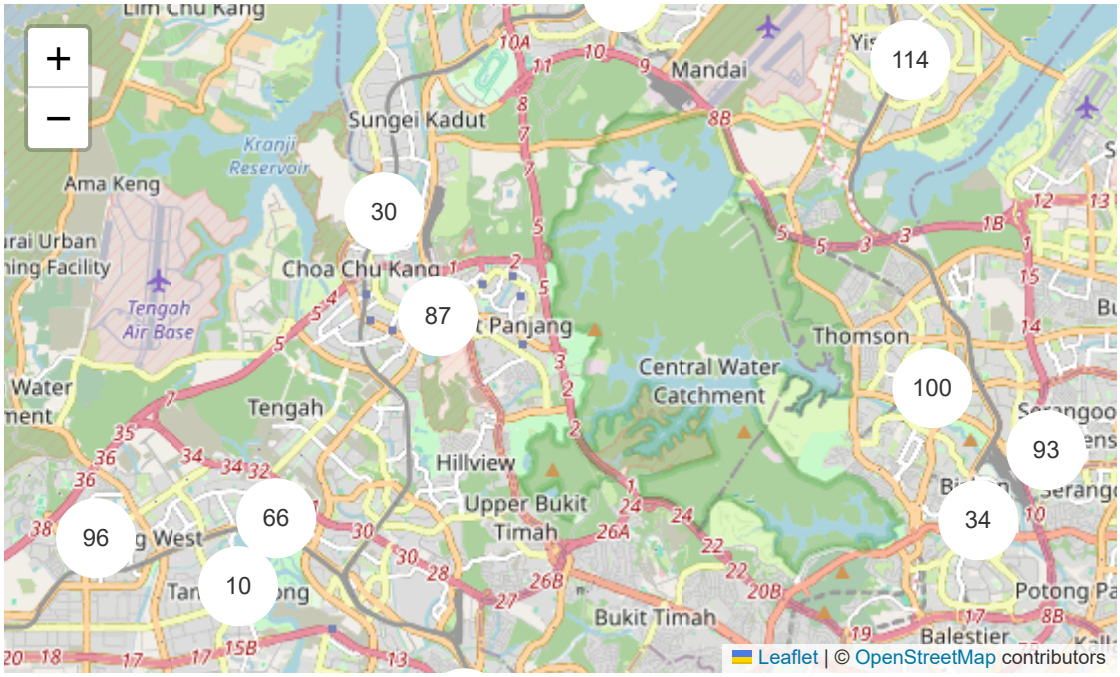
4.6 Clusterizacao de Pontos com Folium

Utilizando `MarkerCluster` conseguimos agrupar automaticamente pontos muito próximos, facilitando a visualização em áreas densas.

In [6]:

```
import pandas as pd
import folium
from folium.plugins import MarkerCluster

# Carregar os dados
df = pd.read_csv("datasets/Singapore/geocodificados.csv")

# Remover registros com coordenadas ausentes
df = df.dropna(subset=["latitude", "longitude"])

# Criar mapa base
m = folium.Map(location=[1.3521, 103.8198], zoom_start=12)

# Criar o cluster de marcadores
marker_cluster = MarkerCluster().add_to(m)

# Adicionar os pontos ao cluster
for _, row in df.iterrows():
    popup_text = f"""
    <b>Preço:</b> ${row['resale_price']:,.0f}<br>
    <b>Área:</b> {row['floor_area_sqm']} m²<br>
    <b>Tipo:</b> {row['flat_type']}<br>
    <b>Bairro:</b> {row['town']}
    """
    folium.Marker(
        location=(row["latitude"], row["longitude"]),
        popup=folium.Popup(popup_text, max_width=250)
    ).add_to(marker_cluster)

# Exibir o mapa
m
```

Out[6]:



4.7 Visualizacao Temporal com Mapa

Mapas com componente temporal permitem explorar como variáveis geoespaciais evoluem ao longo do tempo. Ao associar atributos temporais a localizações, é possível animar eventos, detectar tendências.

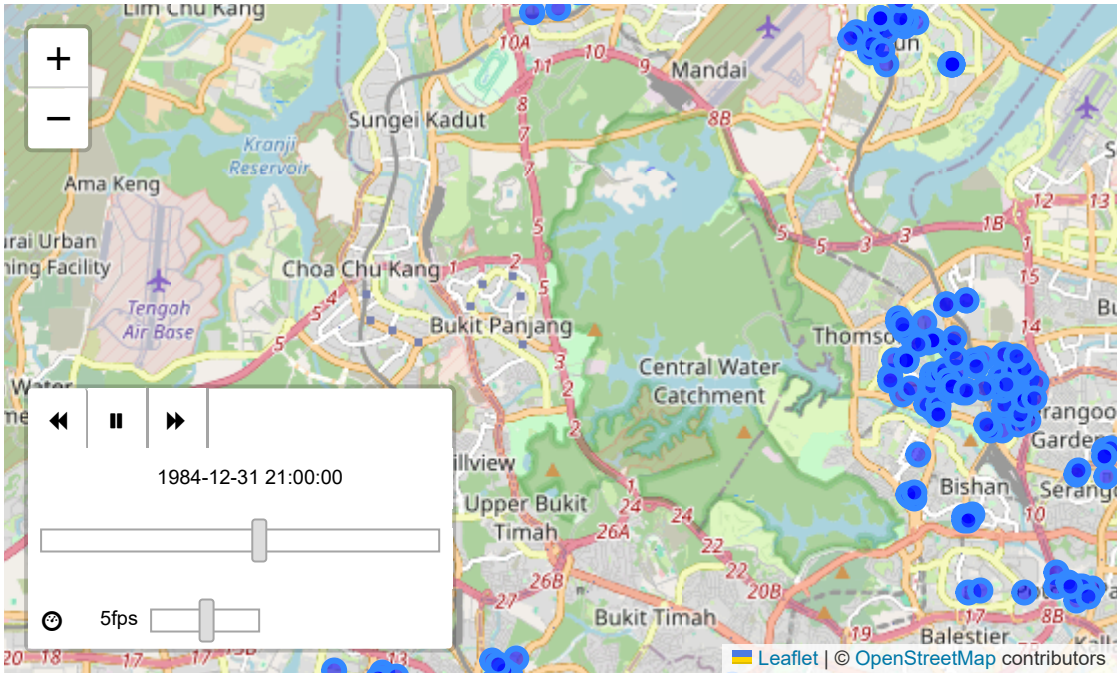
Neste exemplo, utilizamos a data de lançamento dos imóveis para criar uma visualização que combina localização geográfica com a dimensão do tempo. Esse tipo de visualização é útil para identificar padrões de expansão urbana, mudanças na densidade construtiva e movimentos de crescimento regional ao longo dos anos.

In [7]:

```
from folium.plugins import TimestampedGeoJson
import folium
import pandas as pd

# Carregar os dados
df = pd.read_csv("datasets/Singapore/geocodificados.csv")

# Corrigir o tipo de dado e criar timestamp por lease
df["lease_date"] = pd.to_datetime(df["lease_commence_date"].astype(int).astype(s
df["timestamp"] = df["lease_date"].dt.strftime("%Y-%m-%d")

# Construir GeoJSON
features = []
for _, row in df.dropna(subset=["latitude", "longitude"]).iterrows():
    feature = {
        "type": "Feature",
        "geometry": {
            "type": "Point",
            "coordinates": [row["longitude"], row["latitude"]],
        },
        "properties": {
            "time": row["timestamp"],
            "popup": f"{row['town']}<br>Lease: {row['lease_commence_date']}<br>P
            "icon": "circle",
            "iconstyle": {
```

```
                "fillColor": "blue",
                "fillOpacity": 0.6,
                "stroke": "true",
                "radius": 5
            },
        },
    }
    features.append(feature)

geojson = {
    "type": "FeatureCollection",
    "features": features
}

# Criar mapa
m_time = folium.Map(location=[1.3521, 103.8198], zoom_start=12)
TimestampedGeoJson(geojson, period="P1Y", add_last_point=True).add_to(m_time)
m_time
```

Out[7]:



4.8 Heatmap Temporal Acumulativo

Com o `HeatMapWithTime`, é possível visualizar a evolução da densidade geográfica ao longo do tempo. No caso dos imóveis, esse recurso permite observar como a urbanização se expandiu ano após ano.

Ao invés de mostrar apenas os imóveis lançados em cada ano isoladamente, aqui acumulamos os dados: a cada nova etapa da animação, os imóveis dos anos anteriores são mantidos. Isso permite perceber o crescimento urbano e a concentração de novas unidades ao longo das décadas.

Essa visualização é especialmente útil para análises temporais e urbanísticas.

In [8]:

```
from folium.plugins import HeatMapWithTime
import pandas as pd
import folium
```

```
# Carregar e preparar os dados
df = pd.read_csv("datasets/Singapore/geocodificados.csv")
df = df.dropna(subset=["latitude", "longitude", "lease_commence_date"])
df["lease_commence_date"] = df["lease_commence_date"].astype(int)

# Ordenar anos
anos = sorted(df["lease_commence_date"].unique())
heat_data = []
pontos_acumulados = []

# Construir lista acumulativa de pontos
for ano in anos:
    subset = df[df["lease_commence_date"] == ano]
    novos_pontos = subset[["latitude", "longitude"]].values.tolist()
    pontos_acumulados.extend(novos_pontos)
    heat_data.append(pontos_acumulados.copy())

# Criar o mapa
m_heat = folium.Map(location=[1.3521, 103.8198], zoom_start=12)

# Adicionar heatmap com animação acumulativa
HeatMapWithTime(
    heat_data,
    index=anos,
    auto_play=True,
    radius=10,
    max_opacity=0.8
).add_to(m_heat)

# Estatísticas adicionais
contagem = df["lease_commence_date"].value_counts().sort_index()
ano_top = contagem.idxmax()
qtd_top = contagem.max()
print(f"O ano com mais lançamentos foi {ano_top}, com {qtd_top} imóveis.")

m_heat
```

```
O ano com mais lançamentos foi 1985, com 92 imóveis.
```

Out[8]: