ArcGIS API for JavaScript: Building Progressive Web Apps

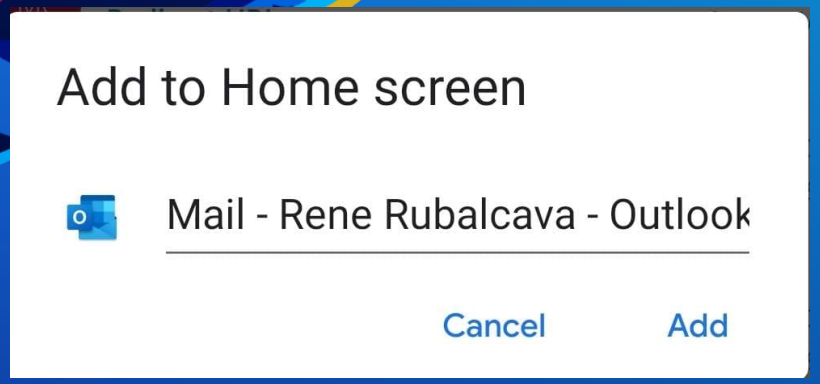
Andy Gup @agup
René Rubalcava @odoenet

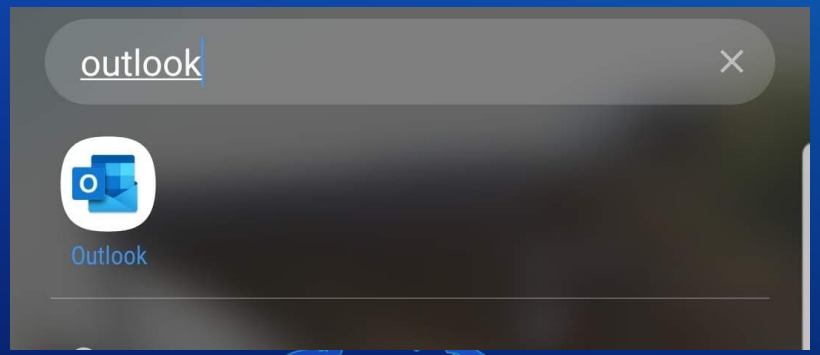
2021 ESRI DEVELOPER SUMMIT

Progressive Web Apps Agenda

- What is a progressive web app (PWA)
- Requirements
- Building progressive web apps with the ArcGIS API for JavaScript
- Mobile Web Performance

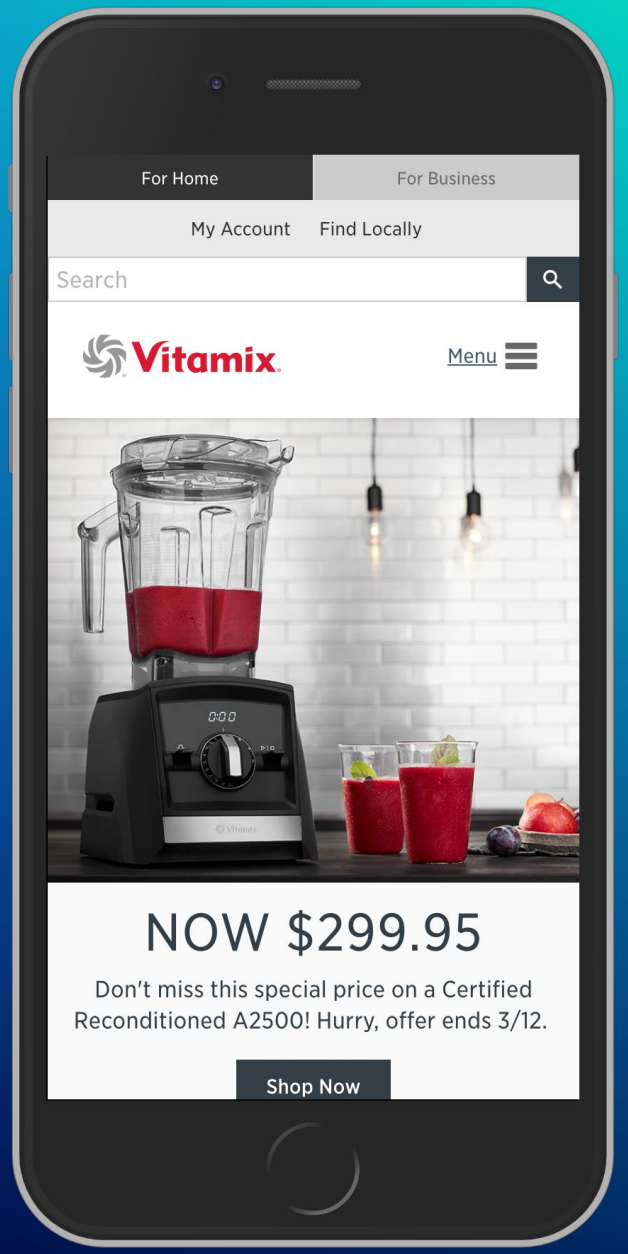
What is a PWA?





- **Web apps that *behave* like native apps**
  - **Prompt add to home screen**
  - **Run in a sandbox environment**

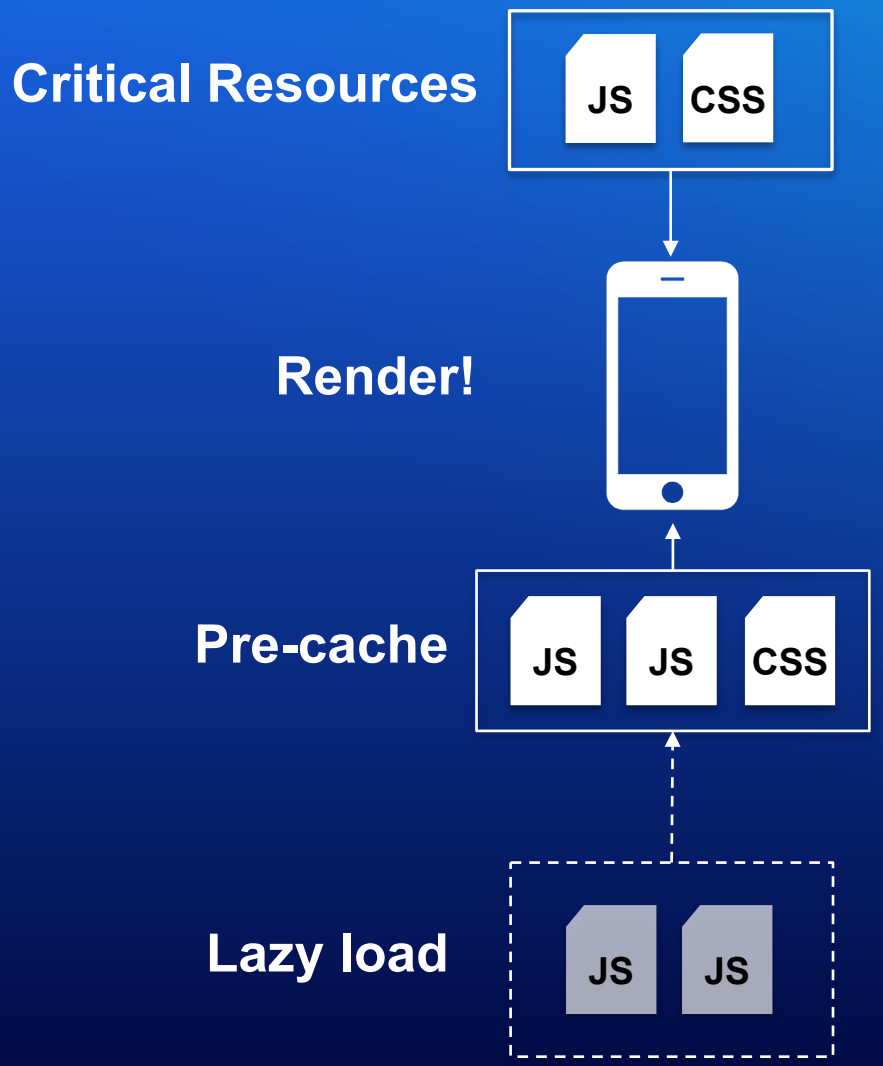
- Smooth transitions
- Responsive
- Fast loading
- Native look and feel



PWA Requirements

PRPL

- Push critical resources
- Render initial page or route
- Pre-cache remaining assets
- Lazy load parts of your application



Service Workers

- *Not* a web worker
- Control and cache network requests
- Series of APIs
  - Notification – send notification to desktop
  - Push – pub/sub capabilities
  - Background Sync – useful for offline to ensure data is sent
  - Channel Messaging – communication with other works, ie: iframes

App Shell

- Useful for fast loading
- Minimal HTML, JS, CSS required to display a UI
- Can allow an app to *look ready* while resources load
- When to use?
  - When UI doesn’t really change
  - But content does
- Separate content from navigation

manifest.json

- How your PWA will behave when installed as a home screen app
- Defines
  - App Name
  - Icons
  - Colors (themes)
  - Display information

```
{
  "icons": [
    {
      "src": "icon_96x96.fa735a06601c1bfae853372d6e77d4dc.png",
      "sizes": "96x96",
      "type": "image/png"
    }
  ],
  "name": "ArcGIS Nearby Places",
  "short_name": "ArcGISNearby",
  "orientation": "portrait",
  "display": "standalone",
  "start_url": ".",
  "description": "Nearby Places",
  "background_color": "#0079c1",
  "theme_color": "#0079c1"
}
```

PWA Tips

Native Image Lazy Loading

- **Built into modern browsers**

```html
<img src="image.png" loading="lazy" alt="…" width="200" height="200">
<iframe src="https://example.com" loading="lazy"></iframe>
```

```html
<picture>
  <source media="(min-width: 800px)" srcset="large.jpg 1x, larger.jpg 2x">
  <img src="photo.jpg" loading="lazy">
</picture>
```

https://web.dev/native-lazy-loading/

IntersectionObserver

- JavaScript can do anything

```
1 const io = new IntersectionObserver(
2   entries => {
3     console.log(entries);
4   }
5 );
6 // Start observing an element
7 io.observe(element);
8
9 // Stop observing an element
10 io.unobserve(element);
11
12 // Disable entire IntersectionObserver
13 io.disconnect();
```

https://developers.google.com/web/updates/2016/04/intersectionobserver

WebP Images

- **Load webp if you can, have jpg/png fallbacks**

```html
<picture>
  <source srcset="img/awesomeWebPImage.webp" type="image/webp">
  <source srcset="img/creakyOldJPEG.jpg" type="image/jpeg">
  <img src="img/creakyOldJPEG.jpg" alt="Alt Text!">
</picture>
```

https://developers.google.com/web/updates/2016/04/intersectionobserver

PWA w/ JSAPI

Considerations when building mapping apps

- Provide app shell when possible
- Delay the loading of the map if you can
- JavaScript API is powerful, lots of capabilities
  - What layers are in webmap?
  - What portals are used in webmaps? (Identity)
  - What are the capabilities of the services?
  - Hosted or enterprise?
  - Does data need to be reprojected?

Considerations when building mapping apps

- Provide app shell when possible
- Delay the loading of the map if you can
- JavaScript API is powerful, lots of capabilities
  - What layers are in webmap?
  - What portals are used in webmaps? (Identity)
  - What are the capabilities of the services?
  - Hosted or enterprise?
  - Does data need to be reprojected?

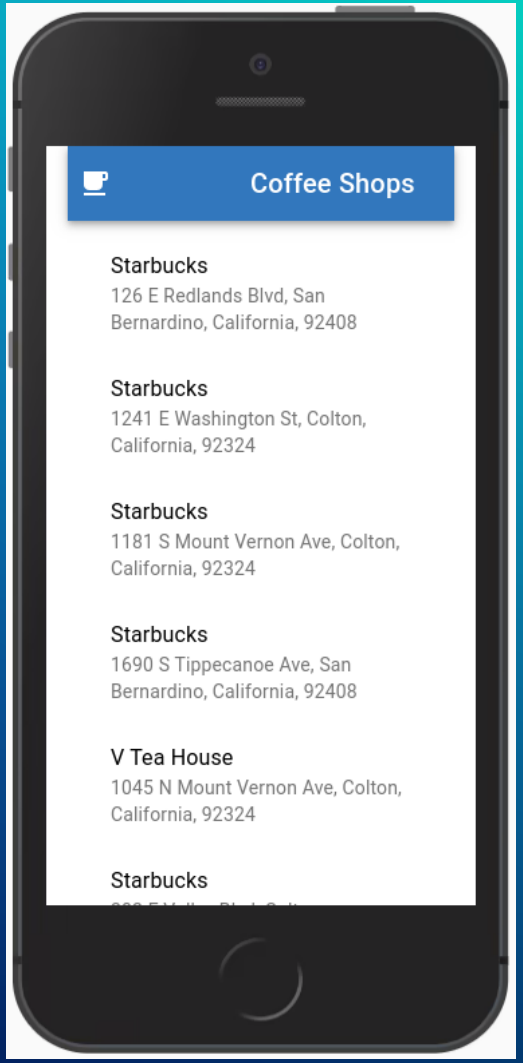
```
type OperationalLayer =
  | "FeatureLayer"
  | "ImageryLayer"
  | "MapImageLayer"
  | "StreamLayer"
  | "TileLayer"
  | "VectorTileLayer"
  | "WebTileLayer"
  | "CSVLayer"
  | "GeoRSSLayer"
  | "KMLLayer"
  | "WMSLayer"
  | "BingMapsLayer"
  | "TileLayer";
```

Patterns

- Determine your app/map pattern
- Mapless apps
  - Map not the initial focus
  - Spatial tasks run in the background
  - Opportunity to delay map loading
  - Opportunity to use routing, lazy loading

Demo Application

- Coffee Shop App
- Uses mapless app pattern



Understanding Performance Focusing on Usability

Andy

User-centric performance

Why focus on performance?

- End user productivity
- End user retention
- End user happiness

What works well on a desktop…

… may not work well on a smaller device.

Your end users are paying attention

They will notice any operation longer than . . .

16ms

Or < 60 fps

Why should we pay attention to performance?

Our users care

Users want to be productive

They want apps to perform well

What performance should be measured?

Initial load performance

User workflow performance

User-centric performance

Where do I start?

- ✓ Do you have downgraded hardware to test against?
- ✓ Do you test with both slow and fast internet download speeds?

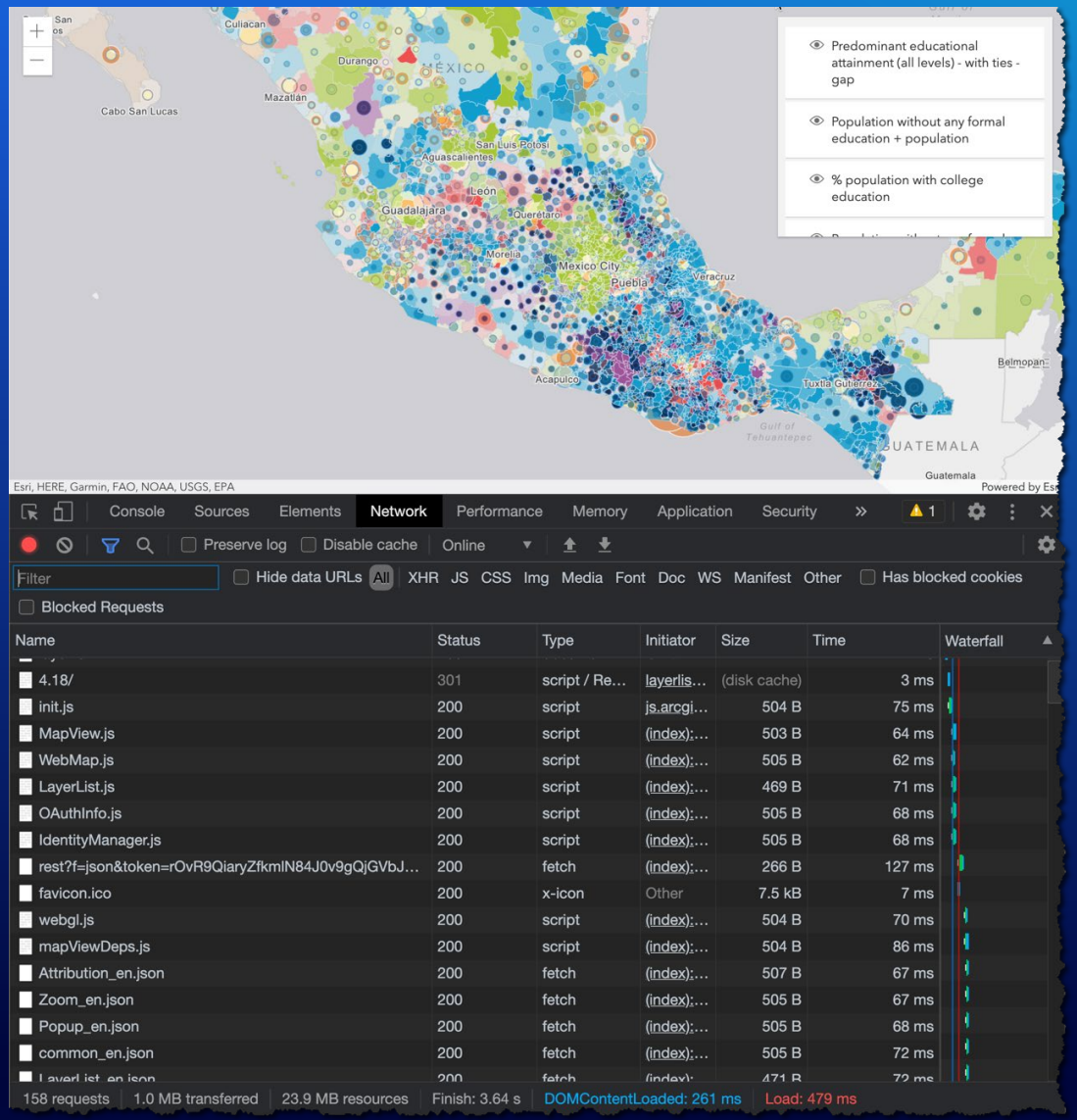
Not all end users have fast internet

Not all end users have the latest mobile devices

Developers often have high(er) end machines

What performance should be measured?

- Time to first map tile to display
- Time for User Interface to fully load
- Time for features completely load
- Time for widget actions to complete



Chrome Devtools

× Headers Preview Response Initiator Timing Cookies

▼ Request Headers

```
:authority: js.arcgis.com
:method: GET
:path: /4.18/esri/views/MapView.js
:scheme: https
accept: */*
accept-encoding: gzip, deflate, br
accept-language: en-US,en;q=0.9
cookie: esri_gdpr=true; sat_track=true; OptanonConsent=nul
l=undefined&geolocation=US%3BCA&datestamp=Thu+Feb+04+2021
+18%3A01%3A40+GMT-0700+(Mountain+Standard+Time)&version=
6.10.0&isIABGlobal=false&hosts=&consentId=10751119-ae91-4
b94-a71a-a8eb38353413&interactionCount=1&landingPath=NotL
andingPage&groups=1%3A1%2C2%3A1%2C3%3A1%2C4%3A1&AwaitingR
econsent=false; OptanonAlertBoxClosed=2021-02-05T01:04:0
3.324Z
```

× Headers Preview Response Initiator Timing Cookies

Queued at 250.93 ms

Started at 251.69 ms

Resource Scheduling	DURATION
Queueing	0.76 ms
Connection Start	**DURATION**
Stalled	0.94 ms
Request/Response	**DURATION**
Request sent	0.19 ms
Waiting (TTFB)	60.33 ms
Content Download	2.41 ms
Explanation	**64.62 ms**

Console | Sources | Elements | Network | Performance | Memory | Application | Security

web.local #1 | Screenshots | Memory | Web Vitals

Disable JavaScript samples | Network: Online

Enable advanced paint instrumentation (slow) | CPU: No throttling

2000 ms | 4000 ms | 6000 ms | 1873 ms | 3873 ms | 5873 ms | 7873 ms | 9873 ms | 1873 ms | 13873 ms

FPS | CPU | NET | HEAP 13.4 MB – 52.4 MB

73 ms | 7373 ms | 7873 ms | 8373 ms | 8873 ms | 9373 ms | 9873 ms | 10373 ms | 10873 n

Web Vitals

LCP 7957.8 ms

LONG TASKS

Frames 1168.0 ms

JS Heap | Documents | Nodes | Listeners | GPU Memory

Summary | Bottom-Up | Call Tree | Event Log

Filter | Group by Activity

Self Time	Total Time	Activity
1740.6 ms 78.1 %	1740.6 ms 78.1 %	JavaScript
129.7 ms 5.8 %	129.7 ms 5.8 %	Minor GC
92.0 ms 4.1 %	92.0 ms 4.1 %	Function Call
57.2 ms 2.6 %	57.2 ms 2.6 %	Run Microtasks
38.3 ms 1.7 %	38.3 ms 1.7 %	Recalculate Style
27.2 ms 1.2 %	27.2 ms 1.2 %	Update Layer Tree

Total blocking time: 987.84ms (estimated) Learn more

Self Time	Total Time	Activity	
1740.6 ms 78.1 %	1740.6 ms 78.1 %	JavaScript	
238.7 ms 10.7 %	238.7 ms 10.7 %	useProgram	
169.3 ms 7.6 %	304.6 ms 13.7 %	F	(index):436
63.4 ms 2.8 %	63.4 ms 2.8 %	fetch	
44.9 ms 2.0 %	141.0 ms 6.3 %	m	(index):252
43.8 ms 2.0 %	47.7 ms 2.1 %	z.buildSDF	mapViewDeps.js:551

Improving performance – using ArcGIS Online services

ArcGIS Online + JS API has many performance enhancements

- Global CDN + CDN caching + high scalability
- Feature tiling
- PBF binary response protocol
- Quantization
- Brotli compression
- Blog: https://bit.ly/3tt0YYv

Performance Enhancements Added	Payload Size	Percent Reduction in Payload
+ Brotli compression	2.3 MB	65
+ Quantization of coordinates	1.0 MB	85
+ Request only required fields	53 KB	99.2
+ PBF	40 KB	99.4
+ Feature tile cache	40 KB	99.4

Improving performance – static vs dynamic module loading

```
require([
  "esri/Map",
  "esri/views/MapView",
  "esri/layers/FeatureLayer",
  "esri/layers/VectorTileLayer",
  "esri/layers/GraphicsLayer",
  "esri/core/watchUtils",
  "esri/core/promiseUtils",
  "esri/geometry/Polygon",
  "esri/geometry/geometryEngine",
  "esri/geometry/support/webMercatorUtils",
  "esri/renderers/DictionaryRenderer",
  "esri/smartMapping/renderers/color",
  "esri/smartMapping/statistics/histogram",
  "esri/widgets/smartMapping/ColorSlider",
  "esri/widgets/Sketch/SketchViewModel",
  "esri/widgets/Expand",
  "esri/widgets/Bookmarks",
  "esri/widgets/Expand",
  "esri/widgets/Slider",
  "esri/widgets/Home",
```

Improving Mobile Performance – dynamic lazy loading

Reduce size of initial app load

Defer loading modules if they aren't needed at load time

Demo

```
function lazyLoadTest(){
    require([
      "esri/geometry/geometryEngine"
    ], function(geometryEngine) {
      console.log("geometryEngine lazy loaded!");
    });
}
```

Improving Performance – Lazy load layers

```
// Load layers immediately
const map = new Map({
  layers: [featureLayer, graphicsLayer]
})

// Or, delay loading layers until you need them
function lazyLoadLayers() {
  map.addMany([featureLayer, graphicsLayer])
}
```

Improving Performance – Use as few layers as possible

Fewer layers means:

- Faster initial load
- Less initial memory usage
- Less CPU/GPU parsing and rendering

Demo 1

Demo 2

Improving Performance – visible vs remove() vs destroy()

`Layer.visible` - Temporary

- Features are still in memory and can be made visible again without re-querying the server.

`Map.remove(layer)` - Longer duration

- Layer completely removed from Map but still in memory
- Can be reused.

Layer.destroy() – permanent removed map from parent and memory

Demo

Improving Performance – Simplify the data

More features == slower load performance

What can I do?

- Set `minScale` and `maxScale` on layers.
- Generalize your data - reduce the number of vertices
- Enable web server compression (gzip, brotli)
- Enabled CORS, remove proxy servers

Demo

Improving Performance – Simplify the data

```
const layerZipcodes = new FeatureLayer({
    url: "https://services.arcgis.com/V6ZHFr6zdgNZuVG0/arc
    maxScale: 0,
    renderer: renderer
});

const layerCounties = new FeatureLayer({
    url: "https://services.arcgis.com/V6ZHFr6zdgNZuVG0/Arc
    minScale: 4000000,
    maxScale: 1000000,
    visible: false,
    renderer: renderer
});

const layerStates = new FeatureLayer({
    url: "https://services.arcgis.com/V6ZHFr6zdgNZuVG0/Arc
    maxScale: 4000000,
    visible: false,
    renderer: renderer
});
```

Contact information:

- **Andy Gup @agup,**
  - agup@esri.com,
  - https://github.com/andygup/pwa-ds2021
- **Rene Rubalcava @odoenet**

esri®

THE SCIENCE OF WHERE®