Angular service worker introduction

Service workers augment the traditional web deployment model and empower applications to deliver a user experience with the reliability and performance on par with code that is written to run on your operating system and hardware. Adding a service worker to an Angular application is one of the steps for turning an application into a Progressive Web App (also known as a PWA).

At its simplest, a service worker is a script that runs in the web browser and manages caching for an application.

Service workers function as a network proxy. They intercept all outgoing HTTP requests made by the application and can choose how to respond to them. For example, they can query a local cache and deliver a cached response if one is available. Proxying isn't limited to requests made through programmatic APIs, such as `fetch`; it also includes resources referenced in HTML and even the initial request to `index.html`. Service worker-based caching is thus completely programmable and doesn't rely on server-specified caching headers.

Unlike the other scripts that make up an application, such as the Angular application bundle, the service worker is preserved after the user closes the tab. The next time that browser loads the application, the service worker loads first, and can intercept every request for resources to load the application. If the service worker is designed to do so, it can *completely satisfy the loading of the application, without the need for the network.*

Even across a fast reliable network, round-trip delays can introduce significant latency when loading the application. Using a service worker to reduce dependency on the network can significantly improve the user experience.

Service workers in Angular

Angular applications, as single-page applications, are in a prime position to benefit from the advantages of service workers. Starting with version 5.0.0, Angular ships with a service worker implementation. Angular developers can take advantage of this service worker and benefit from the increased reliability and performance it provides, without needing to code against low-level APIs.

Angular's service worker is designed to optimize the end user experience of using an application over a slow or unreliable network connection, while also minimizing the risks of serving outdated content.

To achieve this, the Angular service worker follows these guidelines:

- Caching an application is like installing a native application. The application is cached as one unit, and all files update together.
- A running application continues to run with the same version of all files. It does not suddenly start receiving cached files from a newer version, which are likely incompatible.

- When users refresh the application, they see the latest fully cached version. New tabs load the latest cached code.
- Updates happen in the background, relatively quickly after changes are published. The previous version of the application is served until an update is installed and ready.
- The service worker conserves bandwidth when possible. Resources are only downloaded if they've changed.

To support these behaviors, the Angular service worker loads a *manifest* file from the server. The file, called `ngsw.json` (not to be confused with the web app manifest), describes the resources to cache and includes hashes of every file's contents. When an update to the application is deployed, the contents of the manifest change, informing the service worker that a new version of the application should be downloaded and cached. This manifest is generated from a CLI-generated configuration file called `ngsw-config.json`.

Installing the Angular service worker is as straightforward as including an `NgModule`. In addition to registering the Angular service worker with the browser, this also makes a few services available for injection which interact with the service worker and can be used to control it. For example, an application can ask to be notified when a new update becomes available, or an application can ask the service worker to check the server for available updates.

Prerequisites

To make use of all the features of Angular service workers, use the latest versions of Angular and the Angular CLI.

For service workers to be registered, the application must be accessed over HTTPS, not HTTP. Browsers ignore service workers on pages that are served over an insecure connection. The reason is that service workers are quite powerful, so extra care is needed to ensure the service worker script has not been tampered with.

There is one exception to this rule: to make local development more straightforward, browsers do *not* require a secure connection when accessing an application on `localhost`.

Browser support

To benefit from the Angular service worker, your application must run in a web browser that supports service workers in general. Currently, service workers are supported in the latest versions of Chrome, Firefox, Edge, Safari, Opera, UC Browser (Android version) and Samsung Internet. Browsers like IE and Opera Mini do not support service workers.

If the user is accessing your application with a browser that does not support service workers, the service worker is not registered and related behavior such

as offline cache management and push notifications does not happen. More specifically:

- The browser does not download the service worker script and the `ngsw.json` manifest file.
- Active attempts to interact with the service worker, such as calling `SwUpdate.checkForUpdate()`, return rejected promises.
- The observable events of related services, such as `SwUpdate.available`, are not triggered.

It is highly recommended that you ensure that your application works even without service worker support in the browser. Although an unsupported browser ignores service worker caching, it still reports errors if the application attempts to interact with the service worker. For example, calling `SwUpdate.checkForUpdate()` returns rejected promises. To avoid such an error, check whether the Angular service worker is enabled using `SwUpdate.isEnabled`.

To learn more about other browsers that are service worker ready, see the Can I Use page and MDN docs.

Related resources

The rest of the articles in this section specifically address the Angular implementation of service workers.

- App Shell
- Service Worker Communication
- Service Worker Notifications
- Service Worker in Production
- Service Worker Configuration

For more information about service workers in general, see Service Workers: an Introduction.

For more information about browser support, see the browser support section of Service Workers: an Introduction, Jake Archibald's Is Serviceworker ready?, and Can I Use.

For additional recommendations and examples, see:

- Precaching with Angular Service Worker
- Creating a PWA with Angular CLI

Next steps

To begin using Angular service workers, see Getting Started with service workers.