Making a Progressive Web App

The production build has all the tools necessary to generate a first-class Progressive Web App, but **the offline/cache-first behavior is opt-in only**.

Starting with Create React App 4, you can add a `src/service-worker.js` file to your project to use the built-in support for Workbox's `InjectManifest` plugin, which will compile your service worker and inject into it a list of URLs to precache.

If you start a new project using one of the PWA custom templates, you'll get a `src/service-worker.js` file that serves as a good starting point for an offline-first service worker:

```
npx create-react-app my-app --template cra-template-pwa
```

The TypeScript equivalent is:

```
npx create-react-app my-app --template cra-template-pwa-typescript
```

If you know that you won't be using service workers, or if you'd prefer to use a different approach to creating your service worker, don't create a `src/service-worker.js` file. The `InjectManifest` plugin won't be run in that case.

In addition to creating your local `src/service-worker.js` file, it needs to be registered before it will be used. In order to opt-in to the offline-first behavior, developers should look for the following in their `src/index.js` file:

```
// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://cra.link/PWA
serviceWorkerRegistration.unregister();
```

As the comment states, switching `serviceWorker.unregister()` to `serviceWorker.register()` will opt you in to using the service worker.

Why Opt-in?

Offline-first Progressive Web Apps are faster and more reliable than traditional web pages, and provide an engaging mobile experience:

- All static site assets that are a part of your webpack build are cached so that your page loads fast on subsequent visits, regardless of network connectivity (such as 2G or 3G). Updates are downloaded in the background.
- Your app will work regardless of network state, even if offline. This means your users will be able to use your app at 10,000 feet and on the subway.
- On mobile devices, your app can be added directly to the user's home screen, app icon and all. This eliminates the need for the app store.

However, they can make debugging deployments more challenging.

The workbox-webpack-plugin is integrated into production configuration, and it will take care of compiling a service worker file that will automatically precache all of your webpack-generated assets and keep them up to date as you deploy updates. The service worker will use a cache-first strategy for handling all requests for webpack-generated assets, including navigation requests for your HTML, ensuring that your web app is consistently fast, even on a slow or unreliable network.

Note: Resources that are not generated by webpack, such as static files that are copied over from your local public/ directory or third-party resources, will not be precached. You can optionally set up Workbox routes to apply the runtime caching strategy of your choice to those resources.

Customization

Starting with Create React App 4, you have full control over customizing the logic in this service worker, by creating your own src/service-worker.js file, or customizing the one added by the cra-template-pwa (or cra-template-pwa-typescript) template. You can use additional modules from the Workbox project, add in a push notification library, or remove some of the default caching logic. The one requirement is that you keep self.__WB_MANIFEST somewhere in your file, as the Workbox compilation plugin checks for this value when generating a manifest of URLs to precache. If you would prefer not to use precaching, you can assign self.__WB_MANIFEST to a variable that will be ignored, like:

```
// eslint-disable-next-line no-restricted-globals
const ignored = self.__WB_MANIFEST;

// Your custom service worker code goes here.
```

Offline-First Considerations

If you do decide to opt-in to service worker registration, please take the following into account:

1. After the initial caching is done, the service worker lifecycle controls when updated content ends up being shown to users. In order to guard against race conditions with lazy-loaded content, the default behavior is

to conservatively keep the updated service worker in the “waiting” state. This means that users will end up seeing older content until they close (reloading is not enough) their existing, open tabs. See this blog post for more details about this behavior.

2. Users aren’t always familiar with offline-first web apps. It can be useful to let the user know when the service worker has finished populating your caches (showing a “This web app works offline!” message) and also let them know when the service worker has fetched the latest updates that will be available the next time they load the page (showing a “New content is available once existing tabs are closed.” message). Showing these messages is currently left as an exercise to the developer, but as a starting point, you can make use of the logic included in `src/serviceWorkerRegistration.js`, which demonstrates which service worker lifecycle events to listen for to detect each scenario, and which as a default, only logs appropriate messages to the JavaScript console.

3. Service workers require HTTPS, although to facilitate local testing, that policy does not apply to `localhost`. If your production web server does not support HTTPS, then the service worker registration will fail, but the rest of your web app will remain functional.

4. The service worker is only enabled in the production environment, e.g. the output of `npm run build`. It’s recommended that you do not enable an offline-first service worker in a development environment, as it can lead to frustration when previously cached assets are used and do not include the latest changes you’ve made locally.

5. If you *need* to test your offline-first service worker locally, build the application (using `npm run build`) and run a standard http server from your build directory. After running the build script, `create-react-app` will give instructions for one way to test your production build locally and the deployment instructions have instructions for using other methods. *Be sure to always use an incognito window to avoid complications with your browser cache.*

6. By default, the generated service worker file will not intercept or cache any cross-origin traffic, like HTTP API requests, images, or embeds loaded from a different domain. Starting with Create React App 4, this can be customized, as explained above.

Progressive Web App Metadata

The default configuration includes a web app manifest located at `public/manifest.json`, that you can customize with details specific to your web application.

When a user adds a web app to their homescreen using Chrome or Firefox on Android, the metadata in `manifest.json` determines what icons, names, and branding colors to use when the web app is displayed. The Web App Manifest guide provides more context about what each field means, and how your customizations will affect your users' experience.

Progressive web apps that have been added to the homescreen will load faster and work offline when there's an active service worker. That being said, the metadata from the web app manifest will still be used regardless of whether or not you opt-in to service worker registration.