Using Service Workers with create-react-app

Shaumik Daityari Follow
Oct 1 · 6 min read

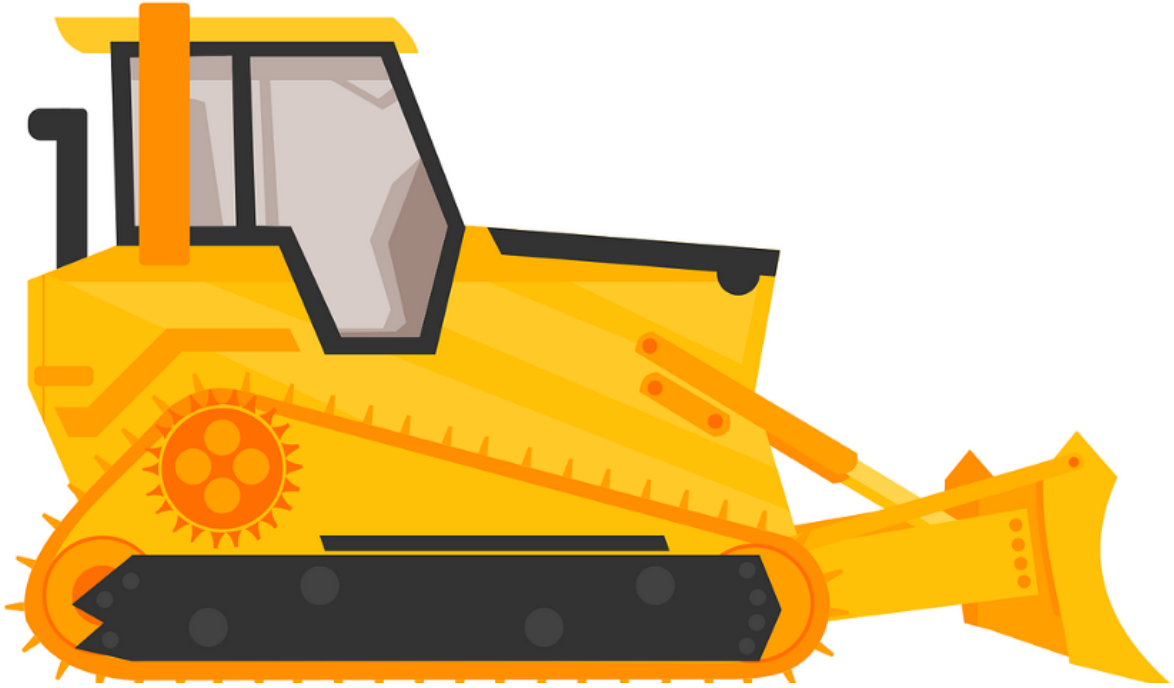


Image by 200 Degrees from Pixabay

If you use React for front end development, chances are that you have heard of service workers. If you are not sure what they do, or how to configure them properly, this beginner's guide to service workers in React should serve as a good first step in creating feature-rich, offline experiences in React.

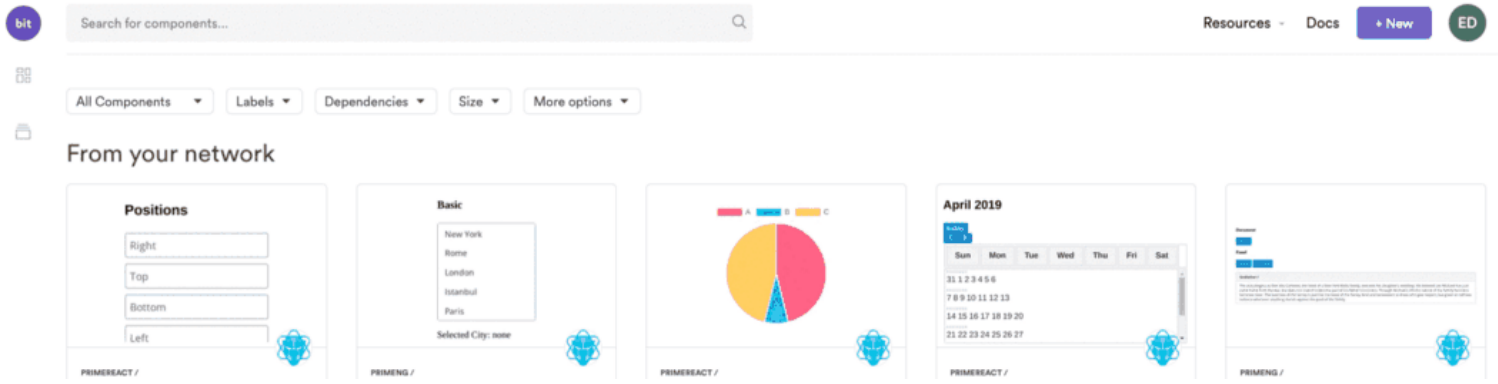
Service workers are scripts that are run by the browser. They do not have any direct relationship with the DOM. They provide many out of the box network-related features. Service workers are the foundation of building an offline experience. They enable features such as push notifications and background sync.

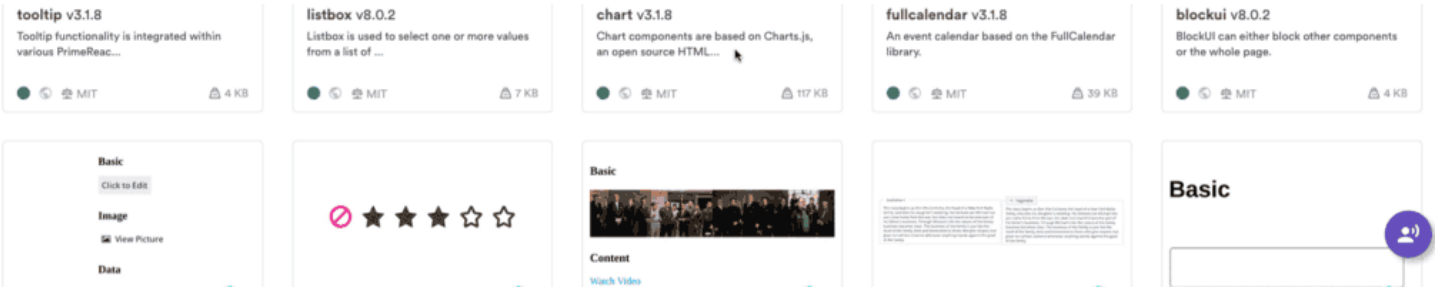
Service workers are scripts that are run by the browser of a client. They do not have any direct relationship with the DOM. They provide many out of the box network-related features. Service workers are the foundation of building an offline experience. They enable features such as push notifications and background sync.

If you develop the ability to activate and properly configure service workers in React, you can utilize endless possibilities by judiciously intercepting and managing network requests. In React, service workers are automatically added when you create your

application through the `create-react-app` command, through `SWPrecacheWebpackPlugin`. Service workers ensure that the network is not a bottleneck to serve new requests.

Let us look at the constituents of a service worker, and then explore how you can configure them to utilize their full potential.





Example: exploring reusable React components shared on Bit.dev

Service Workers: Use Cases

The loss of network is a common issue that developers face in ensuring a seamless connection. Thus, in recent times, the concept of offline applications to ensure superior user experience is gaining popularity. Service workers provide web developers with a lot of benefits:

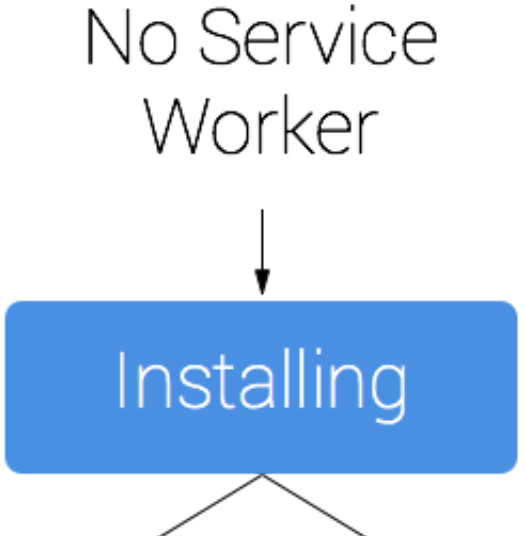
- They improve the performance of your website. Caching key parts of your website only helps in making it load faster.
- Enhances user experience through an offline-first outlook. Even if one loses connectivity, one can continue to use the application normally.
- They enable notification and push APIs, which are not available through traditional web technologies.

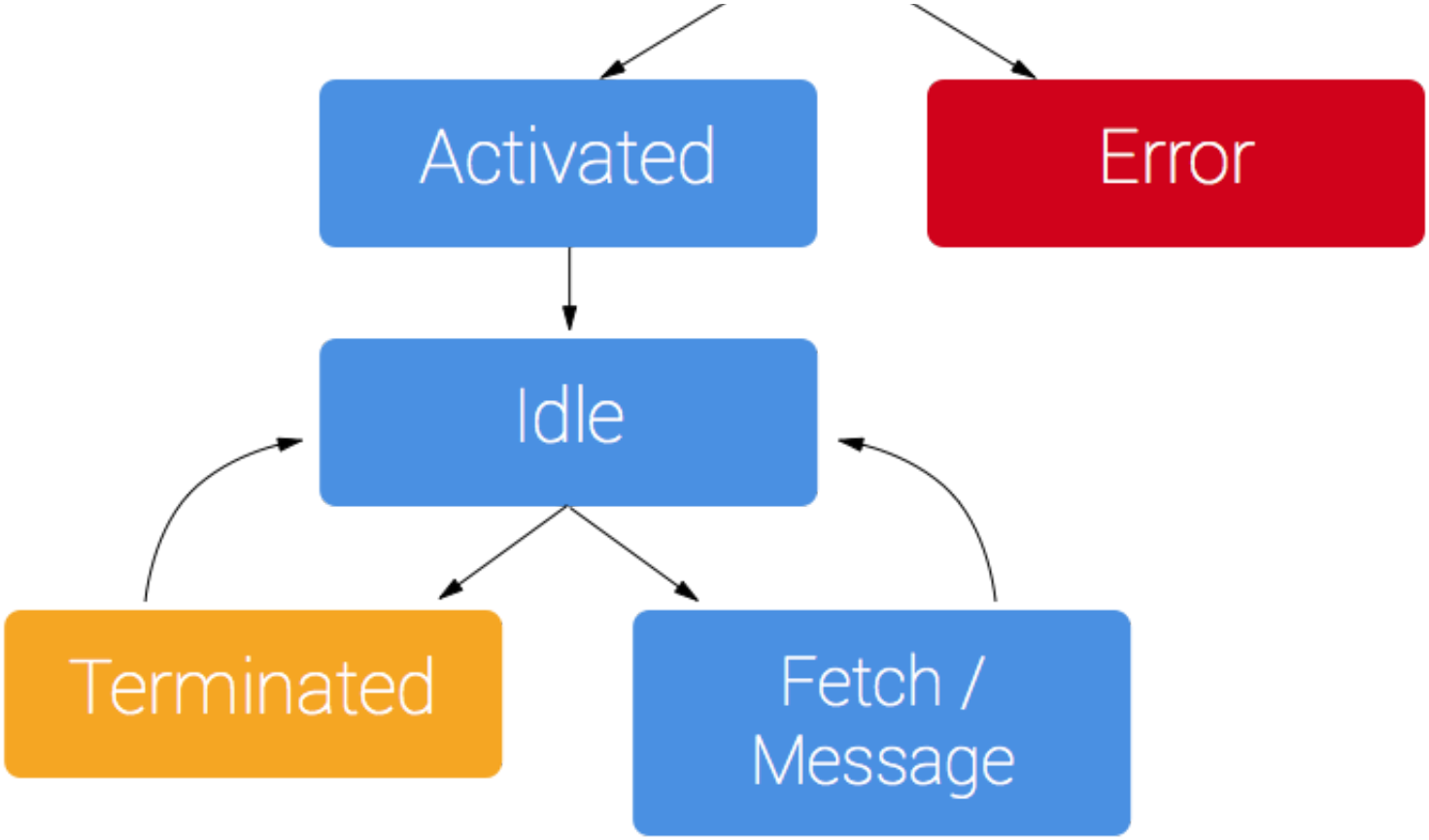
- They enable you to perform background sync. You can defer certain actions until network connectivity is restored to ensure a seamless experience to the user.

Service Workers: Lifecycle

The lifecycle of a service worker is not linked to that of your web application. You install a service worker by registering it using JavaScript. This instructs the browser to begin installing it in the background. This is also the time when you get to cache your required assets. When the installation step is successful, the activation process starts. Once activated, the service worker is associated with any page in its scope. Unless it is invoked by an event, it will be terminated.

The lifecycle of a service worker typically needs to be coded by the developer. In case of service workers in React, the life cycle management is handled by React itself, which makes the process easier for a developer to enable and use service workers.





Service Worker Lifecycle (Source)

React Service Workers: Key Considerations

Before you jump onto the activation and configuration of a React service worker, let us look at the principles and considerations that govern the usage of service workers.

- Service workers are executed by the browser in their own global script context. This means that you do not have direct access to your page's DOM elements. Therefore,

you need an indirect way for service workers to communicate with pages that they are supposed to control. This is handled through the `postMessage` interface.

- Service workers run only on the `HTTPS` protocol. The only exception here is when you run it in localhost.
- They are not tied to a particular page, and therefore, can be reused.
- Service workers are event-driven. This means that service workers can not retain any information once they shut down. In order to access information from earlier states, you need to use the IndexedDB API.

Activate React Service Workers

When you create a React application through the `create-react-app` command, the project layout looks like the structure shown below:

```
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── build
├── public
│   ├── favicon.ico
│   ├── index.html
│   └── manifest.json
└── src
    ├── App.css
    ├── App.js
```

```
├── App.test.js
├── index.css
├── index.js
├── logo.svg
└── serviceWorker.js
```

Notice the `serviceWorker.js` file in the `src` directory. By default, this file is generated when you create a React application.

At this stage, your service worker is not registered, so you will have to first register it before utilizing it in your application.

To register the service worker, navigate to the `src/index.js` file, and look for the following line:

```
serviceWorker.unregister();
```

Change it to the following line.

```
serviceWorker.register();
```

This single line change will now enable you to use service workers in your React application.

In a general web application, you would have to code the whole lifecycle of a service worker. However, React enables service workers by default and allows a developer to directly get into working with them. Navigate to the `src/serviceWorker.js` file and you will notice that the underlying methods of a service worker are present.

Working with React Service Workers in Development Environment

If you explore the function `register()` in the file `serviceWorker.js`, you would notice that by default, it works only in production mode (`process.env.NODE_ENV === 'production'` is set as one of the conditions). There are two workarounds to it.

- You can remove this condition from the function `register()` to enable it in development mode. However, this could potentially lead to some caching issues.
- A cleaner way of enabling service workers is to create a production version of your React app, and then serve it. You can run the following commands to do so:

```
$ yarn global add serve
$ yarn build
$ serve -s build
```

Head over to `localhost:5000` in a browser to check the served application.

Configure a Custom Service Worker to CRA

CRA's default `service-worker.js` caches all static assets. To add any new functionality to your service workers, you need to create a new file `custom-service-worker.js` and then modify the `register()` function to load your custom file.

Around line 34 in the `serviceWorker.js` file, look for the `load()` even listener and add your custom file to it.

```
window.addEventListener('load', () => {
     const swUrl = `${process.env.PUBLIC_URL}/custom-service-
worker.js`;
     ...
}
```

Next, update the `package.json` file as below.

```
"scripts": {
   "start": "react-app-rewired start",
   "build": "react-app-rewired build",
   "test": "react-app-rewired test",
   "eject": "react-app-rewired eject"
},
```

In this step, we will invoke Google's Workbox plugin.

```
npm install --save-dev workbox-build
```

Next, you need to create a config file to instruct CRA to insert our custom service worker.

```
const WorkboxWebpackPlugin = require("workbox-webpack-plugin");
module.exports = function override(config, env) {
  config.plugins = config.plugins.map((plugin) => {
    if (plugin.constructor.name === "GenerateSW") {
      return new WorkboxWebpackPlugin.InjectManifest({
       swSrc: "./src/custom-service-worker.js",
       swDest: "service-worker.js"
      });
    }
    return plugin;
  });
  return config;
};
```

You can then proceed to create the custom service worker to cache a particular directory as shown below.

```
workbox.routing.registerRoute(
  new RegExp("/path/to/cache/directory/"),
  workbox.strategies.NetworkFirst()
);
workbox.precaching.precacheAndRoute(self.__precacheManifest || [])
```

Ensure that you build your application again for the changes to take effect.

Final Thoughts

In this post, we covered what service workers are, why they should be used, and the process of using and configuring service workers in your React application. Service workers form a critical part of modern web development and it is a good idea to incorporate them into your existing React applications.

In case you are interested to explore this further, here is a complete tutorial on creating a progressive web app in React.

Learn More

Building with React for All Platforms: Top Frameworks and Tools

5 recommended frameworks and tools that will help you use React to build for all platforms.

blog.bitsrc.io

React Reactjs JavaScript Web Services Frontend