PWA best practices

1.Introduction

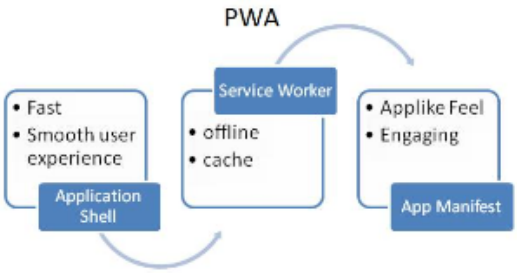
This page describes the best practices for developing Progressive Web Apps (PWA) in generic as well as using ReactJS.

2. What is PWA?

Progressive Web Apps are user experiences that have the reach of the web, and are:

- Reliable- Load instantly even in uncertain network conditions.
- Fast- Respond quickly to user interactions.
- Engaging- Feel like a natural app on the device, with an immersive user experience.

A PWA app will have 3 components to provide different improvements of webapp:



3. Best Practices

3.1 Development

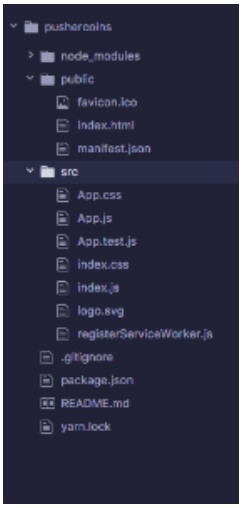
3.1.1 Creating PWA project

Usecreate-react-appto create the project, It creates necessary folder structure and build scripts including PWA service worker and Manifest.

Installation

- `npm install -g create-react-app`
- `create-react-app <PROJECT_NAME>`

Folder structure



3.1.2 Secure Origin

The backend and the development libraries should be secured using HTTPS. Following tips can be used to secure the app :

- Upgrade un secure requests (“HTTP” connections) to “HTTPS” redirecting users as needed.
- Update all links referencing “http://” to “https://”. If you rely on third-party scripts or content, talk to them about making their resources available over HTTPS too.
- Use HTTP Strict Transport Security (HSTS) headers when serving pages. It’s a directive that forces browsers to only talk to your site in HTTPS.

3.1.3 Web Application Manifest

- Userealfavicongenerator.netto generate cross browser favicons.

  Lighthousehave proposedshipping a 192px icon for the home screen icon and a 512px one for your splash screen. Add to Home option should be shown to users to add the app in home screen.

3.1.4 Mobile Friendly

Optimize for multiple devices should include a meta-viewport in the <head> of the document. Create-react-app does include a valid meta-viewport by default and Lighthouse will flag if it is missing.

3.1.4 Push notifications

Interact users more using Push notifications. React service worker can be customized to enable push notification from Firebase or Pusher.

3.2 Performance

3.2.1 Caching Strategy

Static Content

This includesCSS, JavaScript and Font files.*Cache first strategy*will be used, where app fetches first from the cache before requesting the page.

The filenames will have version number appended to reflect the changes as below :

```
'/style.caf603aca47521f652fd678377752dd0.css',

'/main-compiled.9b0987235e60d1a3d1dc571a586ce603.js',
```

More details can be found inhttps://jakearchibald.com/2016/caching-best-practices/.

Dynamic Content

This includes dynamic HTML files.*Network**first strategy*will be used, where app checking network first before cache.

Cashing in reactJs

Following 2 reactJs librarieswere developed to handle two different offline use cases :

sw-precache- to automate precaching of static resources.

sw-toolbox- to handle runtime caching and fallback strategies.

3.2.2 Code-splitting by routes

Use Webpack, which supports code-splitting your app into chunks wherever it notices a require.ensure() being used.React Router also supports a handy 'getComponent' attribute, which is similar to 'component' but is asynchronous and is super nice for getting code-splitting setup quickly.

3.3.3 Server side rendering

React'srenderToString()to render components to an HTML string and injecting state for the application on initial boot up. SSR is synchronous. To improve the rendering furtherHTML Streaming can be used.

3.3.4 HTML Streaming

Browsers can render pages for users before the entire response is completed.

3.3.5The PRPL Performance Pattern

PRPL is a pattern for structuring and serving PWAs, with an emphasis on the performance of app delivery and launch. PRPL stands for:

- **P**ush critical resources for the initial URL route. (HTTP/2 server push)
- **R**ender initial route.
- **P**re-cache remaining routes.
- **L**azy-load and create remaining routes on demand.

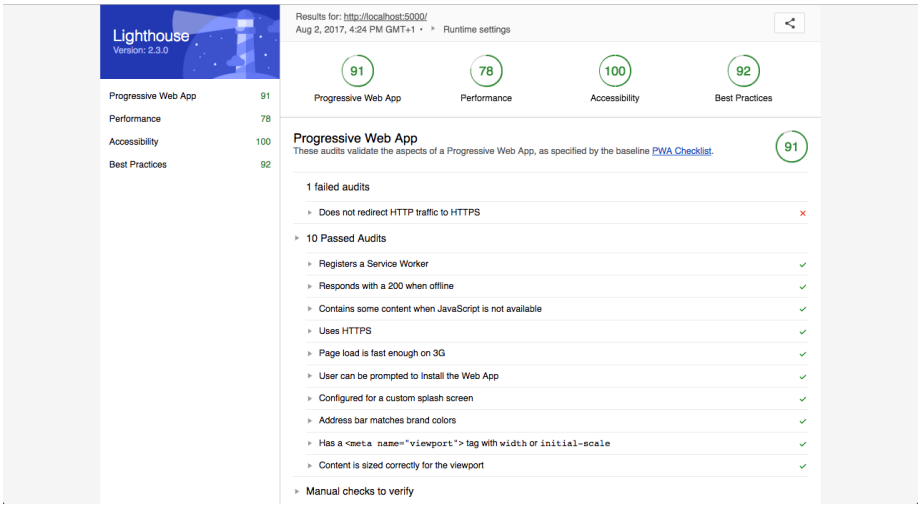
3.3 Tools and Plugins

3.3.1 Progressive Web App Checklist

Google defined a baseline of PWA checklist inhttps://developers.google.com/web/progressive-web-apps/checklist.

3.3.2 Lighthouse tool

Lighthouse is an open-source, automated tool for improving the quality of web pages. You can run it against any web page, public or requiring authentication. It has audits for performance, accessibility, progressive web apps, and more. Above checklist will beautomatically verify by this tool.



Installation

- npm install -g lighthouse
- lighthouse <URL>

OR

- Use Chrome extension (https://chrome.google.com/webstore/detail/lighthouse/blipmdconlkpinefehnmjammfjpmpbjk)

3.3.3 Chrome Devtools

Security Panel

This toolallows you to validate issues with security certificates and mixed content errors.

Application Panel

This tool supports inspecting your Web App Manifest.

3.3.4 CommonsChunkPlugin

Use this webpack plugin to identify common modules used across different routes and put them in a commons chunk.

3.3.5 Webpack community tools

- http://webpack.github.io/analyse
- https://chrisbateman.github.io/webpack-visualizer/
- https://alexkuz.github.io/stellar-webpack/
- https://github.com/danvk/source-map-explorer

4. Limitations & Risk

- Bad cashing policies will make app not refreshing.
- Upgrading service worker code require 24h expiry time.