SWE 363: Web Engineering & Development

Module 11

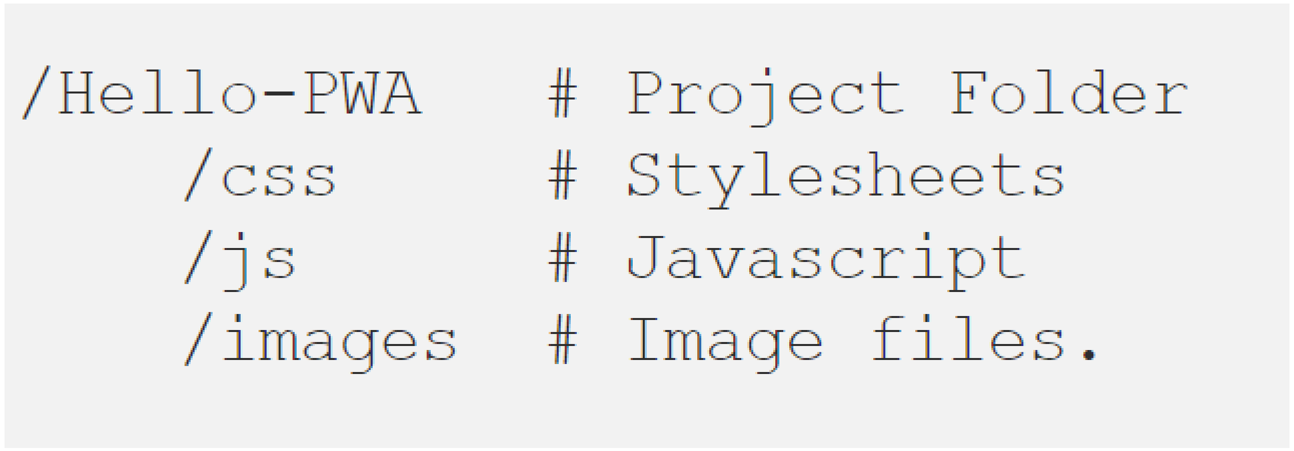
Progressive Web App

References

- https://medium.com/james-johnson/a-simple-progressive-web-app-tutorial-f9708e5f2605
- https://www.youtube.com/watch?v=z2JgN6Ae-Bo

The Setup

- Create a directory for the app and add *js*, *css*, and *images* subdirectories.



```
/Hello-PWA    # Project Folder
    /css      # Stylesheets
    /js       # Javascript
    /images   # Image files.
```

Writing the App Interface

- When writing the markup for a Progressive Web App there are two requirements to keep in mind:
  - The app should display some content even if JavaScript is disabled. This prevents users from seeing a blank page if their internet connection is bad or if they are using an older browser.
  - It should be responsive and display properly on a variety of devices. In other words, it needs to be mobile friendly.
    - By *viewport* tag

Writing the App Interface…

- Create a file named *index.html* in your project root folder and add the following markup:

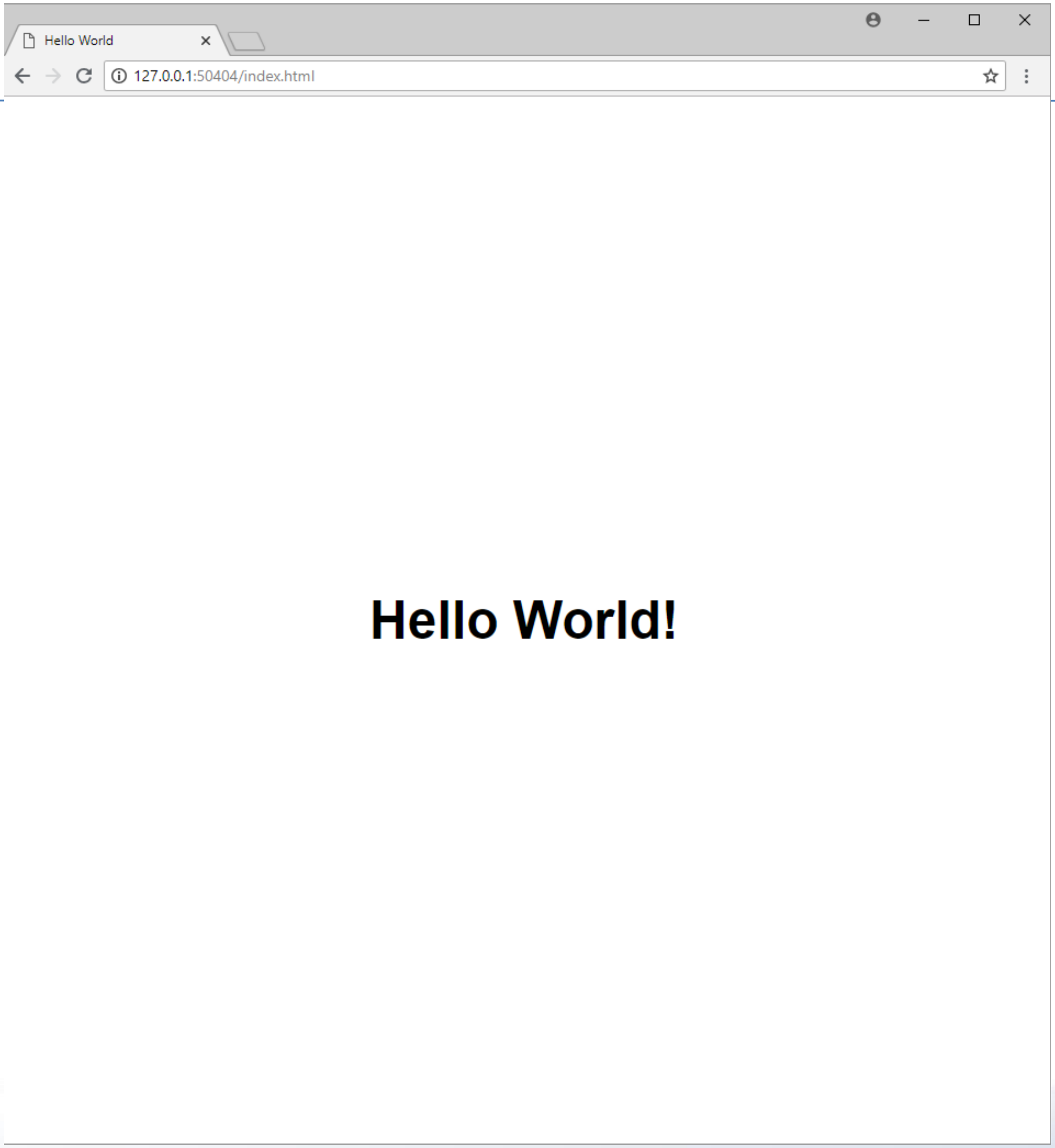
```html
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Hello World</title>
  <link rel="stylesheet" href="css/style.css">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
<body class="fullscreen">
  <div class="container">
    <h1 class="title">Hello World!</h1>
  </div>
</body>
</html>
```

Style.css

- Next, create a file named *style.css* in the *css* folder and add this code:

```css
body {
  font-family: sans-serif;
}

/* Make content area fill the entire browser window */
html,
.fullscreen {
  display: flex;
  height: 100%;
  margin: 0;
  padding: 0;
  width: 100%;
}

/* Center the content in the browser window */
.container {
  margin: auto;
  text-align: center;
}

.title {
  font-size: 3rem;
}
```

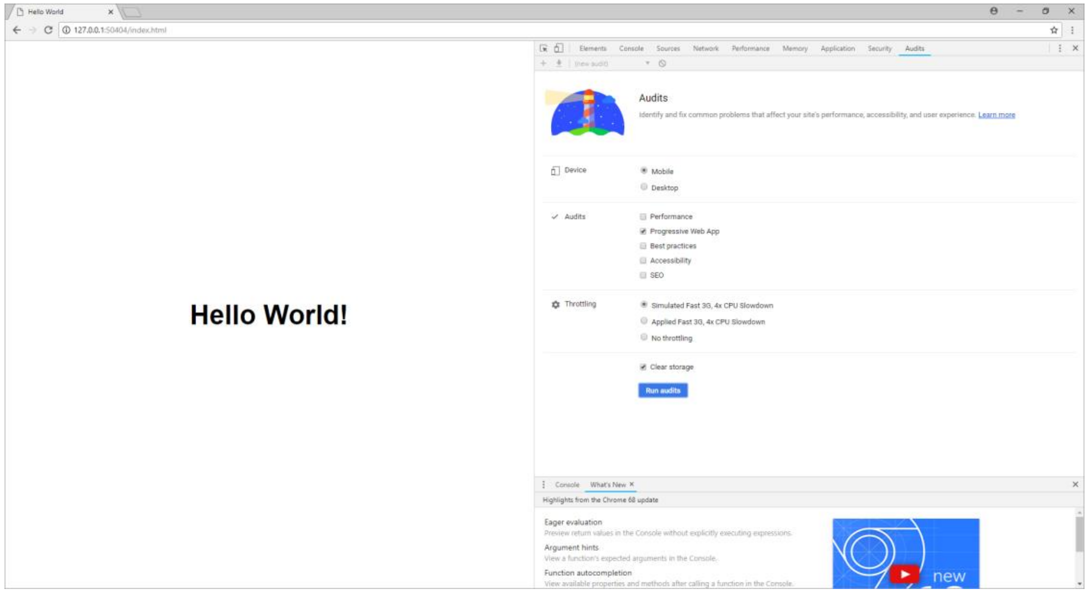
Home page



Hello World!

Testing the App

- ❑ We'll use Google's Lighthouse to test the app and see how well it conforms to PWA standards
- ❑ Press F12 to open the developer panel in Chrome and click on the audits tab to open Lighthouse



Audits

Identify and fix common problems that affect your site's performance, accessibility, and user experience. Learn more

Device: [✓ Mobile] Desktop

Audits: Performance, [✓ Progressive Web App], Best practices, Accessibility, SEO

Throttling: [✓ Simulated Fast 3G, 4x CPU Slowdown] Applied Fast 3G, 4x CPU Slowdown, No throttling

[✓ Clear storage]

Run audits

Console | What's New

Highlights from the Chrome 68 update

Eager evaluation
Preview return values in the Console without explicitly executing expressions.

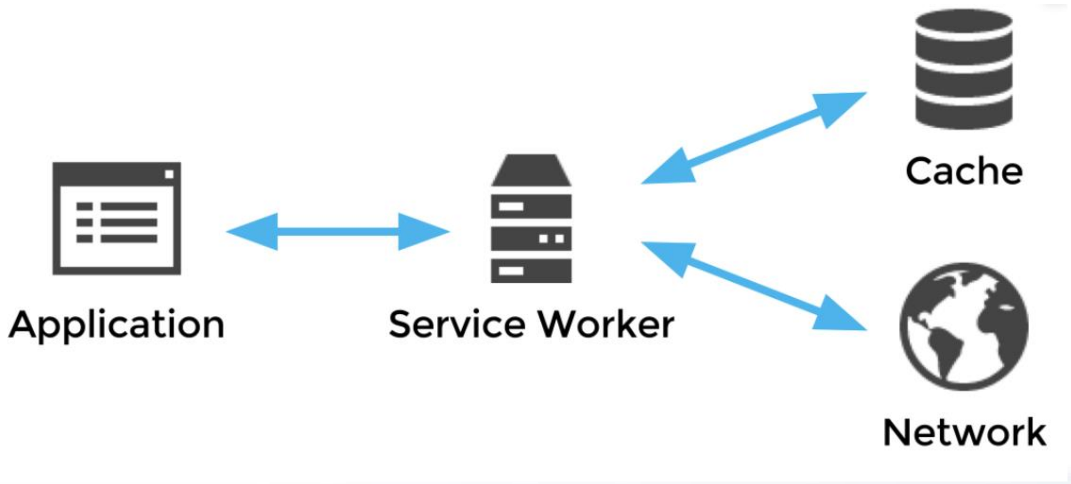
Argument hints
View a function's expected arguments in the Console.

Function autocompletion
View available properties and methods after calling a function in the Console.

- Make sure the "Progressive Web App" option is checked
- Then click on the "run tests" button

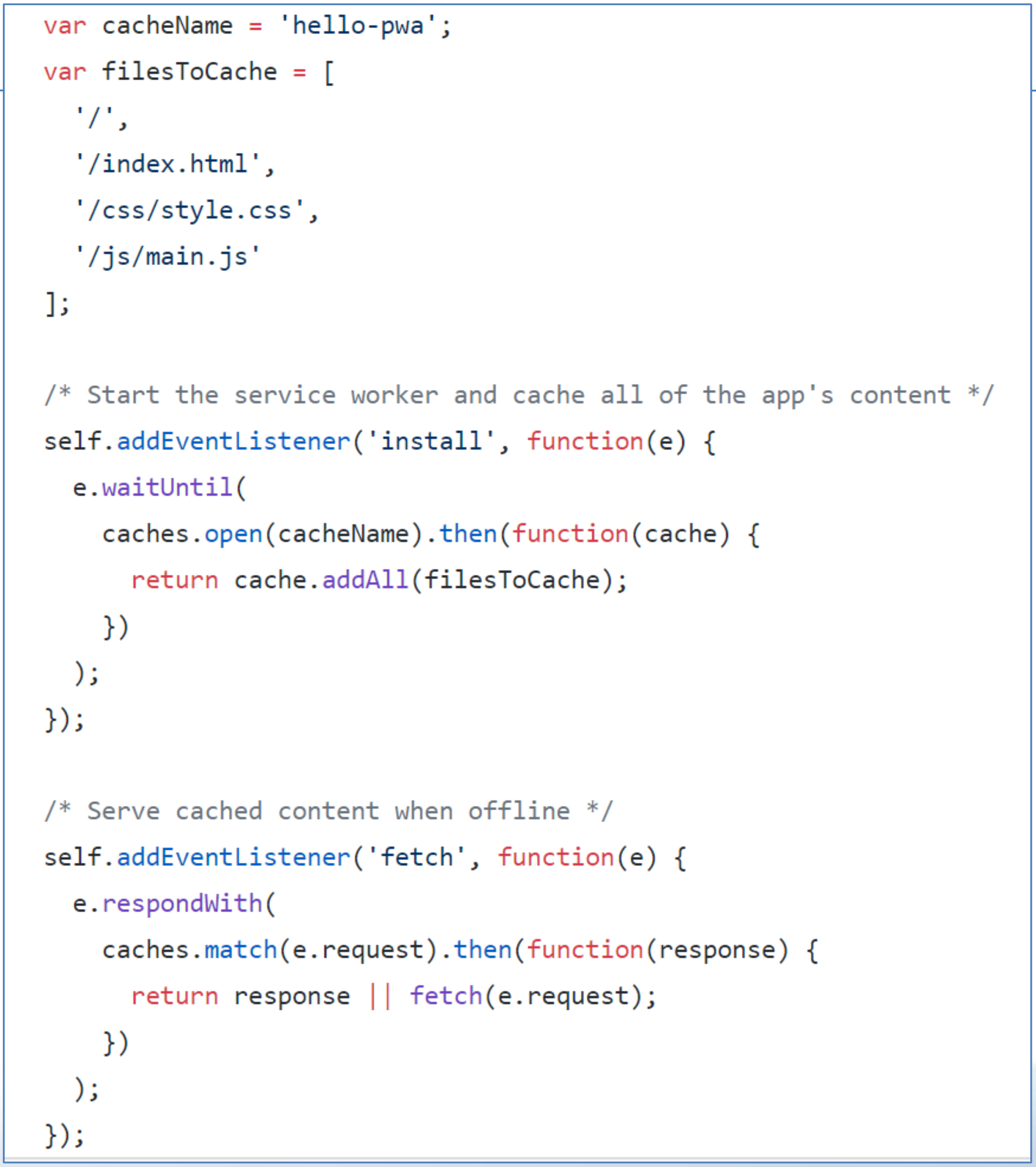
Add a Service Worker

- Service workers are essentially scripts that run in the background to perform tasks that don't require user interaction.
- For our app we'll use one to download and cache our content and then serve it back up from the cache when the user is offline.



Add a Service Worker…

- ❑ Create a file named *sw.js* in your root folder and enter the contents of the script below.



```
var cacheName = 'hello-pwa';
var filesToCache = [
  '/',
  '/index.html',
  '/css/style.css',
  '/js/main.js'
];

/* Start the service worker and cache all of the app's content */
self.addEventListener('install', function(e) {
  e.waitUntil(
    caches.open(cacheName).then(function(cache) {
      return cache.addAll(filesToCache);
    })
  );
});

/* Serve cached content when offline */
self.addEventListener('fetch', function(e) {
  e.respondWith(
    caches.match(e.request).then(function(response) {
      return response || fetch(e.request);
    })
  );
});
```

Add a Service Worker…

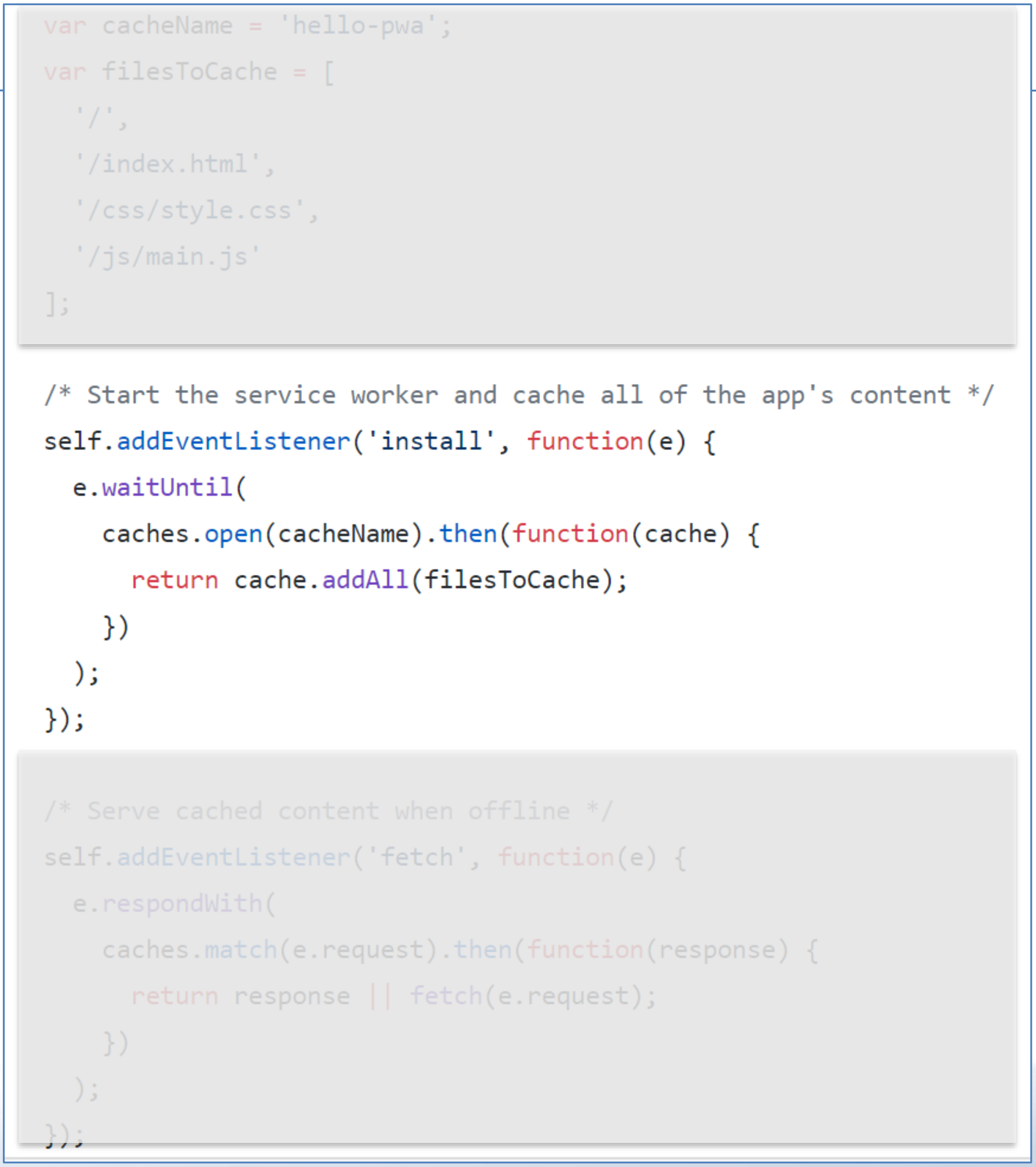
create an offline cache in the browser and give us access to it from Javascript.

filesToCache is an array containing a list of all of the files that need to be cached.

```
var cacheName = 'hello-pwa';
var filesToCache = [
  '/',
  '/index.html',
  '/css/style.css',
  '/js/main.js'
];

/* Start the service worker and cache all of the app's content */
self.addEventListener('install', function(e) {
  e.waitUntil(
    caches.open(cacheName).then(function(cache) {
      return cache.addAll(filesToCache);
    })
  );
});

/* Serve cached content when offline */
self.addEventListener('fetch', function(e) {
  e.respondWith(
    caches.match(e.request).then(function(response) {
      return response || fetch(e.request);
    })
  );
});
```

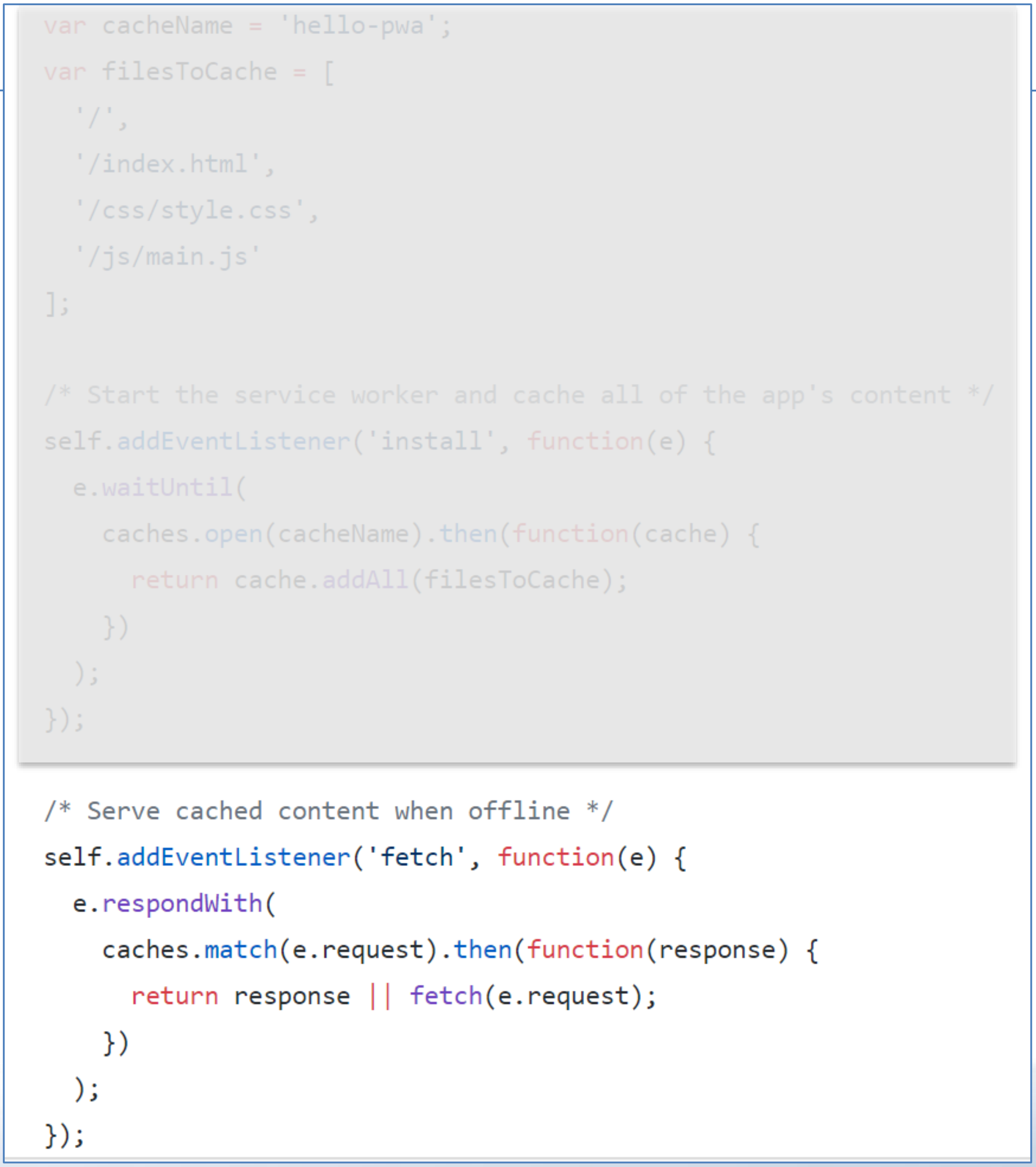
Add a Service Worker…

Function to install the service worker and create the browser cache using *cacheName*.



```
var cacheName = 'hello-pwa';
var filesToCache = [
  '/',
  '/index.html',
  '/css/style.css',
  '/js/main.js'
];

/* Start the service worker and cache all of the app's content */
self.addEventListener('install', function(e) {
  e.waitUntil(
    caches.open(cacheName).then(function(cache) {
      return cache.addAll(filesToCache);
    })
  );
});

/* Serve cached content when offline */
self.addEventListener('fetch', function(e) {
  e.respondWith(
    caches.match(e.request).then(function(response) {
      return response || fetch(e.request);
    })
  );
});
```

Add a Service Worker…

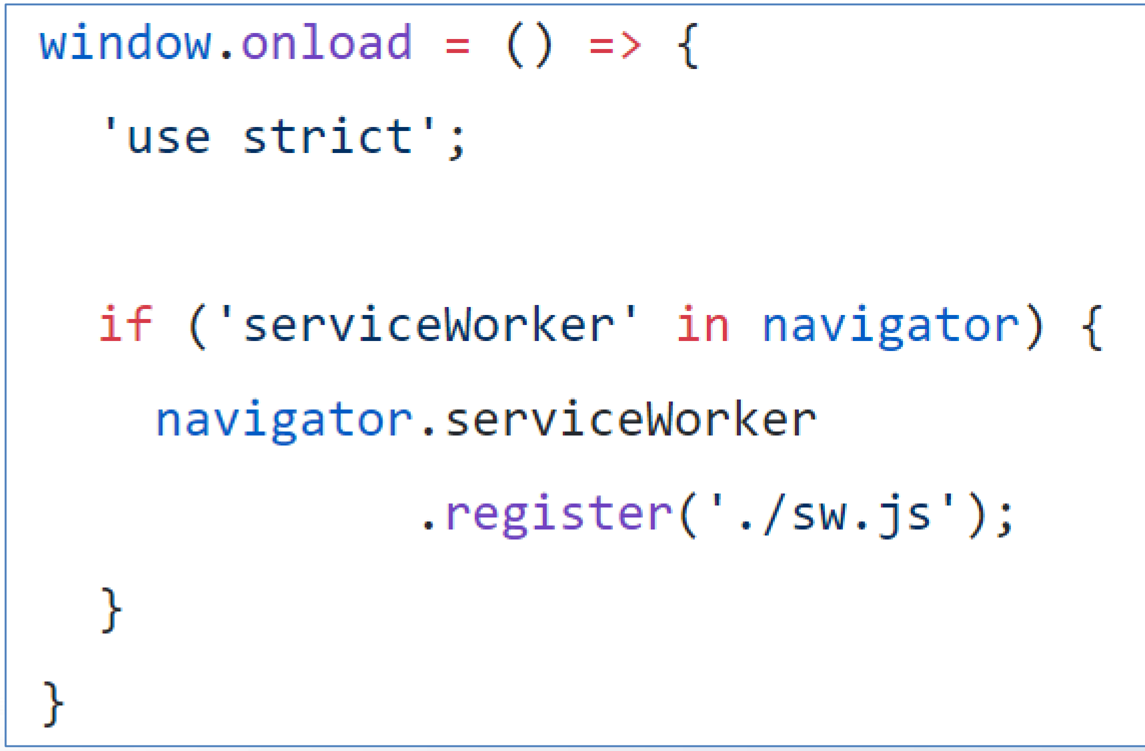


```
var cacheName = 'hello-pwa';
var filesToCache = [
  '/',
  '/index.html',
  '/css/style.css',
  '/js/main.js'
];

/* Start the service worker and cache all of the app's content */
self.addEventListener('install', function(e) {
  e.waitUntil(
    caches.open(cacheName).then(function(cache) {
      return cache.addAll(filesToCache);
    })
  );
});

/* Serve cached content when offline */
self.addEventListener('fetch', function(e) {
  e.respondWith(
    caches.match(e.request).then(function(response) {
      return response || fetch(e.request);
    })
  );
});
```

Function to load in the cached files when the browser is offline

Register Service worker

- Now after the service worker script is created, we need to register it with our app.
- Create a file named *main.js* in the *js* folder and enter the following code:



```
window.onload = () => {
  'use strict';

  if ('serviceWorker' in navigator) {
    navigator.serviceWorker
             .register('./sw.js');
  }
}
```

- This code simply loads up the service worker script and gets it started.

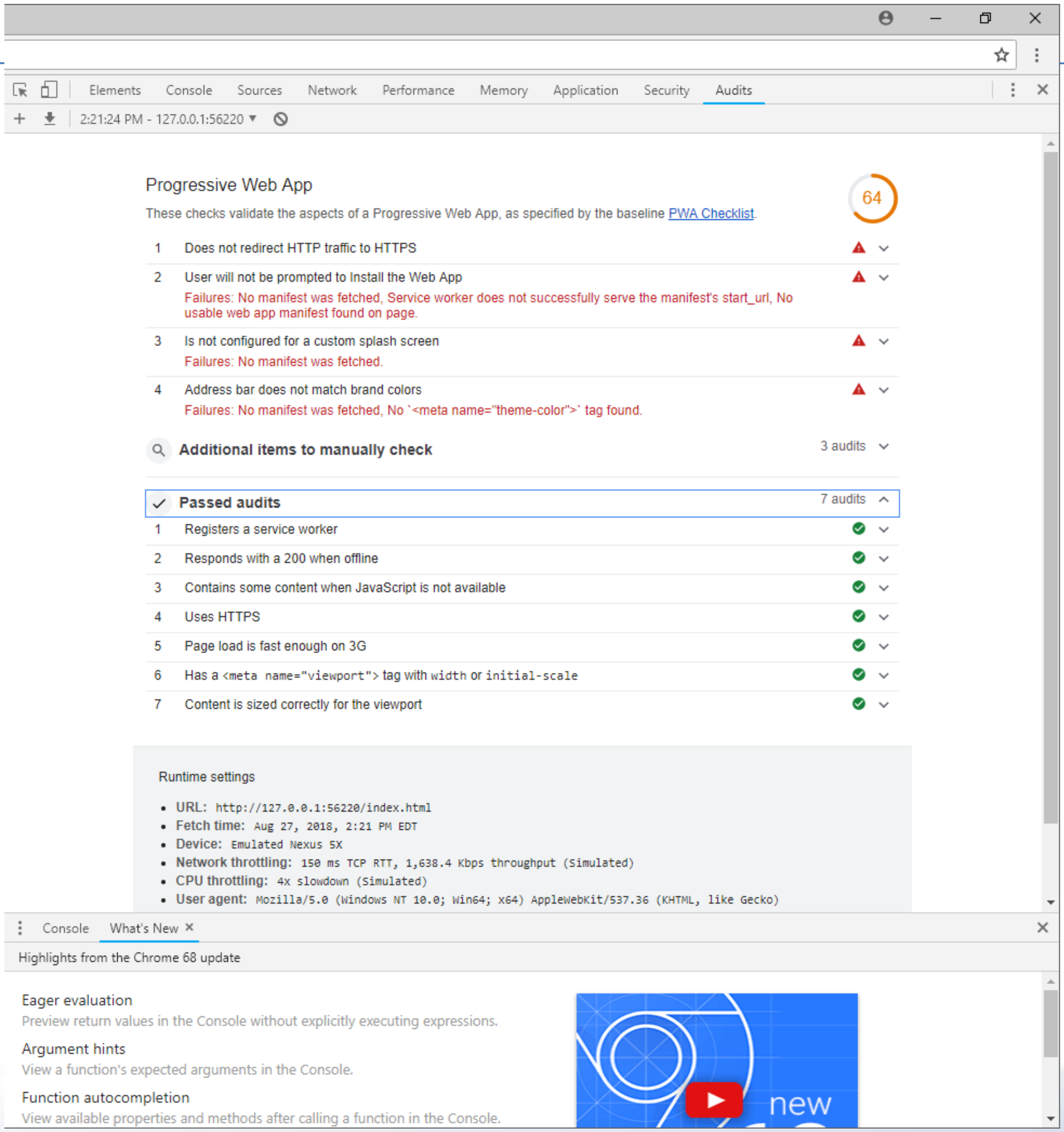
Register Service worker

- ❑ Add the code to your app by including the script just before the closing </body> tag in index.html

```
   <script src="js/main.js"></script>
</body>
```

Checking Lighthouse audits

- If you run the Lighthouse audits now your score should go up to 64 and the service worker requirement will pass.



Elements Console Sources Network Performance Memory Application Security Audits

2:21:24 PM - 127.0.0.1:56220

Progressive Web App

64

These checks validate the aspects of a Progressive Web App, as specified by the baseline PWA Checklist.

1 Does not redirect HTTP traffic to HTTPS

2 User will not be prompted to Install the Web App
Failures: No manifest was fetched, Service worker does not successfully serve the manifest's start_url, No usable web app manifest found on page.

3 Is not configured for a custom splash screen
Failures: No manifest was fetched.

4 Address bar does not match brand colors
Failures: No manifest was fetched, No `<meta name="theme-color">` tag found.

Additional items to manually check — 3 audits

Passed audits — 7 audits

1 Registers a service worker

2 Responds with a 200 when offline

3 Contains some content when JavaScript is not available

4 Uses HTTPS

5 Page load is fast enough on 3G

6 Has a `<meta name="viewport">` tag with `width` or `initial-scale`

7 Content is sized correctly for the viewport

Runtime settings

- URL: `http://127.0.0.1:56220/index.html`
- Fetch time: `Aug 27, 2018, 2:21 PM EDT`
- Device: `Emulated Nexus 5X`
- Network throttling: `150 ms TCP RTT, 1,638.4 Kbps throughput (Simulated)`
- CPU throttling: `4x slowdown (Simulated)`
- User agent: `Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)`

Console What's New

Highlights from the Chrome 68 update

Eager evaluation
Preview return values in the Console without explicitly executing expressions.

Argument hints
View a function's expected arguments in the Console.

Function autocompletion
View available properties and methods after calling a function in the Console.

Add a Manifest

- ❑ The final requirement for a PWA is to have a manifest file.
- ❑ The manifest is a *json* file that is used to specify how the app will look and behave on devices.
  - ▪ For example, you can set the app's orientation and theme color
- ❑ Save a file named *manifest.json* in your root folder and add the following content:

```
{
  "name": "Hello World",
  "short_name": "Hello",
  "lang": "en-US",
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "white",
  "theme_color": "white"
}
```

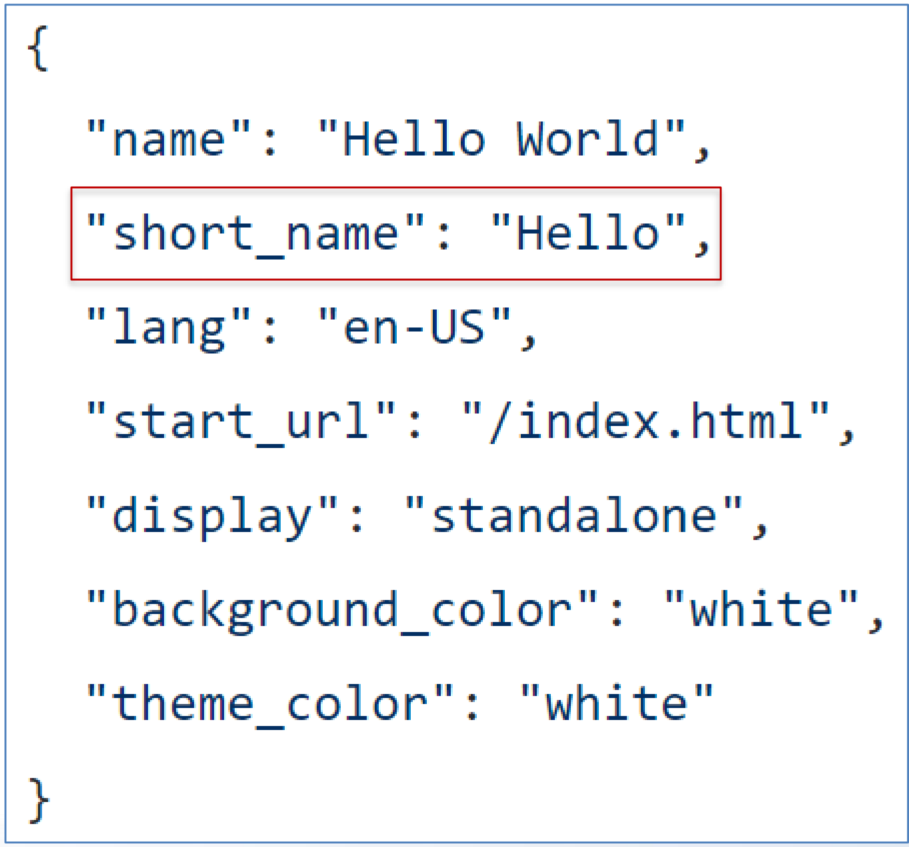
Add a Manifest…

The title of the app. This is used when prompting the user to install the app. It should be the full title of the app.

```
{
  "name": "Hello World",
  "short_name": "Hello",
  "lang": "en-US",
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "white",
  "theme_color": "white"
}
```

Add a Manifest…

Is the name off the app as it will appear on the app icon. This should be short and to the point.



```
{
   "name": "Hello World",
   "short_name": "Hello",
   "lang": "en-US",
   "start_url": "/index.html",
   "display": "standalone",
   "background_color": "white",
   "theme_color": "white"
}
```

Add a Manifest…

Tells the browser which page to load up when the app is launched.

```
{
  "name": "Hello World",
  "short_name": "Hello",
  "lang": "en-US",
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "white",
  "theme_color": "white"
}
```

Add a Manifest…

```
{
  "name": "Hello World",
  "short_name": "Hello",
  "lang": "en-US",
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "white",
  "theme_color": "white"
}
```

The type of shell the app should appear in. For our app, we are using *standalone* to make it look and feel like a standard native app.

Add a Manifest…

The color of the splash screen that opens when the app launches.

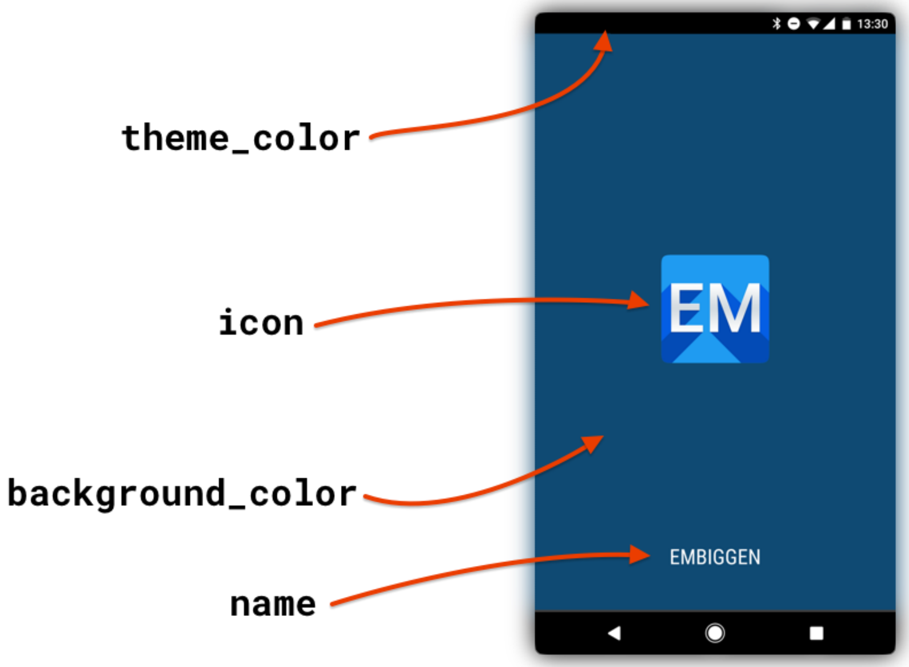
```
{
   "name": "Hello World",
   "short_name": "Hello",
   "lang": "en-US",
   "start_url": "/index.html",
   "display": "standalone",
   "background_color": "white",
   "theme_color": "white"
}
```

Add a Manifest…

```
{
  "name": "Hello World",
  "short_name": "Hello",
  "lang": "en-US",
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "white",
  "theme_color": "white"
}
```

Sets the color of the tool bar and in the task switcher.

Add a Manifest…



```
{
   "name": "Hello World",
   "short_name": "Hello",
   "lang": "en-US",
   "start_url": "/index.html",
   "display": "standalone",
   "background_color": "white",
   "theme_color": "white"
}
```

Add a Manifest…

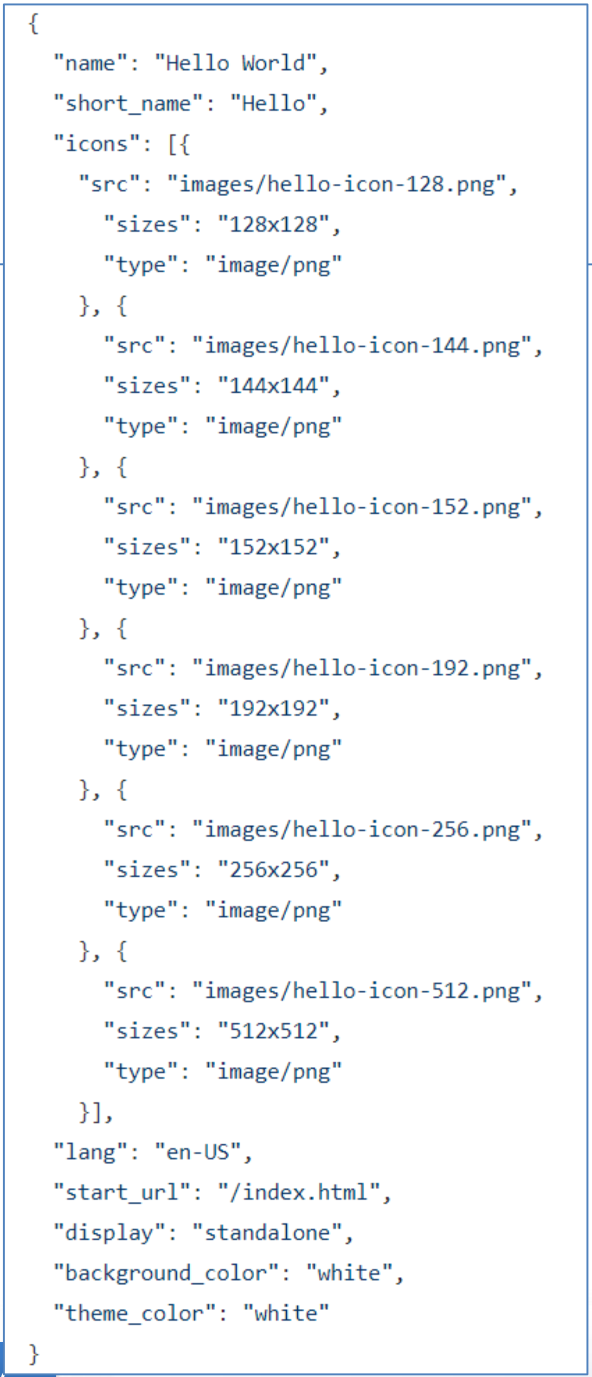
- To add the manifest to your app, link to it inside the *index.html* head tag like this:

```
<head>
...
<link rel="manifest" href="/manifest.json">
...
</head>
```

- You should also declare the theme color to match the one set in your manifest by adding a meta tag inside the head:

<meta name="theme-color" content="white"/>

App Icons



```json
{
  "name": "Hello World",
  "short_name": "Hello",
  "icons": [{
    "src": "images/hello-icon-128.png",
      "sizes": "128x128",
      "type": "image/png"
    }, {
      "src": "images/hello-icon-144.png",
      "sizes": "144x144",
      "type": "image/png"
    }, {
      "src": "images/hello-icon-152.png",
      "sizes": "152x152",
      "type": "image/png"
    }, {
      "src": "images/hello-icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    }, {
      "src": "images/hello-icon-256.png",
      "sizes": "256x256",
      "type": "image/png"
    }, {
      "src": "images/hello-icon-512.png",
      "sizes": "512x512",
      "type": "image/png"
    }],
  "lang": "en-US",
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "white",
  "theme_color": "white"
}
```

- ❑ To add an icon for the app, you'll need an app icon that's been sized for the browser, Windows, Mac/iPhone and Android.
- ❑ That's a minimum of 7 different sizes: 128x128 px, 144x144 px, 152x152 px, 192x192 px, 256x256 px, 512x512px and a 16x16px favicon.
- ❑ Create icons and store them in *images* folder and place *favicon.ico* in the project root folder

Finishing Up

- Finally, add them to index.html in the head tag:

```
<head>
...
<link rel="icon" href="favicon.ico" type="image/x-icon" />
<link rel="apple-touch-icon" href="images/hello-icon-152.png">
<meta name="theme-color" content="white"/>
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="black">
<meta name="apple-mobile-web-app-title" content="Hello World">
<meta name="msapplication-TileImage" content="images/hello-icon-
144.png">
<meta name="msapplication-TileColor" content="#FFFFFF">
...
</head>
```

- The final requirement for a Progressive Web App is that it must be served via *https*.

- You can also get the full source code to this example on Github:
- https://github.com/jamesjohnson280/hello-pwa