Implement a Progressive Web App on Salesforce B2C Commerce

@salesforcedocs

This information was developed in collaboration with representatives from Google, LLC.

CONTENTS

WHAT'S A PROGRESSIVE WEB APP? 4

PWA ADVANTAGES 4

FAST 4

Cache Resources by Leveraging Service Worker 4

Prefetch Next Page Assets to Make Page Transitions Instant 5

INSTALLABLE 7

Prompt Android Customers to Add the Website to Their Home Screen 7

Prompt iOS Customers to Add the Website to Their Home screen 9

Prompt Desktop Customers to Add the Website to Their Home Screen 10

RELIABLE 11

Create a Page Accessible When the Customer is Offline 11

Make the Pages Previously Browsed Available When the Customer is Offline 12

Show the Customer Relevant Information when Offline 13

ENGAGING 14

Notify the Customers about Updates 15

Let Customers Know when the Price of an Item from their Wishlist Drops 15

Notify the Customers about their Abandoned Cart 15

HOW DO I IMPLEMENT THE PWA ON B2C COMMERCE? 16

Multi-Page Progressive Web App 16

Transform an SFRA-Based Website into a PWA 16

Serve over HTTPS 16

Register a Service Worker 16

Web App Manifest 18

Conclusion 21

Single-Page Progressive Web App on B2C 22

Introduction 22

Building a Custom Single-Page PWA on Salesforce B2C Commerce 22

Back-End Code 22

Front-End Code 24

Conclusion 25

Single-Page Progressive Web App for Headless 26

Front-End Code 26

Conclusion 28

Conclusion: Progressive Web App on B2C Commerce 29

RESOURCES 29

WHAT'S A PROGRESSIVE WEB APP?

A Progressive Web App (PWA) is a website that users experience like an app they would find in an app store, for example, Apple Store, or Google Play Store.

Since the PWA is a website, users can access it using a conventional URL with HTTPS, making it available from search engines.

With features such as push notifications, wake, refresh, and instant loading, a PWA provides an optimized web experience for any device.

Instead of developing native apps for multiple operating systems, ecommerce merchants can implement a PWA on top of an existing storefront.

The Progressive Web App allows you to:

- Deliver web pages faster.
- Add a website to the homescreen of a device as a native app.
- Increase reliability, by enabling pages when a customer is offline.
- Engage customers with native-app communications, for example, push notifications.

PWA ADVANTAGES

B2C Commerce provides you with all the elements needed to start building a PWA. In this document, we review each advantage of a PWA and look at implementation use cases.

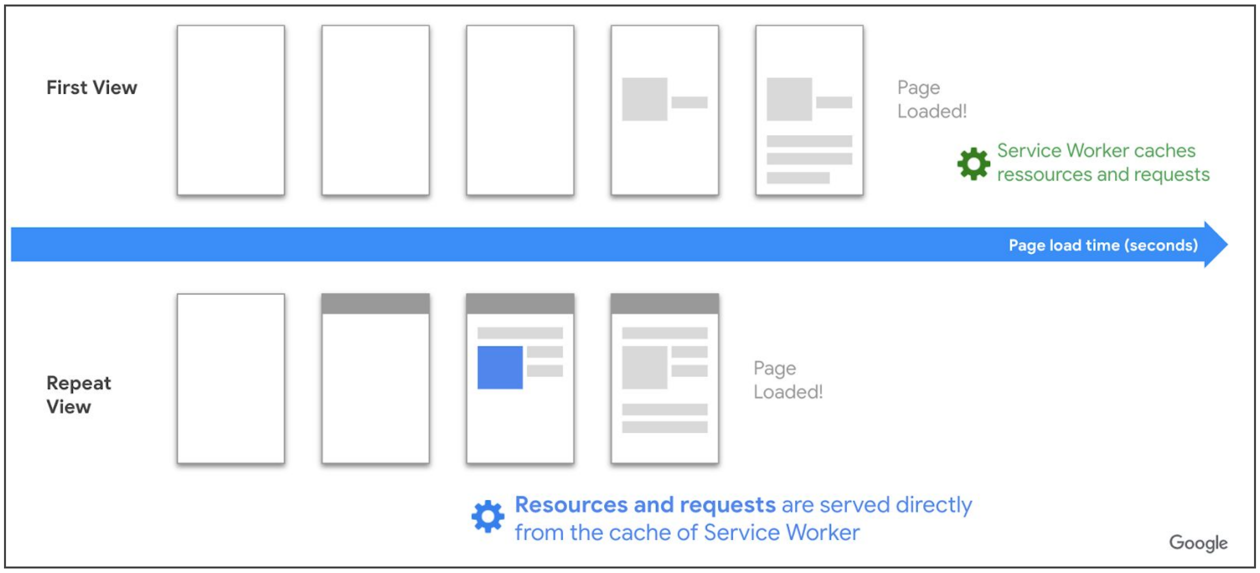
FAST

As indicated by the PWA checklist from Google, performance plays a significant role in the success of any online experience. High performing sites engage and retain customers better than poorly performing ones. Sites should focus on optimizing for user-centric performance metrics.

In fact, as page load times go from one second to ten seconds, the probability of a user bouncing increases by 123%.

Cache Resources by Leveraging Service Worker

Service-worker caching can help you improve your visitors' browsing experience. Service-worker caching lets you implement and control the caching strategy in a more agile way than browser caching. Websites using modern, front-end frameworks such as single-page applications also see a positive effect on loading time with service workers, since cached resources and requests are fetched and served directly from the cache instead of being reloaded from the network.


First View
Page Loaded!
Service Worker caches ressources and requests
Page load time (seconds)
Repeat View
Page Loaded!
Resources and requests are served directly from the cache of Service Worker
Google

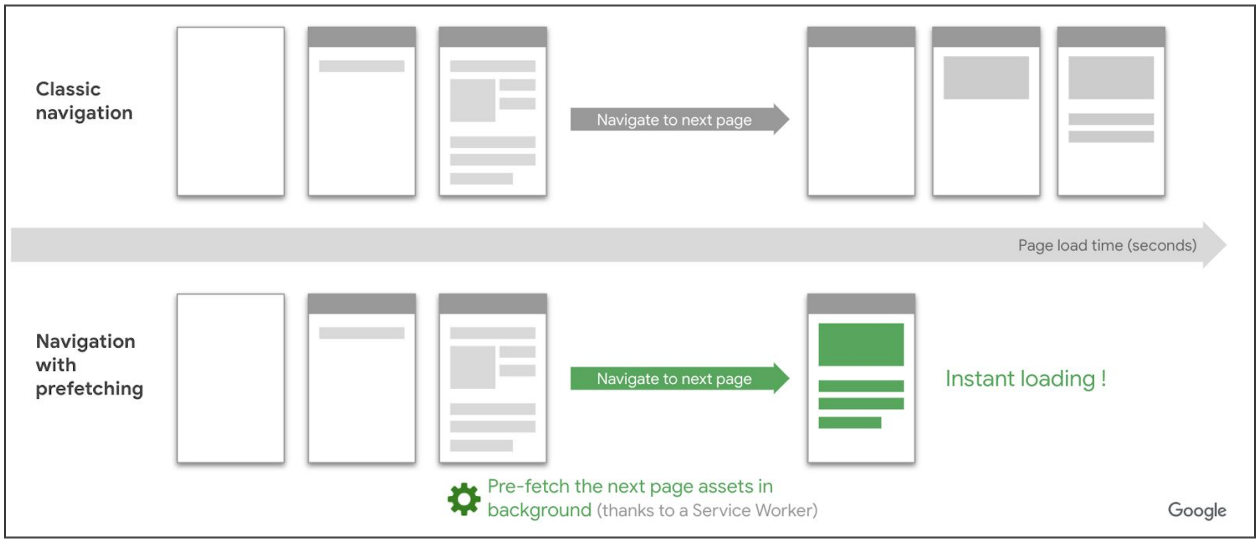
There are various caching strategies you can implement:

- **Common resources**: Cache static resources such as JS, CSS, fonts, and .svg.
- **Above-the-fold items**: Cache content in the Above-the-Fold for faster rendering, and update the cache in the background.
- **Heavy content**: Cache content that may require some time to download, for example, videos or PDF.
- **Appshell**: Components for site layout, for example, headers, footers, and sidebar.
- **JSON responses from Ajax requests**: Store the JSON response from the AJAX requests made on the first load and fetch them from the service worker for the next ones.

Important: Be careful with the storage of time-sensitive information such as prices, discounts, temporary offers, or inventory.

Prefetch Next Page Assets to Make Page Transitions Instant

Use a service worker to make the transitions to the next screens faster. Prefetch the assets for the next pages in the user journey.



Based on navigation patterns you observe in your analytics, you can anticipate which page most users will visit next and prefetch its assets to make the loading instant.
For example:

- From the homepage to the product listing page
- From the product listing page to the product details page
- From the product details page to the cart page

You can prefetch the assets of the pages for which links appear in the viewport of the user's device, using a library such as Quicklink, and cache them with your service worker for more persistent storage.

Note: Quicklink uses the requestIdleCallback method to make sure that such tasks are performed when the main thread is idle, in order to have no impact on the performance of the currently browsed page.

To measure the impact of prefetching on your site speed:

- Implement prefetching on your conversion flow with the higher traffic. For example, Home page > Category page > Product page
- Go to WebPageTest.org and measure your conversion flow with a script such as the following:

```
setEventName    Go to HomePage
navigate    https://example.fr
sleep 5
setEventName    Display Category Page
navigate    https://example.fr/category
```

- Run a test with and without the prefetching (by blocking your service worker file on one of the two tests), and compare the total speed of the user journey. Use the "block sw.js" instruction.

INSTALLABLE

Letting your users add your web app to their device home screen is a great way to provide users with a great app-like experience. Once installed, your PWA sits on your customers' devices' home screens next to native apps, and your users browse your web apps in full screen. The PWA is also installable as an application on desktops by default.

Most PWAs have a better conversion rate and better engagement from users browsing their service from the homescreen.

The basic requirements to make a PWA installable are:

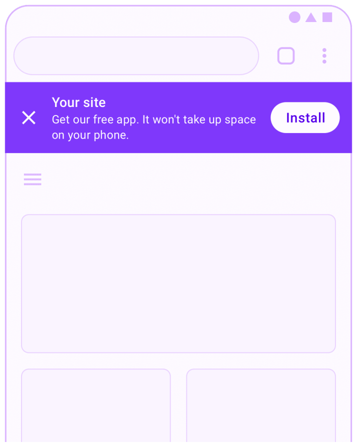
- Serving over HTTPS
- Registering a service worker with a functional `fetch` handler and a web app manifest and icons

You can easily generate a manifest.json file here.

Prompt Android Customers to Add the Website to Their Home Screen

In order to maximize your install rate, make sure you promote the PWA installation at the appropriate point in the customer journey, and with the right messaging. The following is a list of a few patterns you can use.

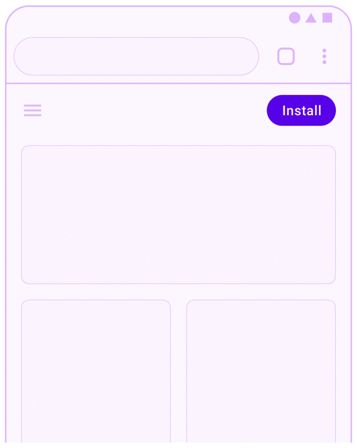
- **Install Banner:**



Most users have encountered installation banners in mobile experiences and are familiar with the interactions offered by a banner. Make sure to:

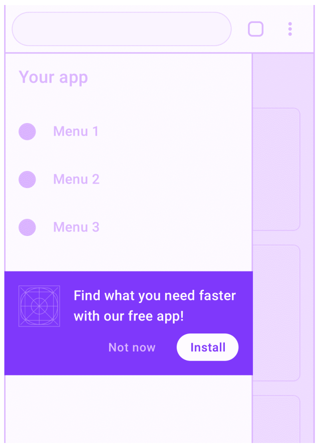
1. Wait until the user has demonstrated interest in your site before showing a banner. For example, wait until they have visited a few pages or interacted with your website in a meaningful way.
2. Provide a brief explanation of the value of installing your PWA in the banner. For example, differentiate the install of a PWA from a native app by mentioning that the PWA uses almost no storage on the user's device or that it will install instantly without a store redirect.

- **Fixed Header:**



When used appropriately, promoting PWA installation from the header of your site is a great way to enable your most loyal customers to easily return to your experience. Pixels in your PWA header are precious, so make sure your installation call to action is appropriately sized, of greater importance than other possible header content, and unobtrusive.

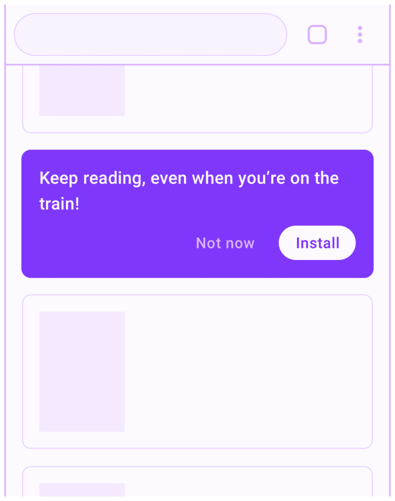
- **Navigation Menu:**



The navigation menu is a great place for promoting the installation of your app since users who open the menu are signaling engagement with your experience. Make sure you:

1. Avoid disrupting important navigational content. Put the PWA install promotion below other menu items.
2. Offer a short, relevant pitch for why the user would benefit from installing your PWA.

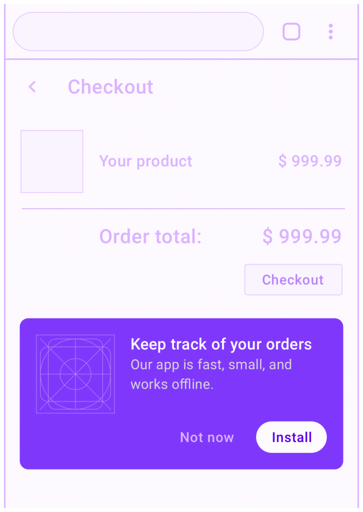
- **Product list:**



You can use an in-feed install promotion between products in a list page for your PWA. Show users how to access the content they're enjoying more conveniently. Focus on promoting features and functionality that is helpful to your users. Make sure you:

1. Limit the frequency of the promotions.
2. Give your customers the ability to dismiss the promotions.
3. Remember your customer's choice to dismiss.

- **Checkout:**



Showing an install promotion after a sequential journey such as a checkout flow is typical. If you're displaying the promotion after the user has completed the journey, you can often make it more prominent. Make sure you:

1. Include a relevant call to action. Which users will benefit from installing your app and why? How is it relevant to the journey they are currently undertaking?
2. If your brand has unique offers for installed app users, mention them here.
3. Keep the promotion out of the way of next steps in your journey so that you don't negatively affect your journey completion rates.

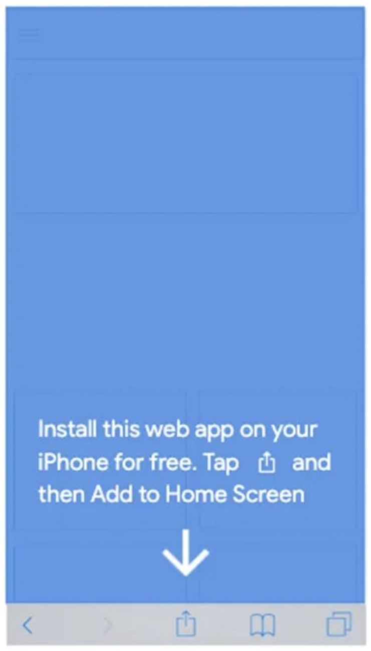
To measure the impact of Installability on your business:

1. Measure users that access your site from the shortcut installed on home screen by adding the utm parameter to the “start_url” property in the web manifest file.
2. In Analytics, create a view to compare conversion rate from people coming from the shortcut vs. others.

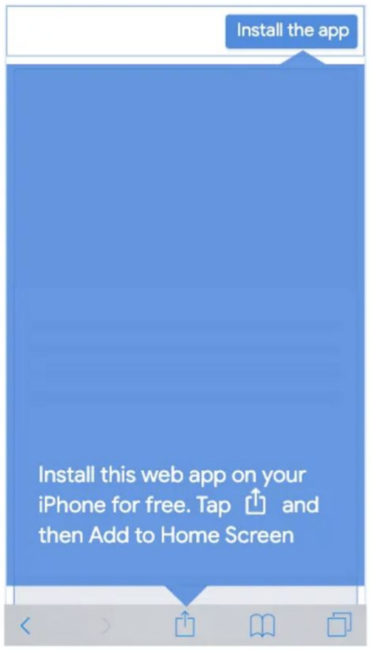
Prompt iOS Customers to Add the Website to Their Home screen

You can use the same installation patterns for your iOS Safari users as for your Android users.

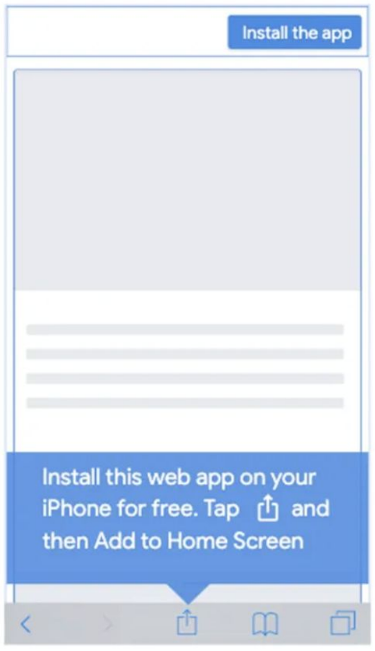
The only difference is that iOS requires users to install from the browser menu. You can guide users through this process with a second install prompt that explains what to do.


Install this web app on your iPhone for free. Tap and then Add to Home Screen

❌ *Pop-ups and overlays usually get ignored and dismissed*


Install the app
Install this web app on your iPhone for free. Tap and then Add to Home Screen

✔ *Native look and feel of the prompt creates a less intrusive experience*


Install the app
Install this web app on your iPhone for free. Tap and then Add to Home Screen

✔ *Native look and feel of the prompt creates a less intrusive experience*

When iOS Safari users add Progressive Web Apps (PWAs) to their home screens, the icon that appears is called the Apple touch icon. You can specify what icon your app should use by including a `<link rel="apple-touch-icon" href="/example.png">` tag in the `<head>` of your page.

If your page doesn't have this link tag, iOS generates an icon by taking a screenshot of the page content. In other words, instructing iOS to download an icon results in a more polished user experience.

To provide a good user experience, make sure that:

- The icon is 180x180 pixels or 192x192 pixels.
- The specified path to the icon is valid.
- The background of the icon is not transparent.

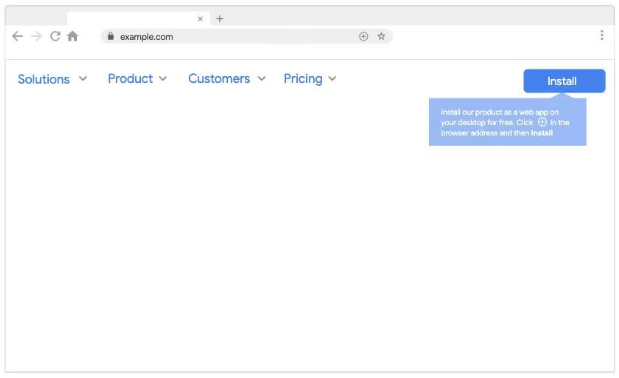
To measure the impact of Installability on your business, you can reuse the following code:

```
// Detects if device is on iOS
const isIos = () => {
  const userAgent = window.navigator.userAgent.toLowerCase();
  return /iphone|ipad|ipod/.test(userAgent);
}
// Detects if device is in standalone mode
const isInStandaloneMode = () => ('standalone' in window.navigator) &&
(window.navigator.standalone);

// Checks if should display install popup notification:
if (isIos() && !isInStandaloneMode()) {
  // The PWA NOT LAUNCHED FROM THE HOME SCREEN
}
```

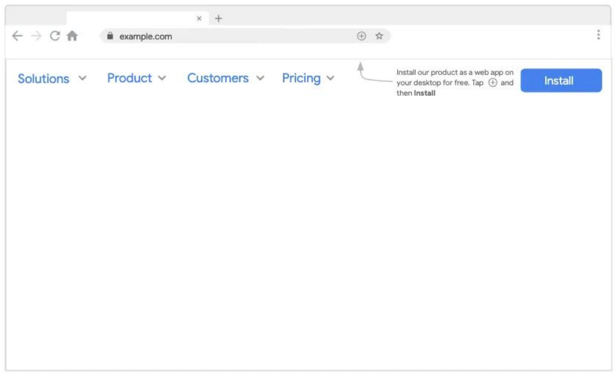
Prompt Desktop Customers to Add the Website to Their Home Screen

On the desktop, PWAs can be installed in Chrome and Edge browsers on all operating systems. A "+" icon appears in the omnibox when the website is installable and wiggles to draw users' attention. Once installed, desktop PWAs can be launched from the app tray into a standalone experience.

Consider providing UI guidance to make sure users are aware that they can install the PWA, such as in the following example:



Install our product as a web app on your desktop for free. Click ⊕ in the browser address and then **Install**



Install our product as a web app on your desktop for free. Tap ⊕ and then **Install**

RELIABLE

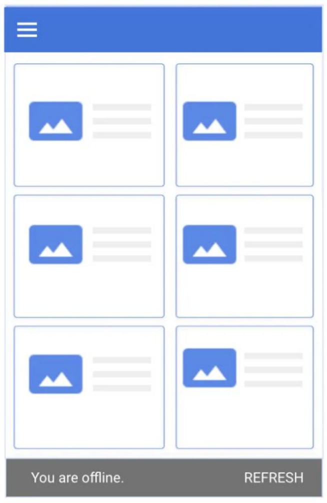
Web users expect a great and reliable experience, even when their device has little to no connection. Providing an offline experience on your e-shop also requires a UX that is consistent with what they find on native apps.

An offline experience on your website gives you the ability to maintain user engagement when their connection is poor to keep them on the path to purchase and conversion.
The following are a few use cases you can consider.

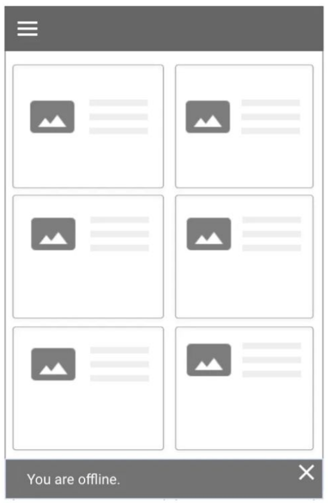
Create a Page Accessible When the Customer is Offline

The first step in developing the offline experience can be to create:

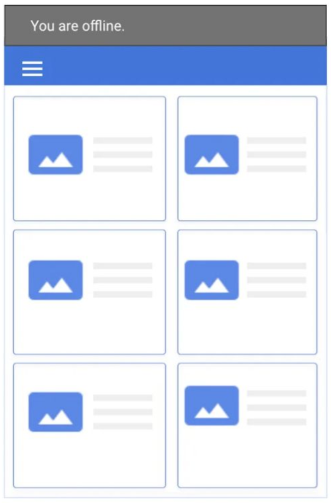
- A simple UI component to inform the user about the change of state (pop up toast / snackbar) to display on any page.



Toast on the bottom of the page with a REFRESH call-to-action

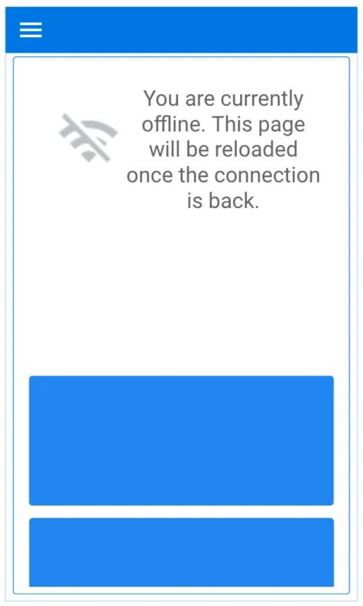


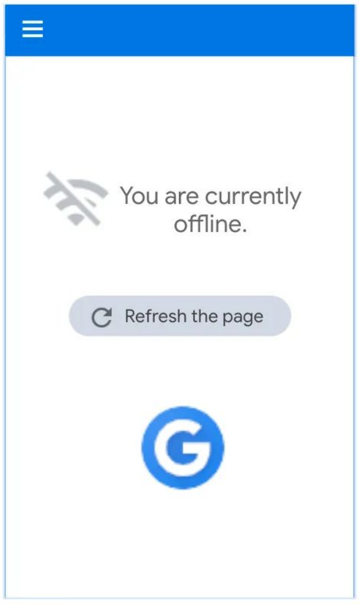
Fade the toast out or let users dismiss it if there is another signal of the ongoing offline state active e.g. greyed out color



Toast placed above the navigation bar

- A simple content page that responds ‘200’ when the network is offline.

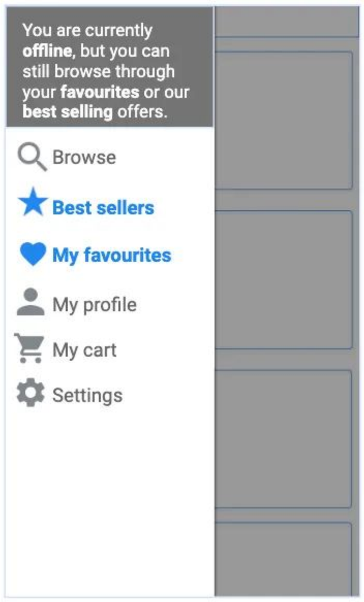

You are currently offline. This page will be reloaded once the connection is back.

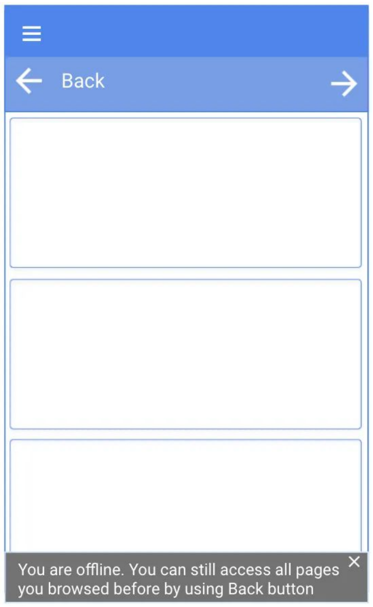


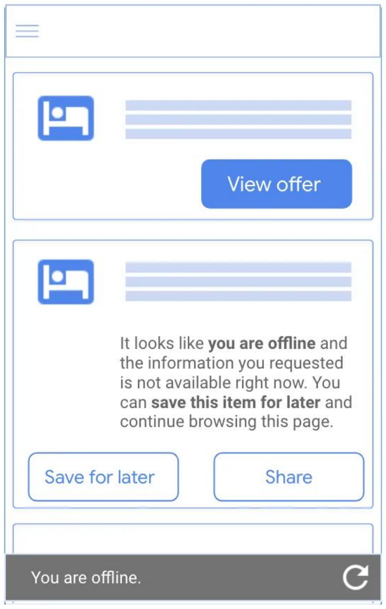
Make the Pages Previously Browsed Available When the Customer is Offline

Create offline states UI, and gray connectivity-sensitive information. Then, you can also let users browse through the pages previously viewed in their current session. This experience provides them with useful content so that they can browse and maintain their engagement on the site.

You can notify your users that although they are offline, they can keep browsing the site with the following UI patterns.


You are currently **offline**, but you can still browse through your **favourites** or our **best selling** offers.


You are offline. You can still access all pages you browsed before by using Back button

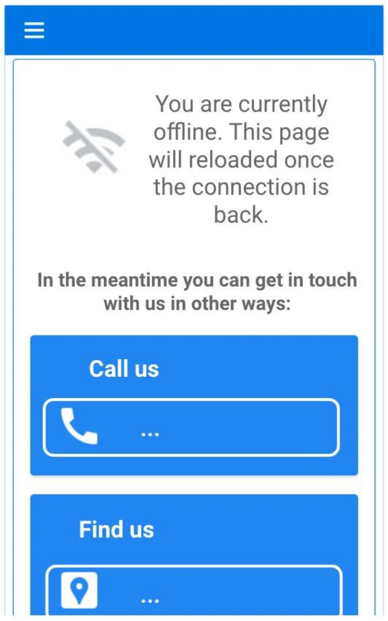

It looks like **you are offline** and the information you requested is not available right now. You can **save this item for later** and continue browsing this page.

You are offline.

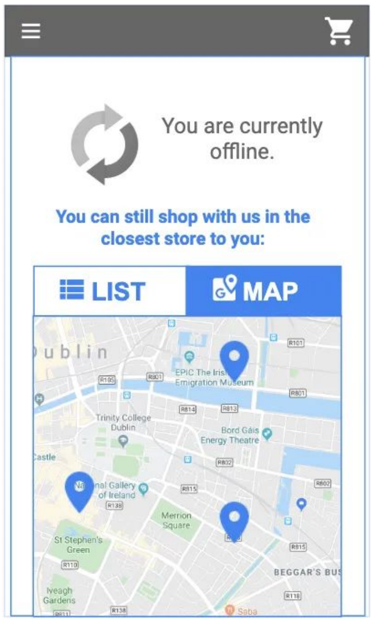
Show the Customer Relevant Information when Offline

You bring value to your customers if you present them relevant information whenever they are experiencing an offline situation. The following are a few examples of great offline experiences during which users can retrieve relevant information or complete significative tasks with the content on the screen.

- Get in touch with the Customer Service or go to the nearest point of sale:

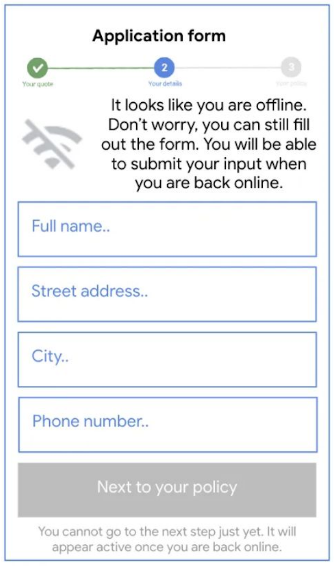

You are currently offline. This page will reloaded once the connection is back.

In the meantime you can get in touch with us in other ways:


You are currently offline.

You can still shop with us in the closest store to you:

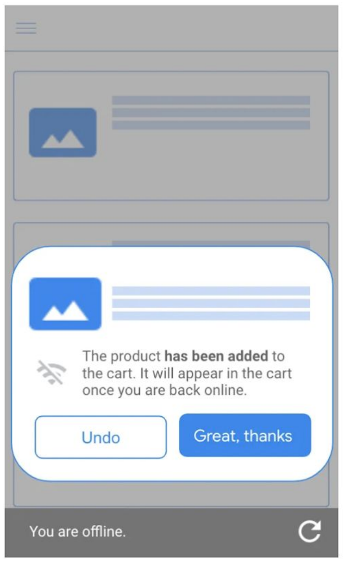
- Complete a form, add to wishlist or cart:


Application form

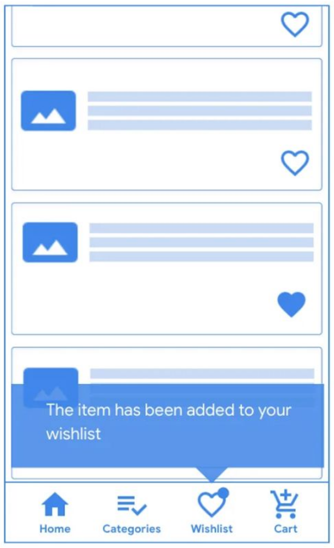
It looks like you are offline. Don't worry, you can still fill out the form. You will be able to submit your input when you are back online.

You cannot go to the next step just yet. It will appear active once you are back online.

Fill out the form


The product **has been added** to the cart. It will appear in the cart once you are back online.

Add to cart


The item has been added to your wishlist

Save to wishlist

Be cautious with time-sensitive information on your e-shop that could be affected by your offline experience, for example, prices, discounts, or stock. Key actions like 'add to cart' should be available only if you have enough stock in your inventory.

Measure offline usage by recording analytics pings when offline, and then replaying them when connection comes back, as described here. For Google Analytics this approach is available in a module for the Workbox library. All analytics events are tracked, even when the user goes offline.

It may be useful to add an explicit tracking for when the user goes offline or online, by using the corresponding browser events. This approach allows measurement of how many users went offline and could be recovered, and how much time is spent offline by calculating the time between online and offline events and sending this information to your analytics. Keep in mind however, that there will be a great deal of offline activity only if there is a good offline experience.

ENGAGING

PWAs support push notifications on Android, with or without the user installing. When and how you ask your users for their consent for push notifications is key. Asking users to sign up for notifications when they first arrive often results in low acceptance rates. Websites that wait until users understand the context and see benefit in receiving notifications before prompting for the permission see great opt-in rates and use notifications to drive business value. Notifications increase repeat visits and consequently sales and conversions.

Notify the Customers about Updates

The completion of an online purchase is a great time to ask users for their permission for web notifications to send them order updates, for example updates about delivery. Make sure your UI component is positioned correctly on the page, after critical transactional information.

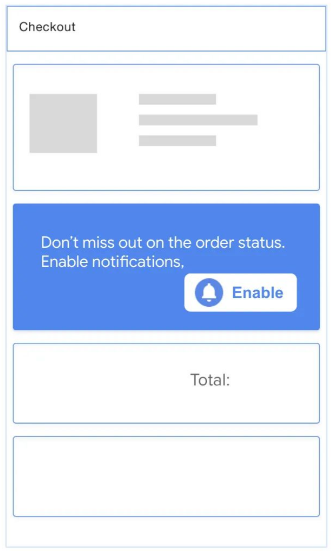
Let Customers Know when the Price of an Item from their Wishlist Drops

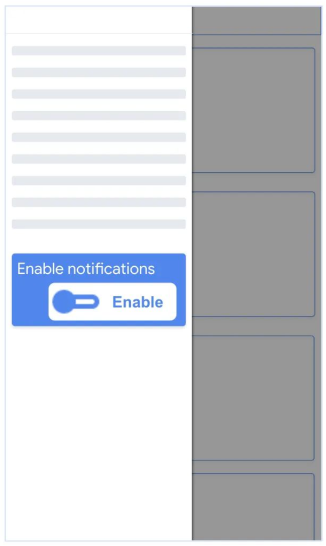
Asking users on the wishlist page for permission to let them know about price drops on their favorite items can also be a good strategy to maximize the opt-in rate.

Notify the Customers about their Abandoned Cart

Notifications are also helpful for e-shops willing to re-engage users who left items in their basket without completing a purchase.

The following are a few UI patterns you can reuse.





-

Implementing push notifications on your PWA requires 1) a push notification service such as Firebase or OneSignal to send messages to your subscribed users and 2) a service worker to:

- Subscribe to a push service using push API.
- Listen to the push events coming from the push service (even if Chrome is closed on the device).
- Show the notification to the user through the notification API.

In order to measure the business impact of notifications:

1. Check in your Notifications Service the open rate of your notifications.
2. Add a utm parameter to your notification links to track notifications inside analytics and see the conversion rate of sessions coming from notifications.

HOW DO I IMPLEMENT THE PWA ON B2C COMMERCE?

Progressive Web Apps enable a great user experience. This section explains options for providing PWA experiences on a B2C Commerce ecommerce site.

Multi-Page Progressive Web App

Hosting: Salesforce B2C Commerce

Here is an example: https://github.com/SalesforceCommerceCloud/plugin_serviceworker

Transform an SFRA-Based Website into a PWA

You can improve an SFRA-based website to fulfill the PWA requirements.

Serve over HTTPS

You don't have to take any action to server over HTTPS. B2C Commerce instances are accessible via HTTPS.

Register a Service Worker

To register a service worker, add some Javascript lines, preferably in ***async*** mode, at the bottom of the document, just before the closing ***</body>*** tag.

```
var swRegistration = null;
if ('serviceWorker' in navigator) {
   window.addEventListener('load', function () {
       navigator.serviceWorker.register('/sw.js').then(function (registration) {
           swRegistration = registration;
       }, function (err) {
           // Do something...
       });
   });
}
```

For example, you can add this code in a specific ***pwa.js*** file and insert it in the SFRA-based website with hook `'app.template.afterFooter'`.

In ***hooks.json***:

```
{
   "hooks": [
       {
           "name": "app.template.afterFooter",
           "script": "./cartridge/scripts/hooks/afterfooter.js"
       }
   ]
}
```

In ***afterfooter.js***:

```
var ISML = require('dw/template/ISML');
/**
* Implements the html after footer hook to declare pwa.js client
*/
function afterFooter() {
   ISML.renderTemplate('afterfooter-hook');
}
module.exports = {
   afterFooter: afterFooter
};
```

The template ***afterfooter-hook.isml***:

```
<script src="${URLUtils.staticURL('/js/pwa.js')}" async></script>
```

Finally, add the registered ***sw.js*** file in the root of the SFRA-based website. You can use a small trick here, to serve the file ***sw.js*** as a root file. The idea is to create a template to render the file and to redirect requests for ***sw.js*** to this file.

The file ***sw.isml***:

```
// Listen event on fetch
self.addEventListener('fetch', function(event) {
   // Do something, like read in the cache or whatever you want
});
```

You can add all our service-worker events here.

In ***RedirectURL.js***:

```
'use strict';

var server = require('server');
server.extend(module.superModule);

server.prepend('Start', function (req, res, next) {
   var URLRedirectMgr = require('dw/web/URLRedirectMgr');
   var Template = require('dw/util/Template');

   var origin = URLRedirectMgr.redirectOrigin;

   if (origin.indexOf('sw.js') > -1) {
       var publicJS = new Template('sw').render().text;
       response.setContentType('text/javascript');
       response.writer.print(publicJS);
   } else {
       next();
   }
});

module.exports = server.exports();
```

The service worker is registered and ready.

Web App Manifest

To add the web app manifest, insert a ***manifest.json*** file in the ***<head>*** tag.

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

For example, you can insert the file in the SFRA-based website with hook `'app.template.htmlHead'`

In ***hooks.json***:

```
{
   "hooks": [
       {
           "name": "app.template.htmlHead",
           "script": "./cartridge/scripts/hooks/htmlhead.js"
       },
       {
           "name": "app.template.afterFooter",
           "script": "./cartridge/scripts/hooks/afterfooter.js"
       }
   ]
}
```

In ***htmlhead.js***:

```
var ISML = require('dw/template/ISML');

/**
* Implements the html head hook to declare manifest.json and register the service
worker
*/
function htmlHead() {
   ISML.renderTemplate('htmlhead-hook');
}

module.exports = {
   htmlHead: htmlHead
};
```

In template ***htmlhead-hook.isml***:

```
<meta name="theme-color" content="#ffffff" />
<link rel="manifest" href="/manifest.json" />
```

Now add the ***manifest.json*** file in the root of the SFRA-based website. To do so, you can use the same trick as for the ***sw.js*** file.

Create the JSON in a ***manifest.js*** script:

```
var URLUtils = require('dw/web/URLUtils');
var manifest = {
   short_name: 'PWA SFRA',
   name: 'My awesome PWA SFRA-based website',
   icons: [
       {
           src: URLUtils.staticURL('/pwaIcon.png').toString(),
           type: 'image/png',
           sizes: '512x512'
       }
   ],
   start_url: URLUtils.url('Home-Show').toString(),
   background_color: '#FFFFFF',
   scope : '/',
   display: 'standalone',
   theme_color: '#000000'
};

module.exports = manifest;
```

Serve the manifest file as JSON in ***RedirectURL.js:***

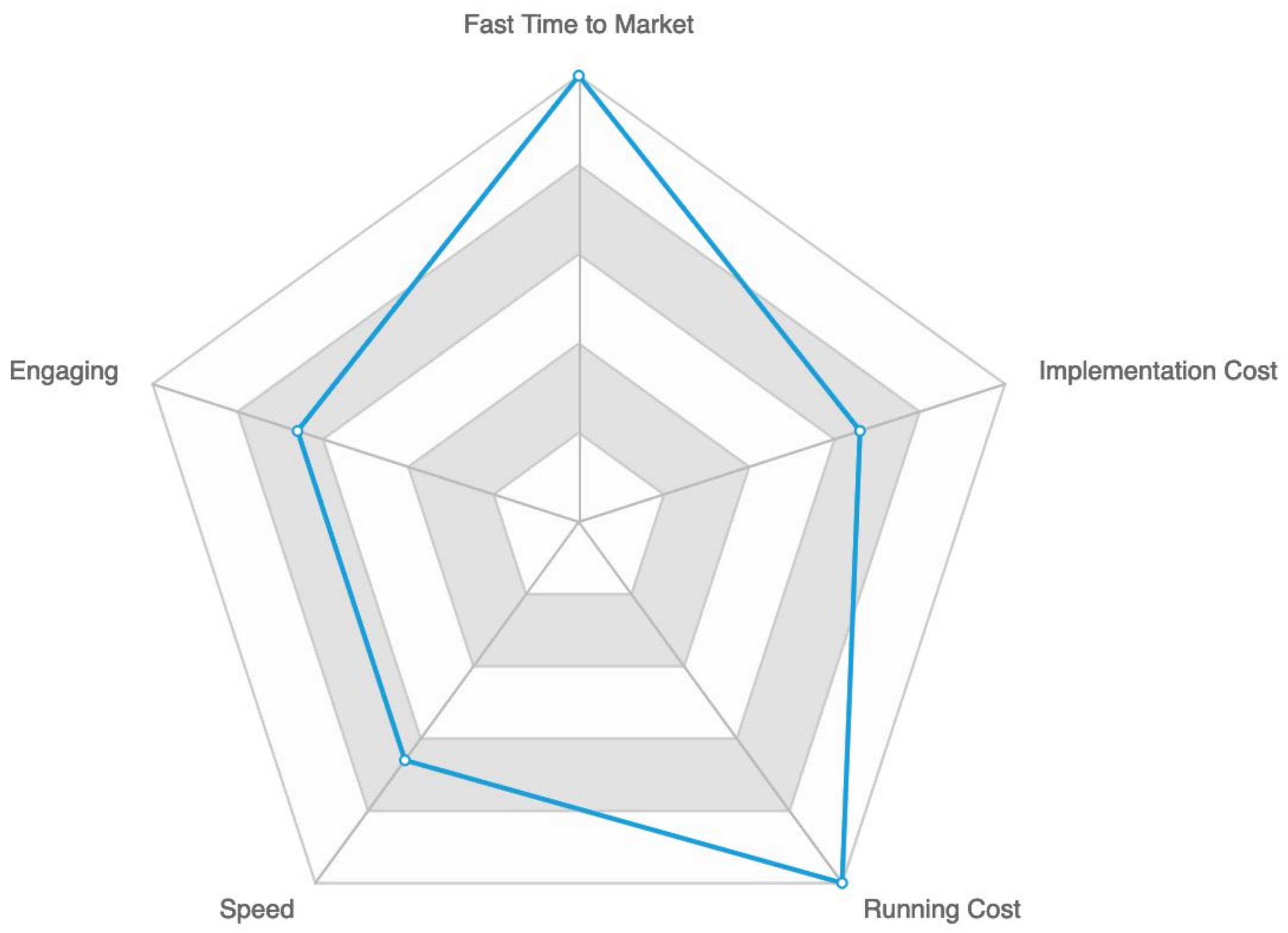
```
'use strict';

var server = require('server');
server.extend(module.superModule);

server.prepend('Start', function (req, res, next) {
   var URLRedirectMgr = require('dw/web/URLRedirectMgr');
   var Template = require('dw/util/Template');

   var origin = URLRedirectMgr.redirectOrigin;

   if (origin.indexOf('sw.js') > -1) {
       var publicJS = new Template('sw').render().text;
       response.setContentType('text/javascript');
       response.writer.print(publicJS);
   } else if (origin.indexOf('manifest.json') > -1) {
       response.setContentType('application/manifest+json');
       response.writer.print(
          JSON.stringify(require('*/cartridge/scripts/manifest')
       ));
   } else {
       next();
   }
});

module.exports = server.exports();
```

The web app manifest is declared.

With the basic setup done, you can now implement the PWA experience in the service worker and on your page.

Conclusion

Fast: ✓ Installable: ✓ Reliable: ✓ Engaging: ✓



Single-Page Progressive Web App on B2C

Hosting: Salesforce B2C Commerce

Example: Georg.com / Hair.com

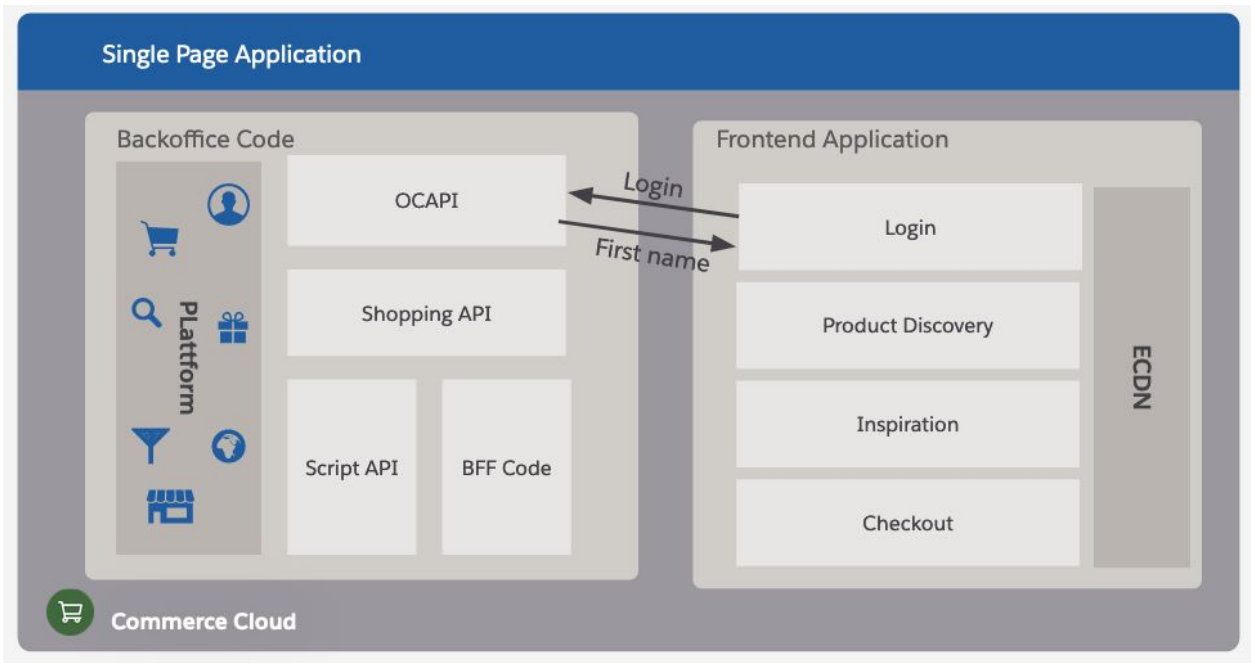
Introduction

Single-page applications are different from traditional website delivery in that a user doesn't download a full page after an interaction. The website updates only areas that the interaction requires.

Typically, the initial page request loads an application that provides all scaffolding website layout and the main application code. The front-end code controls most interactions and state management. Communication between browser and server are handled often via AJAX requests using JSON based REST Services

Building a Custom Single-Page PWA on Salesforce B2C Commerce

With B2C Commerce, you can deliver any web or native experience to shoppers.



Back-End Code

B2C Commerce offers great tools to build single-page applications because the native programming language of the B2C Commerce Engine is JavaScript. JavaScript enables you to quickly exchange data in the native JSON format. The requests can be consumed by out-of-the-box APIs such Open Commerce APIs or shopping APIs.

For more advanced APIs or integrations, you can build custom Back-End for Front-End (BFF) using scripting APIs.

When building the application, apply the following tips:

- Use SFRA
    - Even when building a custom front-end, SFRA is useful since it provides ViewModels to be consumed by the application.
- Be nimble
    - Build requests that are purposeful to the users request. While it might be tempting to call a product OCAPI on a product list as the properties might come in handy, the payload that has to be transferred or generated can cause a slower experience.
- Leverage Caching
    - Since a major driver for PWA is the increased rendering performance, shoppers expect stellar performance. Applying page caching and object cache on B2C Commerce server side ensures quick processing of requests.
- Server Side Rendering
    - To ensure that older search engines can crawl the page efficiently, the server should be able to deliver an HTML version for a URL. At least content and catalog pages should be rendered in pure HTML. SFRA is a perfect base for that approach. Server Side Rendering Libraries for React or Vue are unfortunately not performing well on B2C Commerce.

Example to get the items in cart as JSON:

```
'use strict';

var server = require('server');

server.get('UpdateMiniCart', server.middleware.get, function (req, res, next) {
   var BasketMgr = require('dw/order/BasketMgr');
   var currentBasket = BasketMgr.getCurrentBasket();
   var responseObject = {};
   var itemsToRecalculate = [];
   responseObject.quantityTotal = 0;

   if (currentBasket) {
       responseObject.quantityTotal = currentBasket.productQuantityTotal;
   }

   if (currentBasket) {
       var productLineItems = currentBasket.productLineItems;
       responseObject.recalculate = false;
       for (var i = 0; i < productLineItems.length; i += 1) {
           // we have unsynced items in the cart
           if (!productLineItems[i].custom.calculates) {
               responseObject.recalculate = true;
               itemsToRecalculate.push(productLineItems[i].getProductID());
           }
       }
       responseObject.itemsToRecalculate = itemsToRecalculate.join(',');
   }

   res.json(responseObject);
   next();
});

module.exports = server.exports();
```

Front-End Code

You can build the front-end code with custom code, lightning web components, React, Vue, or any other popular library.

The service worker can be delivered as explained in the previous section. The service worker should apply its caching API to cache the main page decorator and the content for the requested URL.

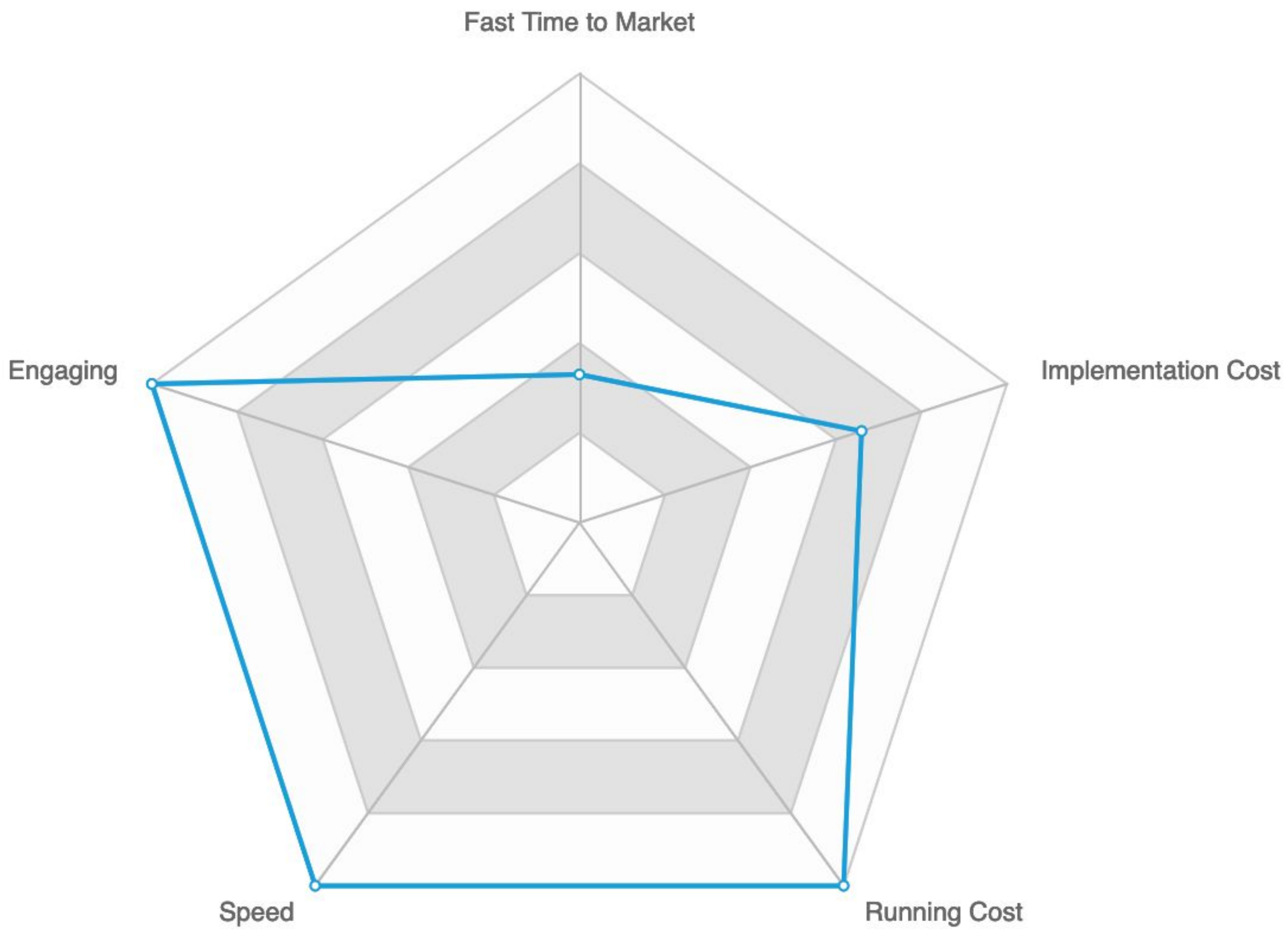
You are free to choose your preferred framework.

However, remember to:

- Be nimble
    - Build requests that are purposeful to the users request. While it might be tempting to call a product OCAPI on a product list as the properties might come in handy, the payload has to be downloaded and parsed, which results in a slower experience.
- Leverage Caching
    - Since a major driver for PWA is the increased rendering performance, shoppers expect stellar performance. Using the browser caches API is an easy way to intercept network calls if they occurred previously..
- Offline Use
    - A steady network connection is never guaranteed. Make sure your application handles offline situations well.

Conclusion

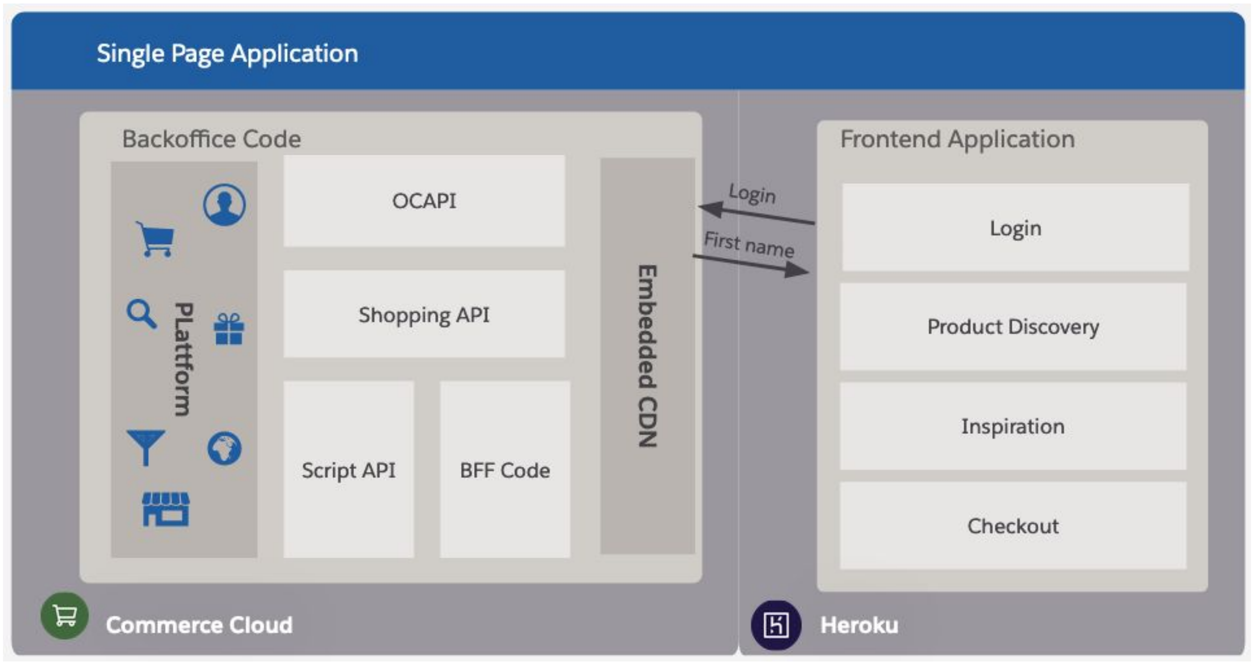
Fast: ✓ Installable: ✓ Reliable: ✓ Engaging: ✓



Single-Page Progressive Web App for Headless

Hosting: Heroku / Mobify / Etc

Example: Lancôme USA



As discussed in the previous section, B2C Commerce can spoon feed a custom front-end application if it is hosted on B2C Commerce. However this technique can also be applied if the front-end stack is not hosted on B2C Commerce.

For the back-end, refer to the previous section.

Front-End Code

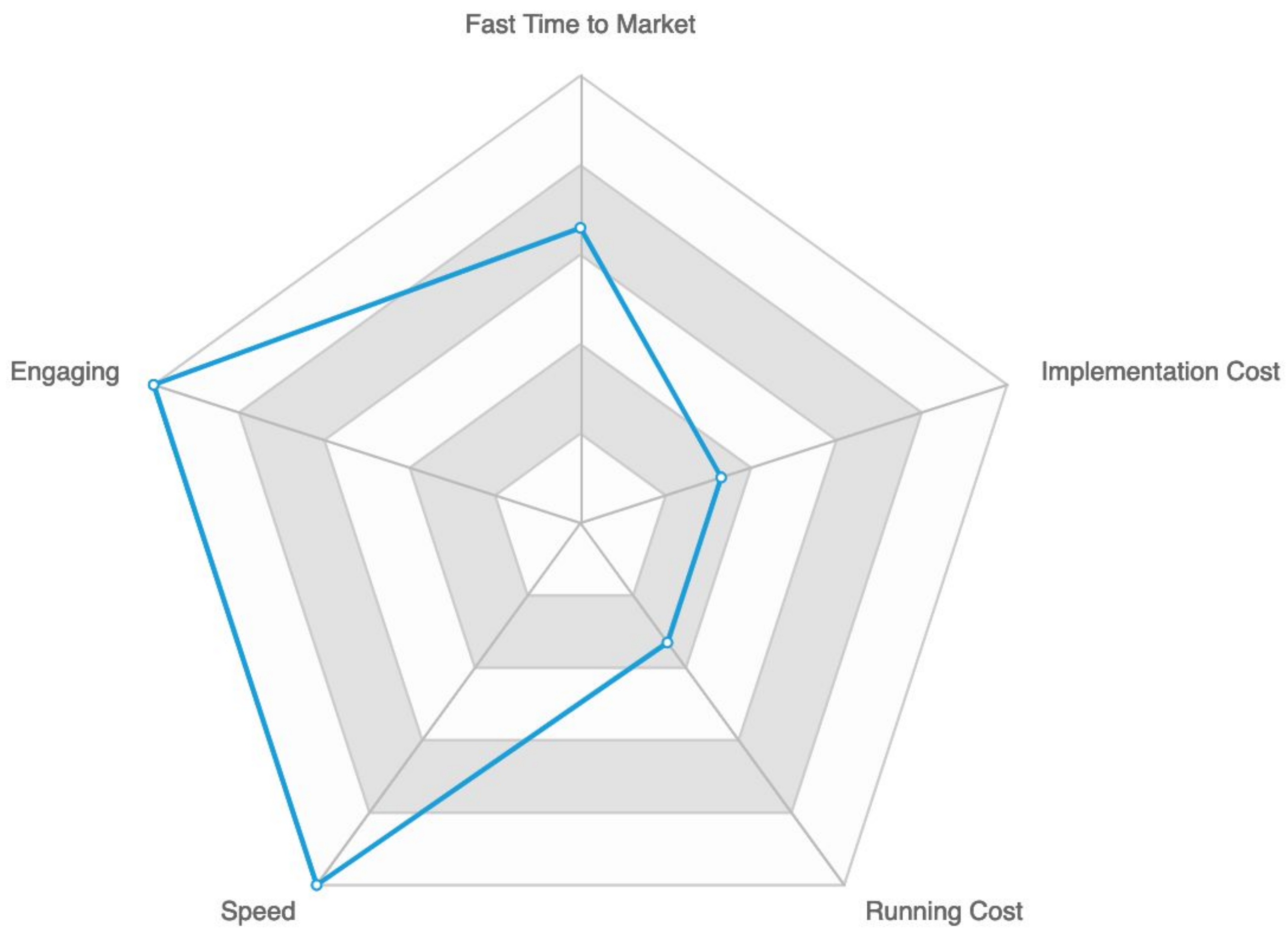
As illustrated in the graphic above, you are free to use any front-end system. Cloud vendors such as Mobify provide a toolkit based on NodeJS. For Platform-as-a-Service vendors like Google Cloud Platform, Heroku, or AWS, any web technology is possible, such as NodeJS, PHP, Java in combination with React, VueJS, WebComponents, or a completely custom solution.

The URLs and SEO capabilities are generated by the new platform, so keep the following items in mind:

- Friendly URLs
    - Provide a speaking URL to share with friends. Some search engines can also use a speaking URL.
- Check Localization Capabilities
    - Since internationalization is important to all vendors, have a plan for localization that is easily matched with B2C Commerce data localization.
- Server Side Rendering
    - For older bots, ensure that an HTML version of the page is accessible to a URL. Use the SSR capabilities of your library.
- CDN
    - Since the embedded CDN can’t be used for an external application, we reccommend that you contract a dedicated CDN.
- Caching
    - Apply a cache layer:
        - On B2C Commerce
        - On your App host
        - In the CDN
        - On the browser network layer
        - In the application

Conclusion

Fast: ✓ Installable: ✓ Reliable: ✓ Engaging: ✓



Conclusion: Progressive Web App on B2C Commerce

This paper demonstrates several options for implementing a PWA on B2C Commerce. You might prefer one approach over another, but PWA Benefits can be achieved with any of the solutions described. Consider cost and flexibility of integration. Ensuring that search engine bots understand the website is achievable with all approaches, but is most easily achieved with a multi-page PWA.

But remember that you can mix and match the approaches. Would you like to implement a multi-page catalog, a Heroku-hosted product configurator, and an SPA Checkout? Give it a try and learn what works best for you and your team.

RESOURCES

- Progressive Web Apps, The Superpowers of The Web and Native Apps Combined by Google & Awwwards
- Web.dev section on PWAs
- Checklist: what makes a good PWA?
- The business impact of a fast, installable and reliable PWA experience featuring Ebay, Weekendesk and Trivago