Web App Push Notification Guide

Date: 2020-05-13
Prepared By: Solutions Engineering

Overview

Wouldn't it be great if your web app could receive the same push notifications from SendBird that you are used to from your mobile applications? When you think of push notifications you probably think of your mobile device, but fortunately, you can also use push notifications with your web app on both desktop and mobile.

In fact, a lot of web applications are now PWAs, progressive web applications, that allow functionality such as offline access and push notifications on both desktop and mobile. We will be using the Notification API and Push API for this guide. There is a **very simple app** to download here. You will need to "npm install" and "npm start" then change the firebase configuration, vapid id, and SendBird credentials.

What is a PWA?

A progressive web app is a web application that looks and acts like a native application. For example, if your PWA is accessed on mobile it could be installed and used while offline or receive push notifications while in the background.

Do I need a PWA?

To receive push notifications from the SendBird server your web app must be a PWA. Most web apps today are accessed on both desktop and mobile browsers, but even if your app is only intended to be used in a desktop browser there are still a number of benefits to a desktop PWA.

If you're still not sold on PWAs you can still use the Notification API to display notifications to users when the application is open and a SendBird event handler is called, but these aren't true push notifications.

Setting up your service worker

Regardless of whether your application is written in vanilla JavaScript or you are using a particular framework such as React, Angular, Vue, or something else, you should still be able to create a PWA, although you may need to move some of this code a slightly different location if using a framework. If you are using React Native, refer to our documentation here.

The key to your PWA is the service worker. This worker runs in the background which is what allows you to receive push notifications as well as other PWA functionality not covered in this guide. Create an empty file called firebase-messaging-sw.js in the project root.

At the bottom of your index.html register the firebase-messaging-sw.js file. This will make sure the browser supports service workers before registering. Omit the first line if you already have it.

index.html

```
<script src="index.js"></script>
<script>
  if ('serviceWorker' in navigator) {
  window.addEventListener('load', () => {
   navigator.serviceWorker
   .register('YOUR_PATH/firebase-messaging-sw.js')
   .then(registration => {
       console.log('SW registered: ', registration);
     }).catch(registrationError => {
       console.log('SW registration failed: ', registrationError);
     });
   });
  }
 </script>
```

Setting up Firebase

We will configure Firebase by creating a new Firebase app, register your web app with Firebase, add the Firebase SDKs you will use, and initialize Firebase in your app.

After creating your Firebase app using the Firebase console we will register your app with Firebase by clicking the web icon in the Firebase console, entering the name of your application, and clicking the register button.

There are a few options to add the Firebase SDKs to your application. For this guide, we will use NPM to add the Firebase SDKs to your index.js file and the Firebase CDN to use those SDKs in the service worker.

To import the SDKs for Firebase in our index.js file, install the Firebase package `npm install --save firebase`.

To initialize Firebase in your app, use the details provided when registering the app in the Firebase console. Make sure to use your own configuration details. Add the following to your index.js file.

Index.js

```
import * as firebase from 'firebase/app';
import 'firebase/messaging';

var firebaseConfig = {
  apiKey: "fake_key",
  authDomain: "app-name.firebaseapp.com",
  databaseURL: "https://app-name.firebaseio.com",
  projectId: "project-id",
  storageBucket: "app-name.appspot.com",
  messagingSenderId: "12345",
  appId: "12345"
};
firebase.initializeApp(firebaseConfig);
```

Next add the following to your firebase-messaging-sw.js in your root folder to configure Firebase in the service worker. This will allow us to configure background notifications later. Be sure to use the messagingSenderId from your Firebase console.

Also, be sure to update the snippet below to use the latest firebase JS versions.

Firebase-messaging-sw.js

```
importScripts('https://www.gstatic.com/firebasejs/7.3.0/firebase-app.js');
importScripts('https://www.gstatic.com/firebasejs/7.3.0/firebase-messaging.js');

const firebaseConfig = {
 messagingSenderId: "12345"
};
firebase.initializeApp(firebaseConfig);
```

Setting up Firebase Messaging

Next, we will configure our app to work with firebase cloud messaging.

If you don't have one, create a manifest.json file in your root directory and add the following. Don't change the gcm_sender_id.

manifest.json

```
{
 "gcm_sender_id": "103953800507"
}
```

Generate a VAPID keypair in your Firebase console. In your index.js file, add the following below your firebase initialization. This can be found in your Firebase console:

Firebase__console → Project_overview → Project_settings → Cloud_messaging → Web_push_certificates

Add the Vapid ID to your index.js

index.js

```
const messaging = firebase.messaging();

 messaging.usePublicVapidKey(
   'YOUR PUBLIC KEY'
 );
```

Connect to SendBird, request permission to display notifications to the user, and register your push token with SendBird (don't forget to turn on push in your Sendbird Dashboard and register your FCM credentials).

index.js

```
import SendBird from "sendbird"
const sb = new SendBird({appId});

sb.connect(userId, nickname, (user, error) => {
 Notification.requestPermission().then(permission => {
   if (permission === 'granted') {
     messaging
       .getToken()
       .then(currentToken => {
         if (currentToken) {
           console.log(currentToken)
           sb.
           registerGCMPushTokenForCurrentUser(currentToken, (response, error) => {
             if(error) console.log(error)
           });
         }
       })
       .catch(err => {
```

```
        console.log('An error occurred while retrieving token. ', err);
      });
  } else {
    console.log('Unable to get permission to notify.');
  }
  })
})
```

You will want to listen for and handle changes to your push token. In your index.js file add the following.

index.js

```
 messaging.onTokenRefresh(() => {
   messaging
     .getToken()
     .then(refreshedToken => {
       SendBirdAction.getInstance().registerGCMPushTokenForCurrentUser(refreshedToken)
       .then(response => console.log('Successfully registered token with SendBird.',
response))
       .catch(error => console.log('Could not register token with SendBird.', error));
     })
     .catch(err => {
       console.log('Unable to retrieve refreshed token ', err);
     });
 });
```

Receiving push notifications

By default, SendBird will only send push notifications to users that are offline and receive a message in a group channel. The browser must be open to receive push notifications, but the application doesn't have to be active or in the foreground.

If you would like push notifications to be sent when a user is connected you need to use the channelEventListener's onMessageReceived event. The handler can be added right after instantiating the new SendBird instance. From the `onMessageReceived` callback consider that you could display an indication of the new message via the same Notification method used by the browsers push service.

Alternatively, it is now possible to alter Sendbird's default settings and send a push notification for all messages. The tradeoff is that there will be two identical messages arriving at the device if the user happens to be online. That is to say one push notification via FCM will arrive and one real time message via websocket will also arrive.

```
const sb = new SendBird({appId: APP_ID});
```

```
const ChannelHandler = new sb.ChannelHandler();

ChannelHandler.onMessageReceived = function(channel, message) {
 // Consider calling the Notification service from here.
};

sb.addChannelHandler(UNIQUE_HANDLER_ID, ChannelHandler);
```

Refer to the chart below to see what type of notification will be generated based on connection status, background state, and whether the application's tab is active:

Default → Send when all devices are offline

User Connected	Browser Running	Tab Focus	setBackgroundState() called	Channel Event Handler Applied	Notification Type	Notes
FALSE	TRUE	TRUE	TRUE	N/A	FCM Data Message	If firebase-messaging-sw is registered
FALSE	TRUE	FALSE	TRUE	N/A	FCM Data Message	If firebase-messaging-sw is registered
TRUE	TRUE	TRUE	FALSE	TRUE	onMessageReceived Fires via SendBird Websocket	
TRUE	TRUE	FALSE	FALSE	TRUE	onMessageReceived Fires via SendBird Websocket	
FALSE	TRUE	FALSE	FALSE	N/A	FCM Data Message	If firebase-messaging-sw is registered

Send to all devices online and offline (Dashboard → Settings → Notifications)

User Connected	Browser Running	Tab Focus	Channel Event Handler Applied	Notification Type	Notes
FALSE	TRUE	TRUE	N/A	FCM Data Message	If firebase-messaging-sw is registered
FALSE	TRUE	FALSE	N/A	FCM Data Message	If firebase-messaging-sw is registered
TRUE	TRUE	TRUE	TRUE	onMessageReceived Fires via SendBird Websocket and FCM Data Message	
TRUE	TRUE	FALSE	TRUE	onMessageReceived Fires via SendBird Websocket and FCM Data Message	
FALSE	TRUE	FALSE	N/A	FCM Data Message	If firebase-messaging-sw is registered

Listening for FCM data messages (Push Notifications)

To set up our message handler for handling FCM data messages push notifications add the following to your firebase-messaging-sw.js file.

```
const messaging = firebase.messaging();

messaging.setBackgroundMessageHandler(function(payload) {
 console.log('[firebase-messaging-sw.js] Received background message ', payload);
 const channelName = JSON.parse(payload.data.sendbird).channel.name;
 const notificationTitle = `Background New message in ${channelName}`;
 var notificationOptions = {
   body: payload.data.message,
 };

 return self.registration.showNotification(notificationTitle,
   notificationOptions);
});
```

That's it, you should be able to receive push notifications while the application is in the background, and or foreground depending on your Sendbird Dashboard preferences.

Push Notification Test Flow

Step 1: Check the push notification set up works without Sendbird

- Firebase team has not developed a UI to send `data-messages` to your devices. Sendbird uses only FCM `data-messages` This means without exception there is no way to use the Firebase Console for testing your PWA's push notifications in relation to Sendbird.
- To test FCM `data-messages` use PostMan and close all tabs with your application in:
    - POST → https://fcm.googleapis.com/fcm/send
    - Headers:
    - **Key**: Content-Type, **Value**: application/json
    - **Key**: Authorization, **Value**: **key=**<your-fcm-server-key>
        - Your-fcm-server-key → Firebase Console → Your project → Project Overview Cog → Project Settings → Cloud Messaging → Server Key
    - Collect the device token.

        - See the index.js code above for a log of the device token or check
    - Body:

```
{
  "data": {
    "my_custom_key": "my_custom_value",
    "my_custom_key2": true
  },
  "registration_ids": [
    "{device-token}"    ]
}
```

Step 2: Testing with Sendbird

Register a device token to Sendbird

- Check in the Sendbird Dashboard you have push notifications turn on and have registered an FCM Server Key
- Check that the server key you have registered in the Sendbird Dashboard matches with the one used in the PostMan testing phase.
- Create two new users `target_user_1`, `sending_user_1`
    - Use the Sendbird Dashboard or PostMan to create one Sendbird Group Channel with the name and channel url `push_testing`

Curl for creating a channel

```
curl --location --request POST
'https://api-885C2616-DBF8-4BDC-9178-4A1A662614E3.sendbird.com/v3/group_channels/push_testing' \
--header 'Api-Token: f1f8361964b0c24b2eb9483c595eeb1d646bbc55' \
--header 'Content-Type: application/json' \
--data-raw '{
"user_ids" : ["target_user_1", "sending_user_1"],
"name": "push_testing",
"channel_url": "push_testing"
}'
```

- In your application login with `target_user_1` .
    - Next, check in the Sendbird Dashboard that `target_user_1` has a registered device token.
    - Dashboard → Users → Chat → `target_user_1` → Push Token → Android

- Do the test again by deleting the device token in the Dashboard and logging `target_user_1` in again.
    - Do this a couple of times to be sure the device token always updates in the Sendbird Dashboard.

Send a push notification

- Make the testing user `target_user_1` offline by closing all of your application's browser tabs
- Using PostMan to send a message to Sendbird to the `push_test` channel on behalf of the `sending_user_1`

```
curl --location --request POST
'https://api-885C2616-DBF8-4BDC-9178-4A1A662614E3.sendbird.com/v3/group_channels/push_testing/messages' \
--header 'Api-Token: f1f8361964b0c24b2eb9483c595eeb1d646bbc55' \
--header 'Content-Type: application/json' \
--data-raw '{
"user_id" : "sending_user_1",
"message": "test1"
}'
```