Don't forget the **Chrome Dev Summit**, starting Monday at 10:00am (Pacific) and streaming live on YouTube. Schedule. (https://developer.chrome.com/devsummit/schedule)

Adding Push Notifications to a Web App

By Matt Gaunt (https://developers.google.com/web/resources/contributors#mattgaunt)
Matt is a contributor to Web**Fundamentals**

Overview

Push messaging provides a simple and effective way to re-engage with your users and in this code lab you'll learn how to add push notifications to your web app.

What you'll learn

- How to subscribe and unsubscribe a user for push messaging
- How to handle incoming push messages
- How to display a notification
- How to respond to notification clicks

What you'll need

- Chrome 52 or above
- Web Server for Chrome (https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb)
  , or your own web server of choice

- A text editor
- Basic knowledge of HTML, CSS, JavaScript, and Chrome DevTools
- The sample code, see Get setup

Get Setup

Download the sample code

You can get the sample code for this code by either downloading the zip here:

Download source code (https://github.com/googlechrome/push-notifications/archive/master.zip)

or by cloning this git repo:

```
git clone https://github.com/GoogleChrome/push-notifications.git
```

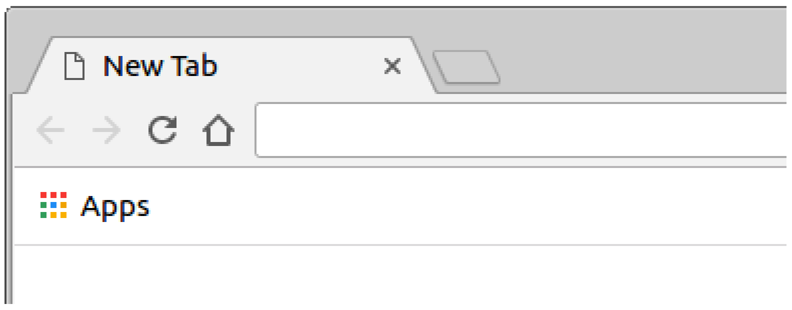
If you downloaded the source as a zip, unpacking it should give you a root folder **`push-notifications-master`**.

Install and verify web server

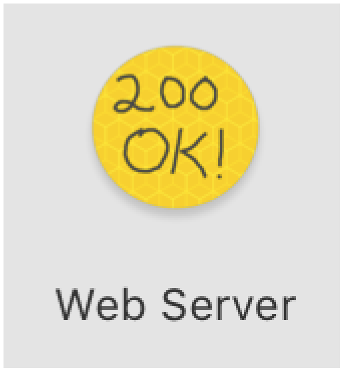
While you're free to use your own web server, this codelab is designed to work well with the Chrome Web Server. If you don't have that app installed yet, you can install it from the Chrome Web Store.

Install Web Server for Chrome (https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb)

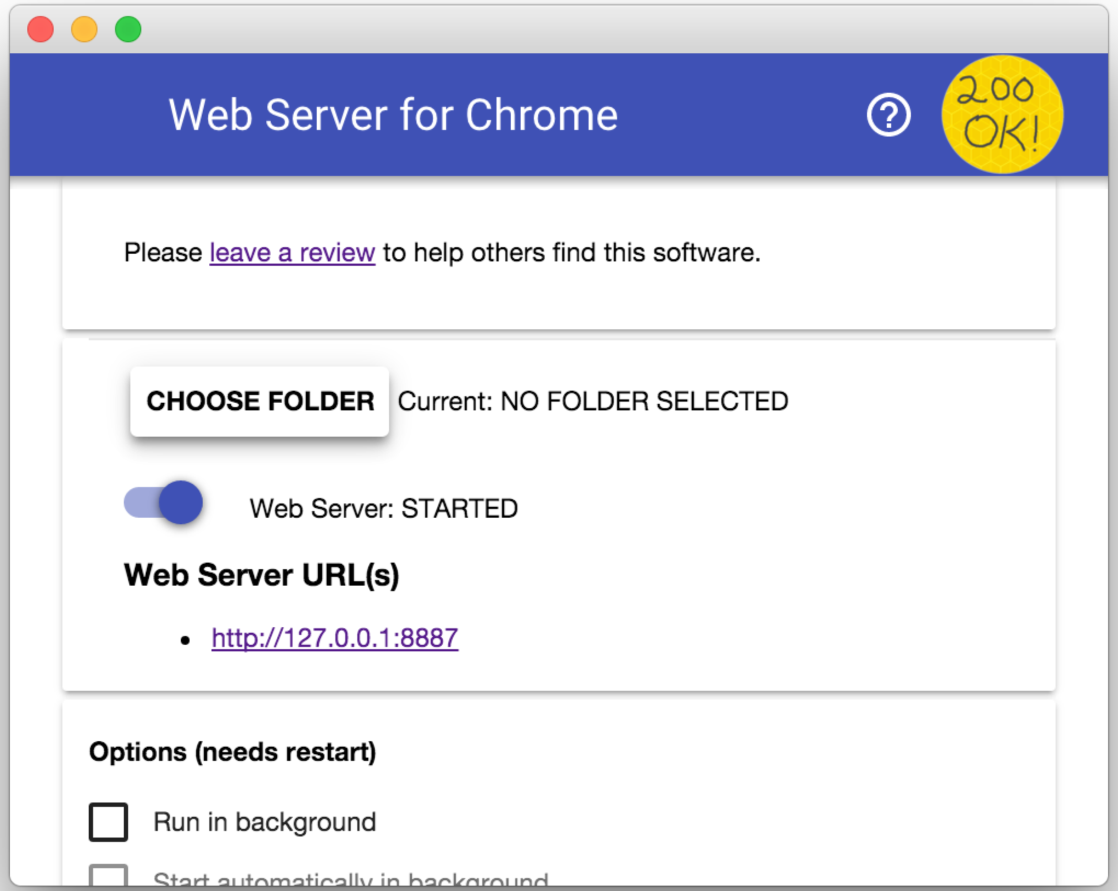
After installing the Web Server for Chrome app, click on the Apps shortcut on the bookmarks bar:



In the ensuing window, click on the Web Server icon:



You'll see this dialog next, which allows you to configure your local web server:



Please leave a review to help others find this software.

Current: NO FOLDER SELECTED

Web Server: STARTED

Web Server URL(s)

- http://127.0.0.1:8887

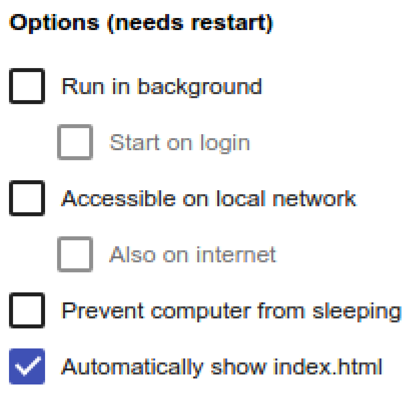
Options (needs restart)

☐ Run in background

☐ Start automatically in background

Click the **choose folder** button, and select the app folder. This will enable you to serve your work in progress via the URL highlighted in the web server dialog (in the **Web Server URL(s)** section).

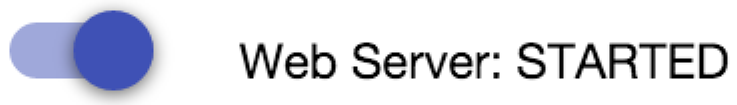
Under Options, check the box next to "Automatically show index.html", as shown below:



Options (needs restart)

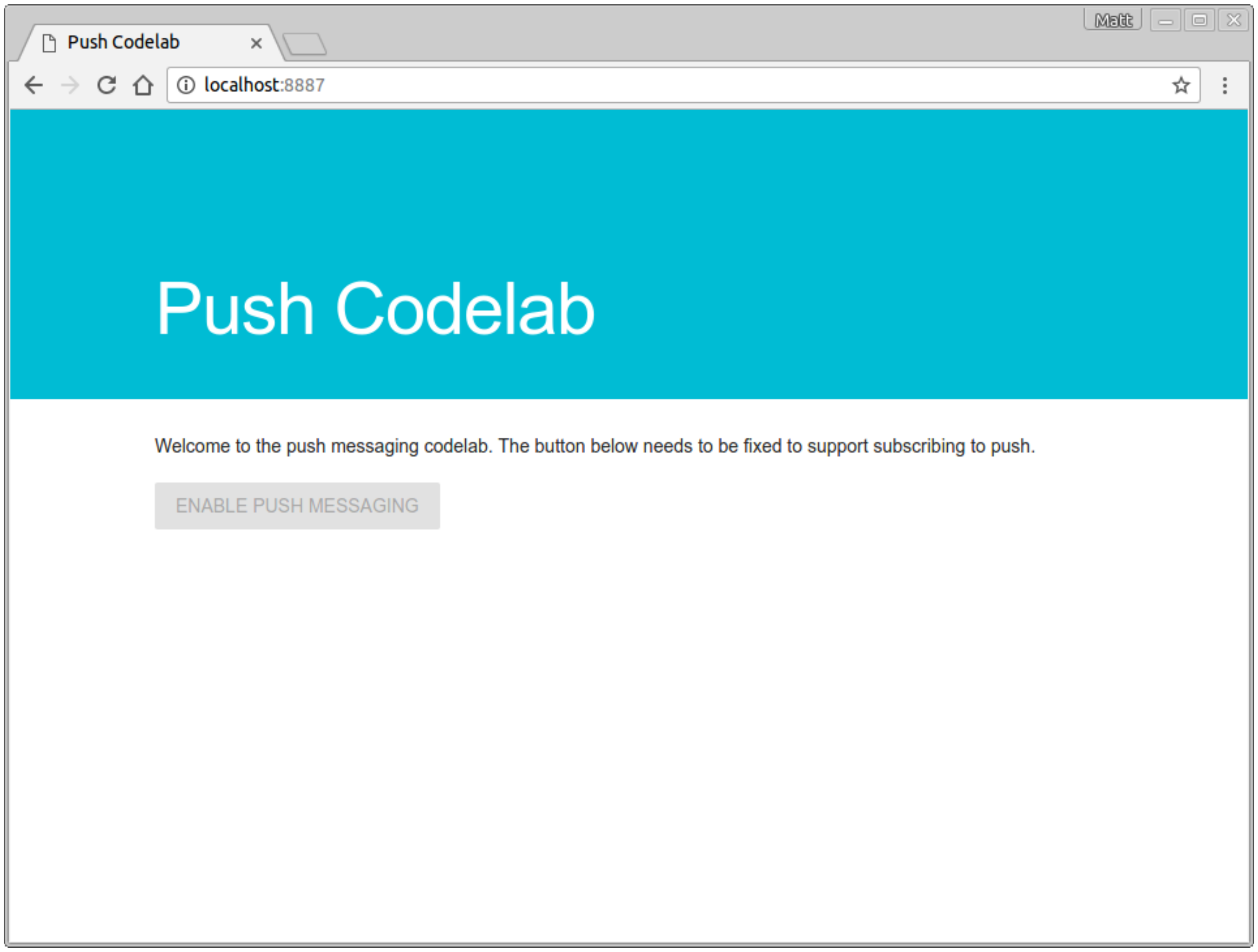
- [ ] Run in background
  - [ ] Start on login
- [ ] Accessible on local network
  - [ ] Also on internet
- [ ] Prevent computer from sleeping
- [x] Automatically show index.html

Then stop and restart the server by sliding the toggle labeled "Web Server: STARTED" to the left and then back to the right.



Web Server: STARTED

Now visit your site in your web browser (by clicking on the highlighted Web Server URL) and you should see a page that looks like this:



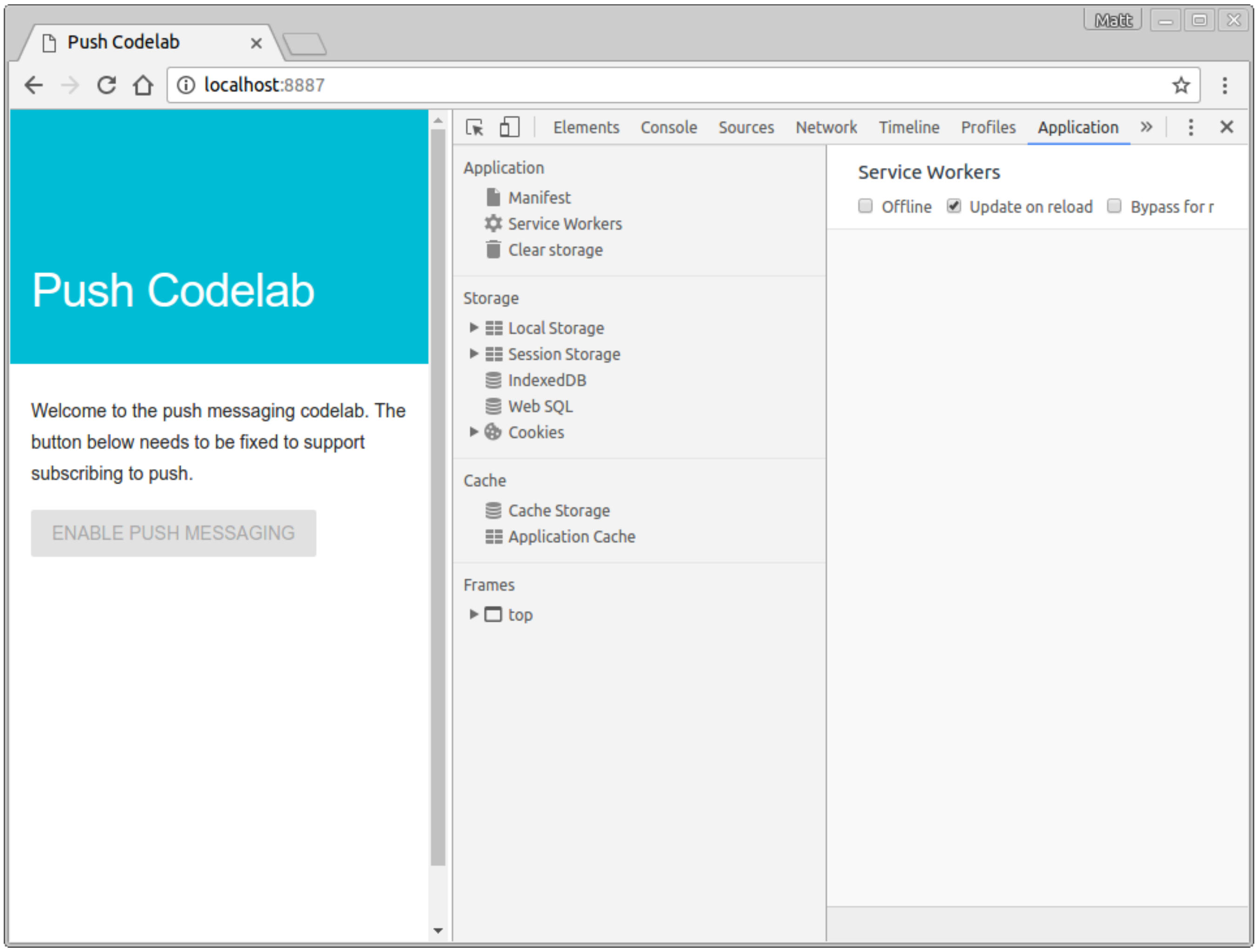
Push Codelab

Welcome to the push messaging codelab. The button below needs to be fixed to support subscribing to push.

Always update the service worker

During development it's helpful to ensure your service worker is always up to date and has the latest changes.

To set this up in Chrome, open DevTools (Right Click > Inspect) and go to the **Application** panel, click the **Service Workers** tab and check the **Update on Reload** checkbox. When this checkbox is enabled the service worker is forcibly updated every time the page reloads.



Push Codelab

Welcome to the push messaging codelab. The button below needs to be fixed to support subscribing to push.

Register a Service Worker

In your **app** directory, notice that you have an empty file named **sw.js**. This file will be your service worker, for now it can stay empty and we'll be adding code to it later.

First we need to register this file as our Service Worker.

Our **app/index.html** page loads **scripts/main.js** and it's in this JavaScript file that we'll register our service worker.

Add the following code to **scripts/main.js**:

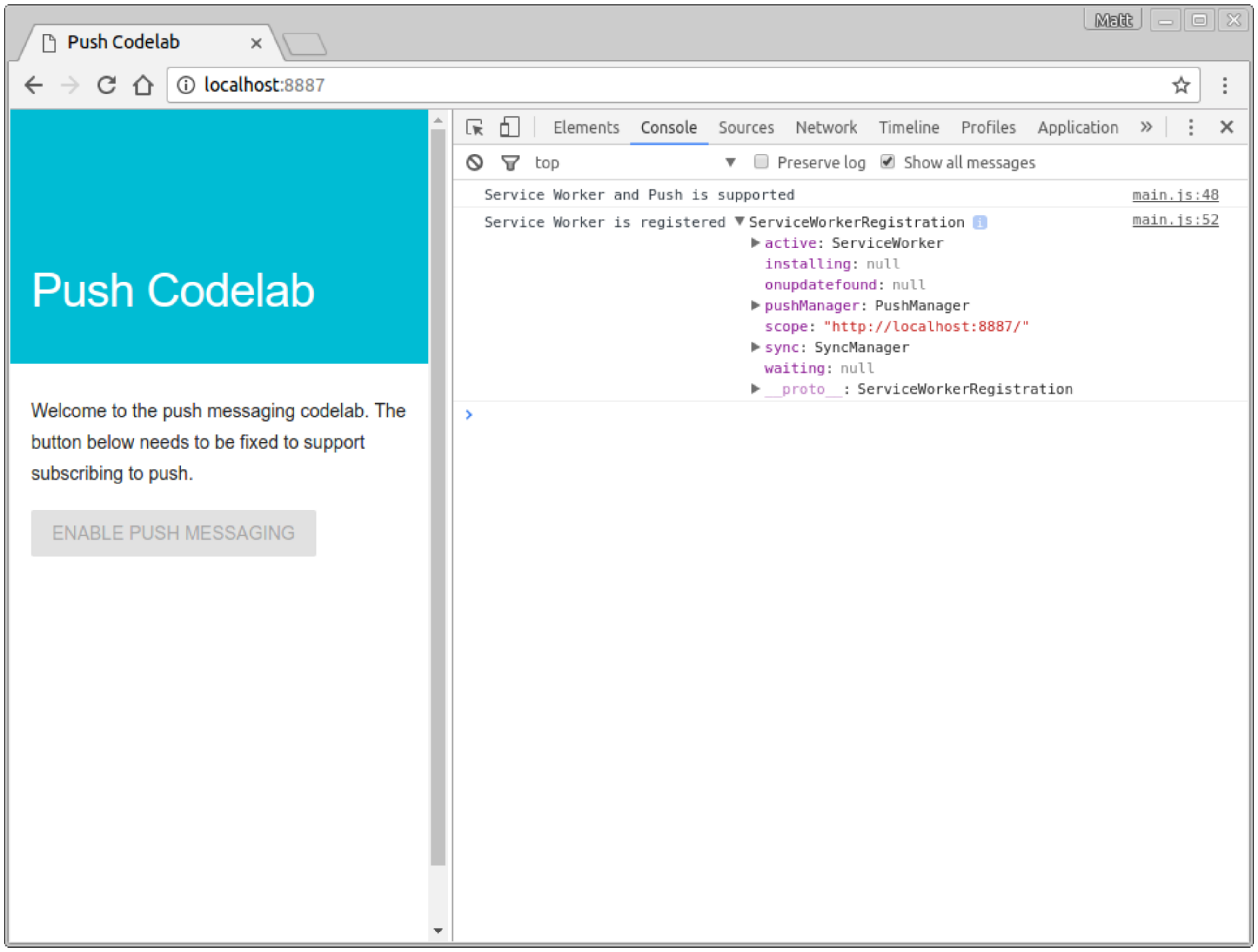
```
if ('serviceWorker' in navigator && 'PushManager' in window) {
  console.log('Service Worker and Push is supported');

  navigator.serviceWorker.register('sw.js')
  .then(function(swReg) {
    console.log('Service Worker is registered', swReg);

    swRegistration = swReg;
  })
  .catch(function(error) {
    console.error('Service Worker Error', error);
  });
} else {
  console.warn('Push messaging is not supported');
  pushButton.textContent = 'Push Not Supported';
}
```

This code checks if service workers and push messaging is supported by the current browser and if it is, it registers our `sw.js` file.

Try it out

Check your changes by refreshing the app in the browser.

Check the console in Chrome DevTools for **`Service Worker is registered`**, like so:



Push Codelab

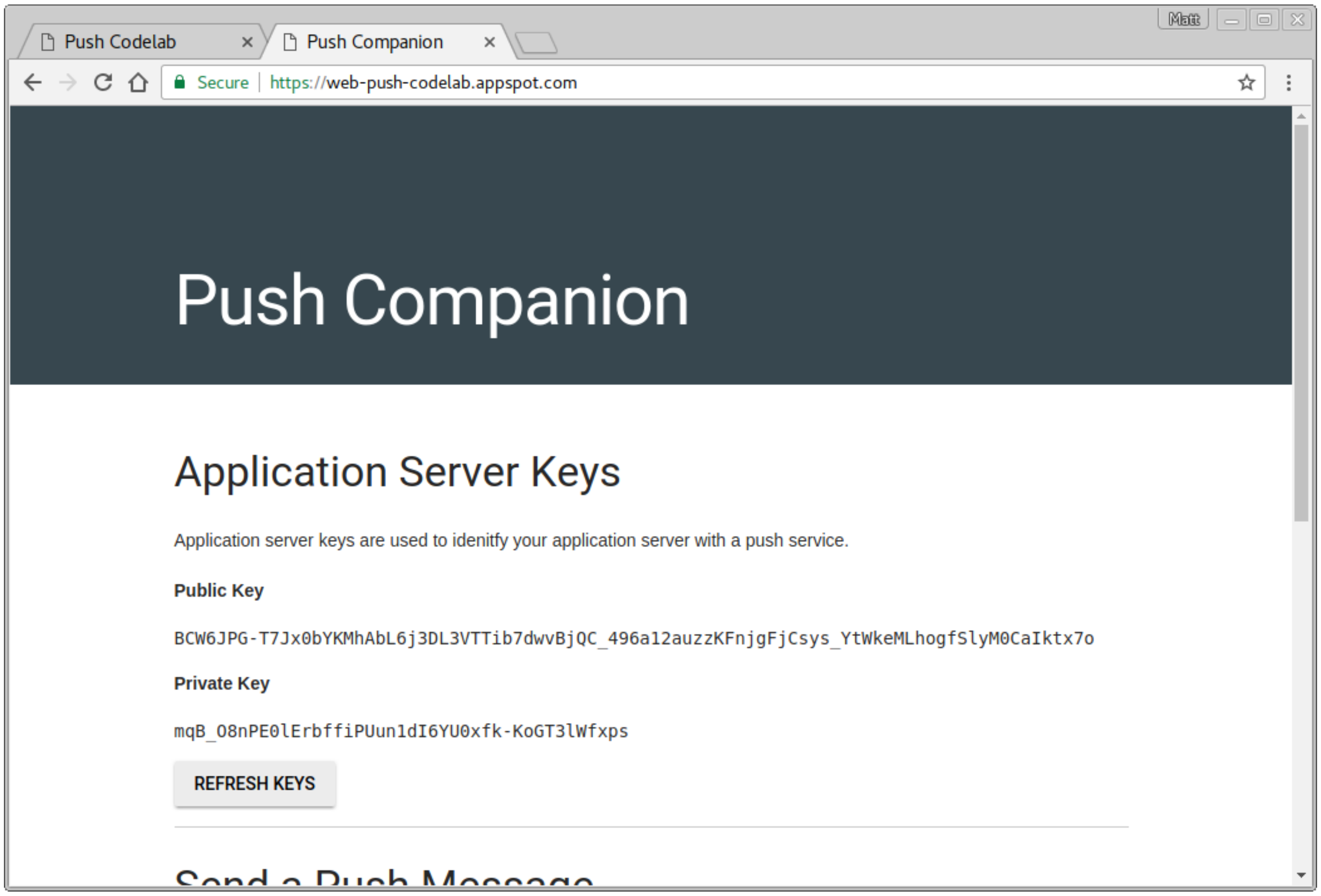
Welcome to the push messaging codelab. The button below needs to be fixed to support subscribing to push.

```
Service Worker and Push is supported                                      main.js:48
Service Worker is registered ▼ServiceWorkerRegistration                   main.js:52
                              ▶ active: ServiceWorker
                                installing: null
                                onupdatefound: null
                              ▶ pushManager: PushManager
                                scope: "http://localhost:8887/"
                              ▶ sync: SyncManager
                                waiting: null
                              ▶ __proto__: ServiceWorkerRegistration
>
```

Get Application Server Keys

To work with this code lab you need to generate some application server keys which we can do with this companion site: https://web-push-codelab.glitch.me/ (https://web-push-codelab.glitch.me/)

Here you can generate a Public and Private key pair.



Push Companion

Application Server Keys

Application server keys are used to idenitfy your application server with a push service.

Public Key

`BCW6JPG-T7Jx0bYKMhAbL6j3DL3VTTib7dwvBjQC_496a12auzzKFnjgFjCsys_YtWkeMLhogfSlyM0CaIktx7o`

Private Key

`mqB_O8nPE0lErbffiPUun1dI6YU0xfk-KoGT3lWfxps`

REFRESH KEYS

Copy your public key into **`scripts/main.js`** replacing the **`<Your Public Key>`** value:

```
const applicationServerPublicKey = '<Your Public Key>';
```

Note: You should never put your private key in your web app!

Initialize State

At the moment the web app's button is disabled and can't be clicked. This is because it's good practice to disable the push button by default and enable it once you know push is supported and can know if the user is currently subscribed or not.

Let's create two functions in **`scripts/main.js`**, one called **`initializeUI`**, which will check if the user is currently subscribed, and one called **`updateBtn`** which will enable our button and change the text if the user is subscribed or not.

We want our **`initializeUI`** function to look like this:

```
function initializeUI() {
  // Set the initial subscription value
  swRegistration.pushManager.getSubscription()
  .then(function(subscription) {
    isSubscribed = !(subscription === null);

    if (isSubscribed) {
      console.log('User IS subscribed.');
    } else {
      console.log('User is NOT subscribed.');
    }

    updateBtn();
  });
}
```

Our new method uses the **`swRegistration`** from the previous step and calls **`getSubscription()`** on it's **`pushManager`**. **`getSubscription()`** is a method that returns a promise that resolves with the current subscription if there is one, otherwise it'll return **`null`**. With this we can check if the user is already subscribed or not, set some state and then call **`updateBtn()`** so the button can be enabled with some helpful text.

Add the following code to implement the **`updateBtn()`** function.

```
function updateBtn() {
  if (isSubscribed) {
    pushButton.textContent = 'Disable Push Messaging';
  } else {
    pushButton.textContent = 'Enable Push Messaging';
  }

  pushButton.disabled = false;
}
```

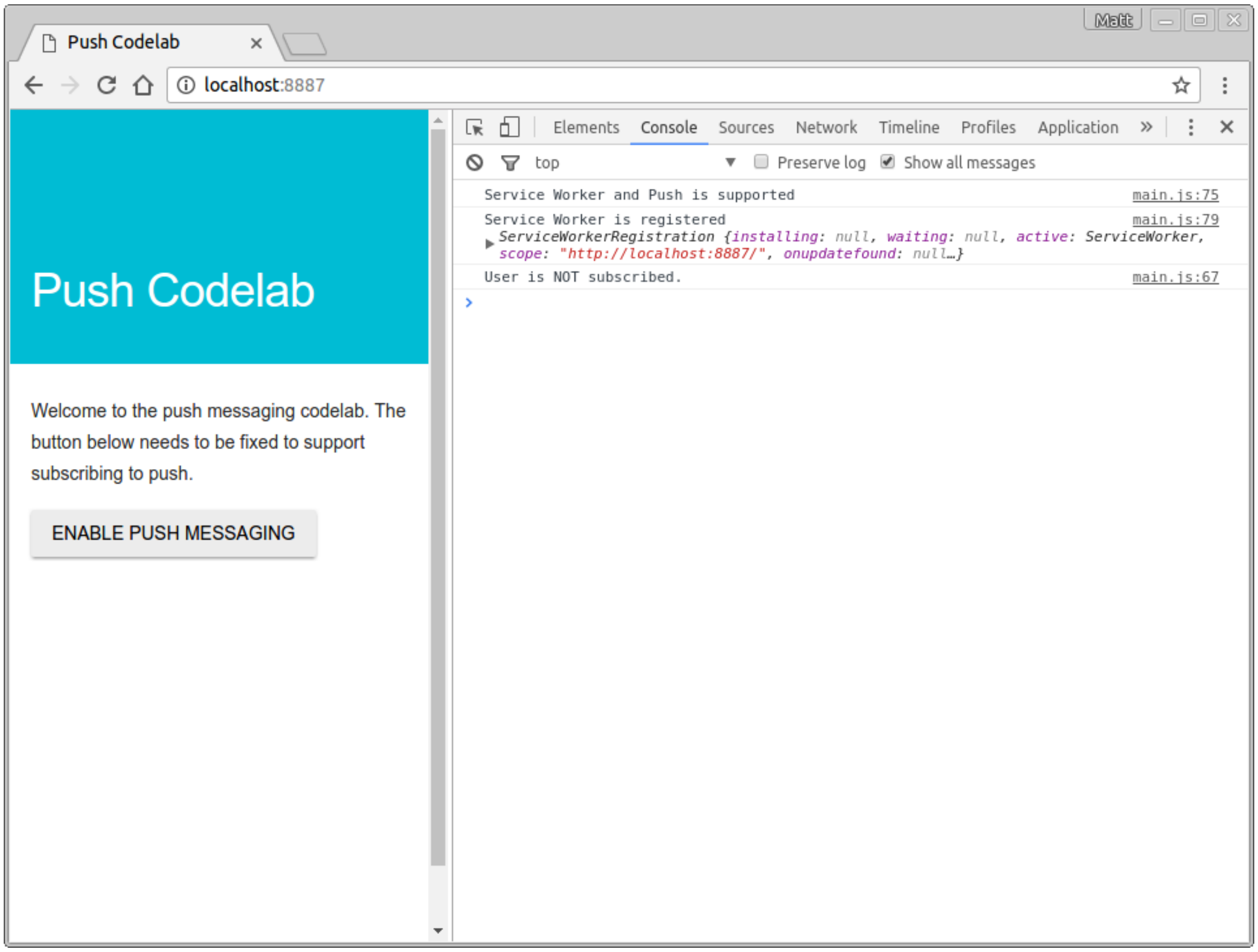
This function simply changes the text depending on the whether the user is subscribed or not and then enables the button.

The last thing to do is call **`initializeUI()`** when our service worker is registered.

```
navigator.serviceWorker.register('sw.js')
.then(function(swReg) {
  console.log('Service Worker is registered', swReg);

  swRegistration = swReg;
  initializeUI();
})
```

Try it out

Refresh your web app and you should see the 'Enable Push Messaging' button is now enabled (you can click it) and you should see 'User is NOT subscribed.' in the console.



Welcome to the push messaging codelab. The button below needs to be fixed to support subscribing to push.

```
Service Worker and Push is supported                                        main.js:75
Service Worker is registered                                                main.js:79
▶ServiceWorkerRegistration {installing: null, waiting: null, active: ServiceWorker,
  scope: "http://localhost:8887/", onupdatefound: null…}
User is NOT subscribed.                                                     main.js:67
```

When we progress through the rest of the code lab you should see the button text change when the user subscribed / un-subscribed.

Subscribe the user

At the moment our ‘Enable Push Messaging' button doesn't do too much, so let's fix that.

Add a click listener to our button in the **`initializeUI()`** function, like so:

```
function initializeUI() {
  pushButton.addEventListener('click', function() {
    pushButton.disabled = true;
    if (isSubscribed) {
      // TODO: Unsubscribe user
    } else {
      subscribeUser();
    }
  });

  // Set the initial subscription value
  swRegistration.pushManager.getSubscription()
  .then(function(subscription) {
    isSubscribed = !(subscription === null);

    updateSubscriptionOnServer(subscription);

    if (isSubscribed) {
      console.log('User IS subscribed.');
    } else {
      console.log('User is NOT subscribed.');
    }

    updateBtn();
  });
}
```

When the user clicks the push button, we first disable the button just to make sure the user can't click it a second time while we're subscribing to push as it can take some time.

Then we call **`subscribeUser()`** when we know the user isn't currently subscribed, so copy and paste the following code into **`scripts/main.js`**.

```
function subscribeUser() {
  const applicationServerKey = urlB64ToUint8Array(applicationServerPublicKey);
  swRegistration.pushManager.subscribe({
    userVisibleOnly: true,
    applicationServerKey: applicationServerKey
  })
  .then(function(subscription) {
    console.log('User is subscribed.');

    updateSubscriptionOnServer(subscription);

    isSubscribed = true;

    updateBtn();
  })
  .catch(function(err) {
    console.log('Failed to subscribe the user: ', err);
    updateBtn();
  });
}
```

Lets step through what this code is doing and how it's subscribing the user for push messaging.

First we take the application server's public key, which is base 64 URL safe encoded, and we convert it to a `UInt8Array` as this is the expected input of the subscribe call. We've already given you the function `urlB64ToUint8Array` at the top of `scripts/main.js`.

Once we've converted the value, we call the `subscribe()` method on our service worker's `pushManager`, passing in our application server's public key and the value `userVisibleOnly: true`.

```
const applicationServerKey = urlB64ToUint8Array(applicationServerPublicKey);
swRegistration.pushManager.subscribe({
  userVisibleOnly: true,
  applicationServerKey: applicationServerKey
})
```

The `userVisibleOnly` parameter is basically an admission that you will show a notification every time a push is sent. At the time of writing this value is required and must be true.

Calling `subscribe()` returns a promise which will resolve after the following steps:

1. The user has granted permission to display notifications.
2. The browser has sent a network request to a push service to get the details to generate a PushSubscription.

The `subscribe()` promise will resolve with a `PushSubscription` if these steps were successful. If the user doesn't grant permission or if there is any problem subscribing the user, the promise will reject with an error. This gives us the following promise chain in our codelab:

```
swRegistration.pushManager.subscribe({
  userVisibleOnly: true,
  applicationServerKey: applicationServerKey
})
.then(function(subscription) {
  console.log('User is subscribed.');

  updateSubscriptionOnServer(subscription);

  isSubscribed = true;

  updateBtn();

})
.catch(function(err) {
  console.log('Failed to subscribe the user: ', err);
  updateBtn();
});
```

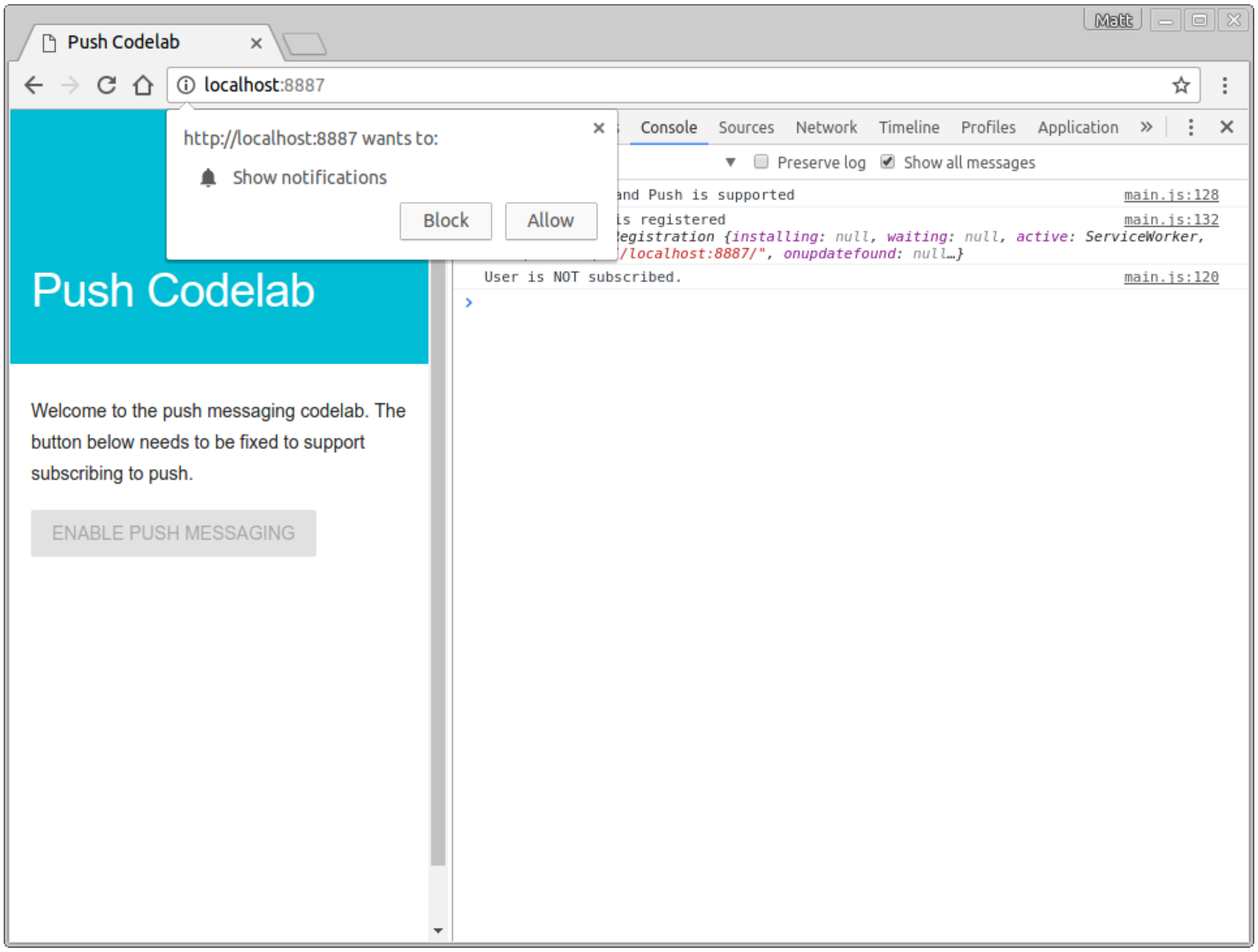
With this, we get a subscription and treat the user as subscribed or we catch the error and print it to the console. In both scenarios we call `updateBtn()` to ensure the button is re-enabled and has the appropriate text.

The method `updateSubscriptionOnServer` is a method where in a real application we would send our subscription to a backend, but for our codelab we are going to print the subscription in our UI which will help us later on. Add this method to `scripts/main.js`:

```
function updateSubscriptionOnServer(subscription) {
  // TODO: Send subscription to application server

  const subscriptionJson = document.querySelector('.js-subscription-json');
  const subscriptionDetails =
    document.querySelector('.js-subscription-details');

  if (subscription) {
    subscriptionJson.textContent = JSON.stringify(subscription);
    subscriptionDetails.classList.remove('is-invisible');
  } else {
    subscriptionDetails.classList.add('is-invisible');
    }
}
```

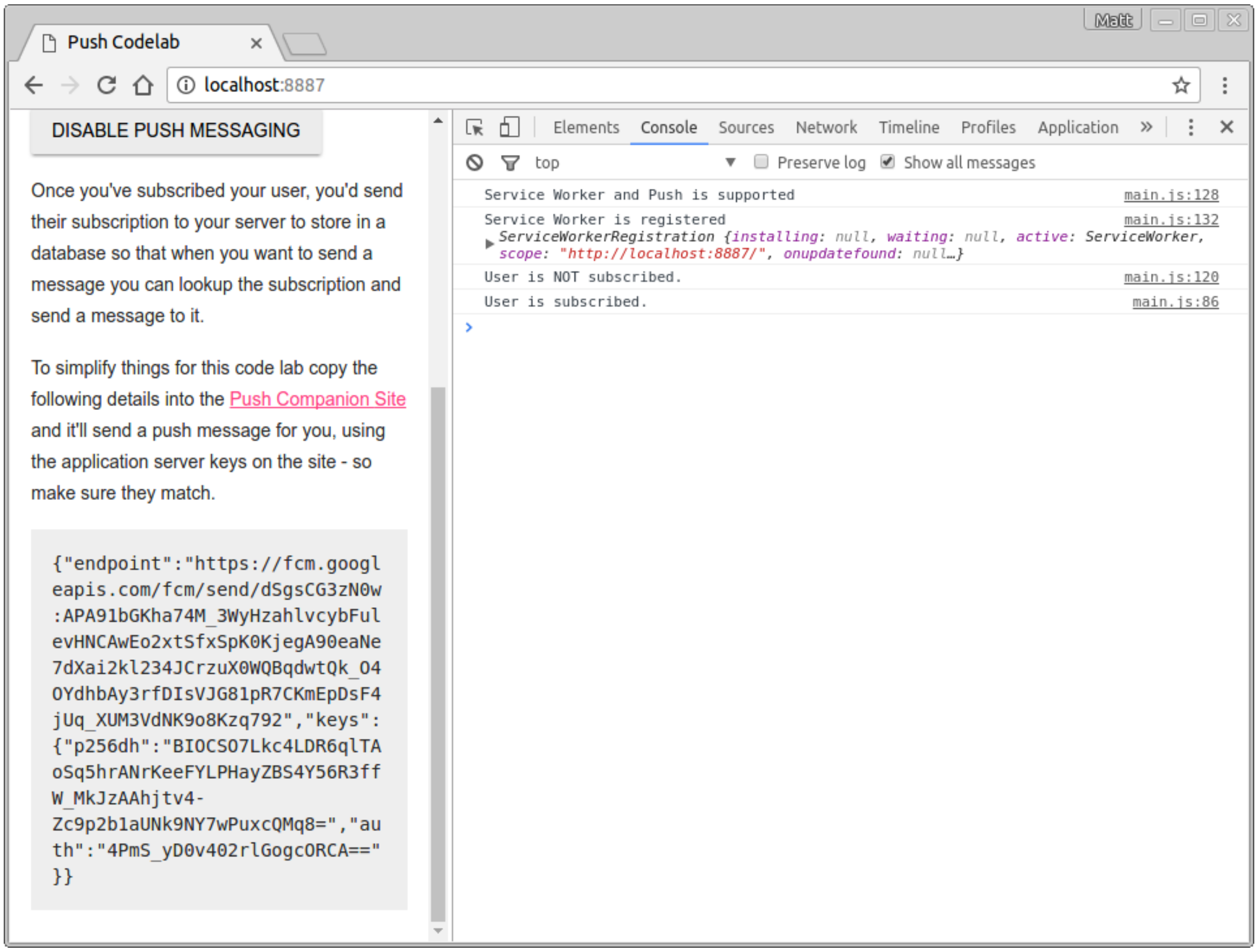
Try it out

If you go back to your web app and try clicking the button you should see a permission prompt like this:



```
and Push is supported                                          main.js:128
is registered                                                  main.js:132
egistration {installing: null, waiting: null, active: ServiceWorker,
/localhost:8887/", onupdatefound: null…}
User is NOT subscribed.                                        main.js:120
```

If you grant the permission you should see the console print "User is subscribed.", the button's text will change to ‘Disable Push Messaging' and you'll be able to view the subscription as JSON at the bottom of the page.



Once you've subscribed your user, you'd send their subscription to your server to store in a database so that when you want to send a message you can lookup the subscription and send a message to it.

To simplify things for this code lab copy the following details into the Push Companion Site and it'll send a push message for you, using the application server keys on the site - so make sure they match.

```
{"endpoint":"https://fcm.googl
eapis.com/fcm/send/dSgsCG3zN0w
:APA91bGKha74M_3WyHzahlvcybFul
evHNCAwEo2xtSfxSpK0KjegA90eaNe
7dXai2kl234JCrzuX0WQBqdwtQk_O4
OYdhbAy3rfDIsVJG81pR7CKmEpDsF4
jUq_XUM3VdNK9o8Kzq792","keys":
{"p256dh":"BIOCSO7Lkc4LDR6qlTA
oSq5hrANrKeeFYLPHayZBS4Y56R3ff
W_MkJzAAhjtv4-
Zc9p2b1aUNk9NY7wPuxcQMq8=","au
th":"4PmS_yD0v402rlGogcORCA=="
}}
```

```
Service Worker and Push is supported                                    main.js:128
Service Worker is registered                                            main.js:132
▶ ServiceWorkerRegistration {installing: null, waiting: null, active: ServiceWorker,
  scope: "http://localhost:8887/", onupdatefound: null…}
User is NOT subscribed.                                                 main.js:120
User is subscribed.                                                      main.js:86
>
```

Handle Permission Denied

One thing that we haven't handled yet is what happens if the user blocks the permission request. This needs some unique consideration because if the user blocks the permission, our web app will not be able to re-show the permission prompt and will not be able to subscribe the user, so we need to at least disable the push button so the user knows it can't be used.

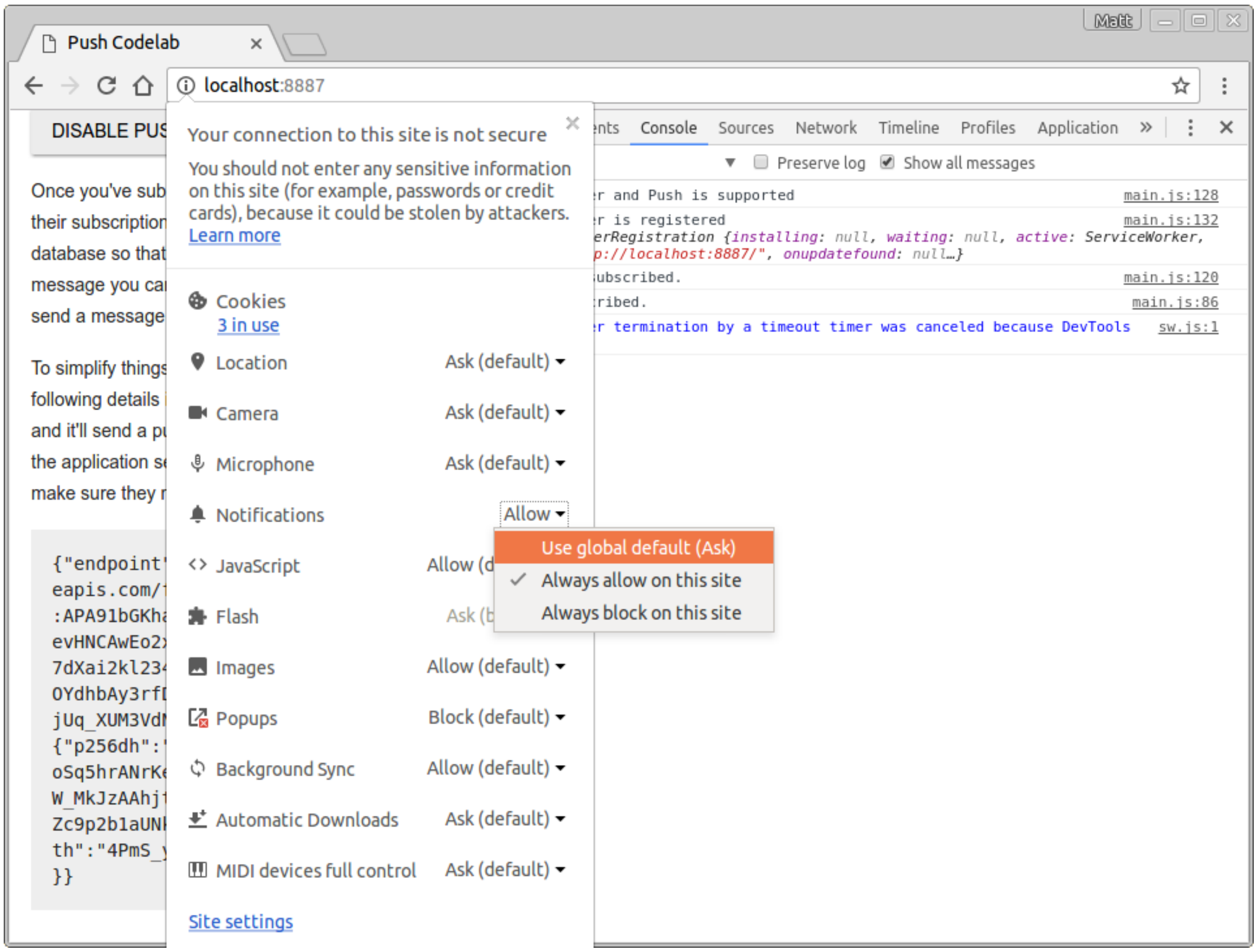
The obvious place for us to handle this scenario is in the `updateBtn()` function. All we need to do is check the `Notification.permission` value, like so:

```
function updateBtn() {
  if (Notification.permission === 'denied') {
    pushButton.textContent = 'Push Messaging Blocked.';
    pushButton.disabled = true;
    updateSubscriptionOnServer(null);
    return;
  }

  if (isSubscribed) {
    pushButton.textContent = 'Disable Push Messaging';
  } else {
    pushButton.textContent = 'Enable Push Messaging';
  }

  pushButton.disabled = false;
}
```

We know that if the permission is **`denied`**, then the user can't be subscribed and there is nothing more we can do, so disabling the button for good is the best approach.

Try it out

Since we've already granted permission for our web app from the previous step we need to click the **`i`** in a circle in the URL bar and change the notifications permission to *Use global default (Ask)* .



Once you've sub
their subscription
database so that
message you ca
send a message

To simplify things
following details
and it'll send a pu
the application s
make sure they r

```
{"endpoint"
eapis.com/f
:APA91bGKha
evHNCAwEo2x
7dXai2kl234
0YdhbAy3rfD
jUq_XUM3VdN
{"p256dh":"
oSq5hrANrKe
W_MkJzAAhj1
Zc9p2b1aUNh
th":"4PmS_y
}}
```

Your connection to this site is not secure

You should not enter any sensitive information on this site (for example, passwords or credit cards), because it could be stolen by attackers. Learn more

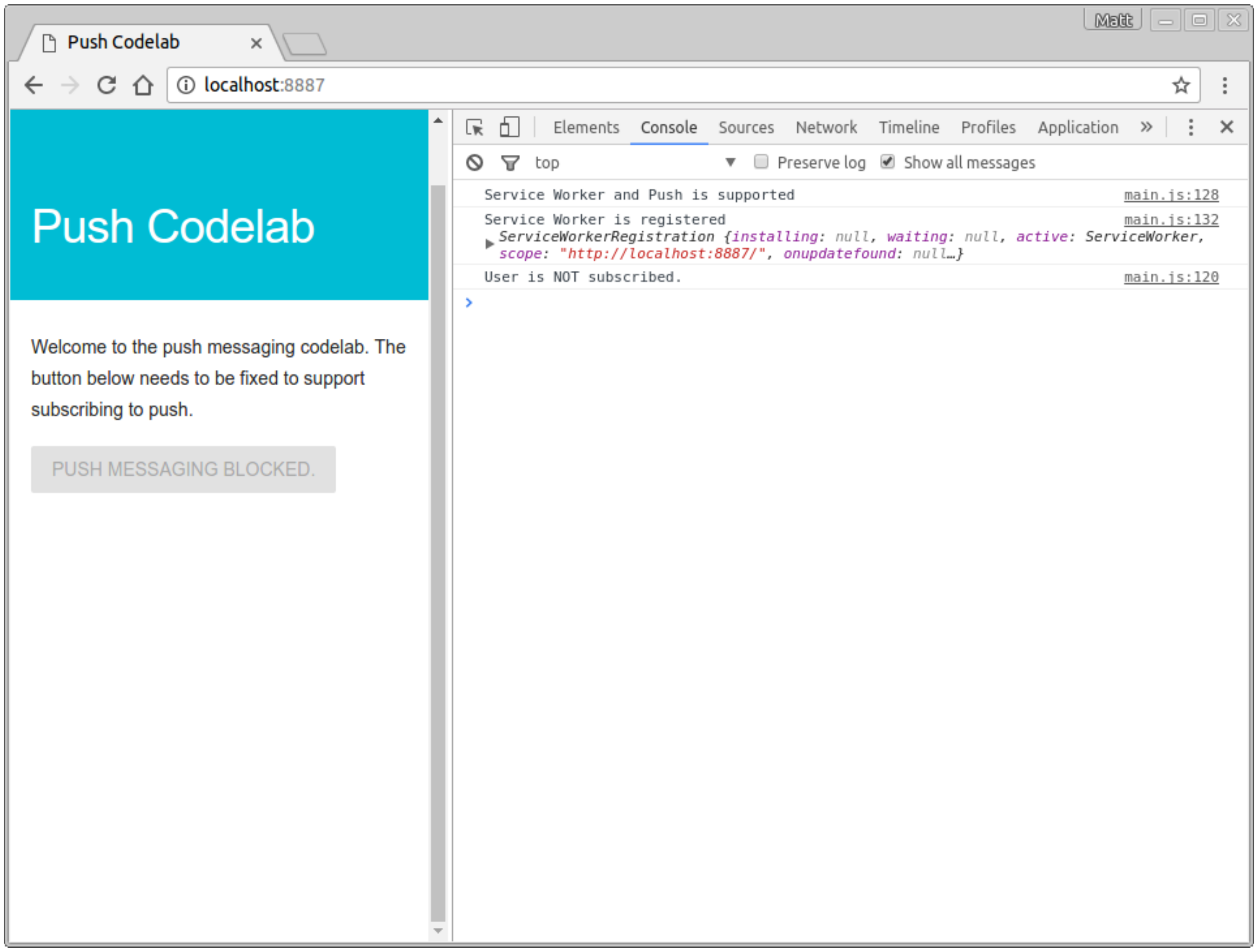
Cookies 3 in use	
Location	Ask (default)
Camera	Ask (default)
Microphone	Ask (default)
Notifications	Allow
JavaScript	Allow (d
Flash	Ask (b
Images	Allow (default)
Popups	Block (default)
Background Sync	Allow (default)
Automatic Downloads	Ask (default)
MIDI devices full control	Ask (default)

Use global default (Ask)
✓ Always allow on this site
Always block on this site

Site settings

```
r and Push is supported                                              main.js:128
r is registered                                                      main.js:132
erRegistration {installing: null, waiting: null, active: ServiceWorker,
p://localhost:8887/", onupdatefound: null…}
ubscribed.                                                           main.js:120
ribed.                                                                main.js:86
r termination by a timeout timer was canceled because DevTools        sw.js:1
```

After you've changed this setting, refresh the page and click the *Enable Push Messaging* button and this time select *Block* on the permission dialog. The button text will now be *Push Messaging Blocked* and be disabled.



Welcome to the push messaging codelab. The button below needs to be fixed to support subscribing to push.

```
Service Worker and Push is supported                                    main.js:128
Service Worker is registered                                            main.js:132
▶ServiceWorkerRegistration {installing: null, waiting: null, active: ServiceWorker,
 scope: "http://localhost:8887/", onupdatefound: null…}
User is NOT subscribed.                                                 main.js:120
```

With this change we can now subscribe the user and we're taking care of the possible permission scenarios.

Handle a Push Event

Before we cover how to send a push message from your backend, we need to consider what will actually happen when a subscribed user receives a push message.

When we trigger a push message, the browser receives the push message, figures out what service worker the push is for before waking up that service worker and dispatching a push event. We need to listen for this event and show a notification as a result.

Add the following code to your `sw.js` file:

```
self.addEventListener('push', function(event) {
  console.log('[Service Worker] Push Received.');
  console.log(`[Service Worker] Push had this data: "${event.data.text()}"`);

  const title = 'Push Codelab';
  const options = {
    body: 'Yay it works.',
    icon: 'images/icon.png',
    badge: 'images/badge.png'
  };

  event.waitUntil(self.registration.showNotification(title, options));
});
```

Let's step through this code. We are listening for push events in our service worker by adding an event listener to our service worker, which is this piece of code:

```
self.addEventListener('push', ...... );
```

Unless you've played with Web Workers before, **`self`** is probably new. **`self`** is referencing the service worker itself, so we are adding an event listener to our service worker.

When a push message is received, our event listener will be fired, and we create a notification by calling **`showNotification()`** on our registration. **`showNotification()`** expects a **`title`** and we can give it an **`options`** object. Here we are going to set a body message, icon and a badge in the options (the badge is only used on Android at the time of writing).

```
const title = 'Push Codelab';
const options = {
  body: 'Yay it works.',
  icon: 'images/icon.png',
  badge: 'images/badge.png'
};
self.registration.showNotification(title, options);
```

The last thing to cover in our push event is **`event.waitUntil()`**. This method takes a promise and the browser will keep your service worker alive and running until the promise passed in has resolved.

To make the code above a little easier to understand we can re-write it like so:

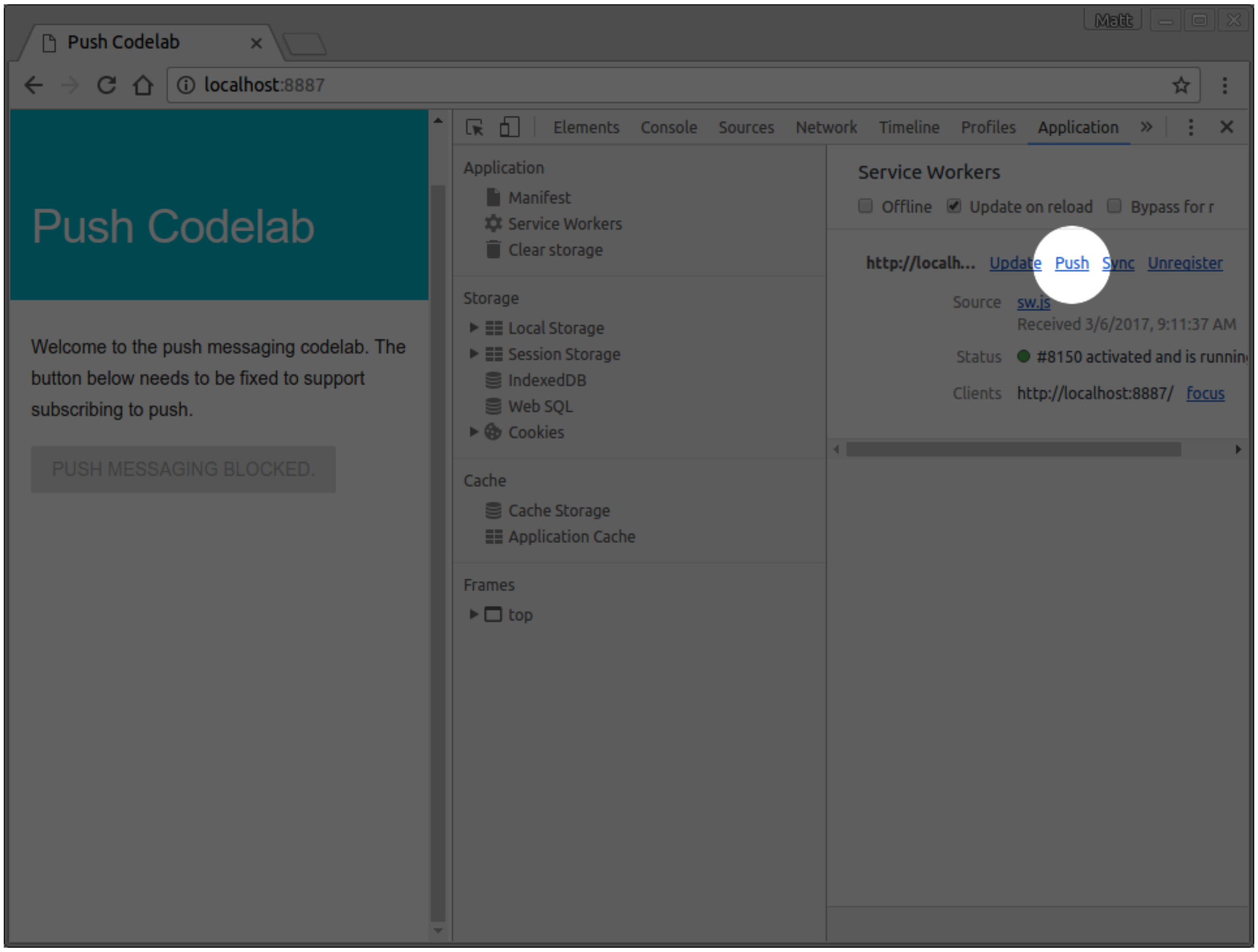
```
const notificationPromise = self.registration.showNotification(title, options);
event.waitUntil(notificationPromise);
```

Now that we've stepped through the push event, let's test out a push event.

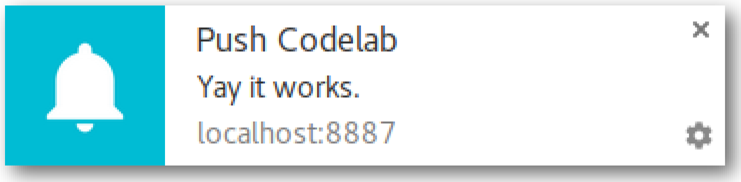
Try it out

With our push event in the service worker we can test what happens when a message is received by triggering a fake push event using DevTools.

In your web app, subscribe to push messaging, making sure you have *User IS subscribed* in your console, then go to the *Application* panel in DevTools and under the *Service Workers* tab click on the *Push* link under your service worker.


Push Codelab
localhost:8887
Push Codelab
Welcome to the push messaging codelab. The button below needs to be fixed to support subscribing to push.
PUSH MESSAGING BLOCKED
Elements Console Sources Network Timeline Profiles Application
Application
Manifest
Service Workers
Clear storage
Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies
Cache
Cache Storage
Application Cache
Frames
top
Service Workers
Offline Update on reload Bypass for r
http://localh... Update Push Sync Unregister
Source sw.js
Received 3/6/2017, 9:11:37 AM
Status #8150 activated and is runnin
Clients http://localhost:8887/ focus

Once you've clicked it you should see a notification like this:



Note: If this step doesn't work, try unregistering your service work, via the *Unregister* link in the DevTools Application panel, wait for the service worker to be stopped, and then reload the page.

Notification click

If you click on one of these notifications you'll notice nothing happens. We can handle notification clicks by listening for **`notificationclick`** events in your service worker.

Start by adding a **`notificationclick`** listener in **`sw.js`** like so:

```
self.addEventListener('notificationclick', function(event) {
  console.log('[Service Worker] Notification click Received.');

  event.notification.close();

  event.waitUntil(
    clients.openWindow('https://developers.google.com/web/')
  );
});
```

When the user clicks on the notification, the **`notificationclick`** event listener will be called.

In this code lab we first close the notification that was clicked with:

```
event.notification.close();
```

Then we open a new window / tab loading the url ‘https://developers.google.com/web/' , feel free to change this :)

```
clients.openWindow('https://developers.google.com/web/')
```

We are calling `event.waitUntil()` again to ensure the browser doesn't terminate our service worker before our new window has been displayed.

Try it out

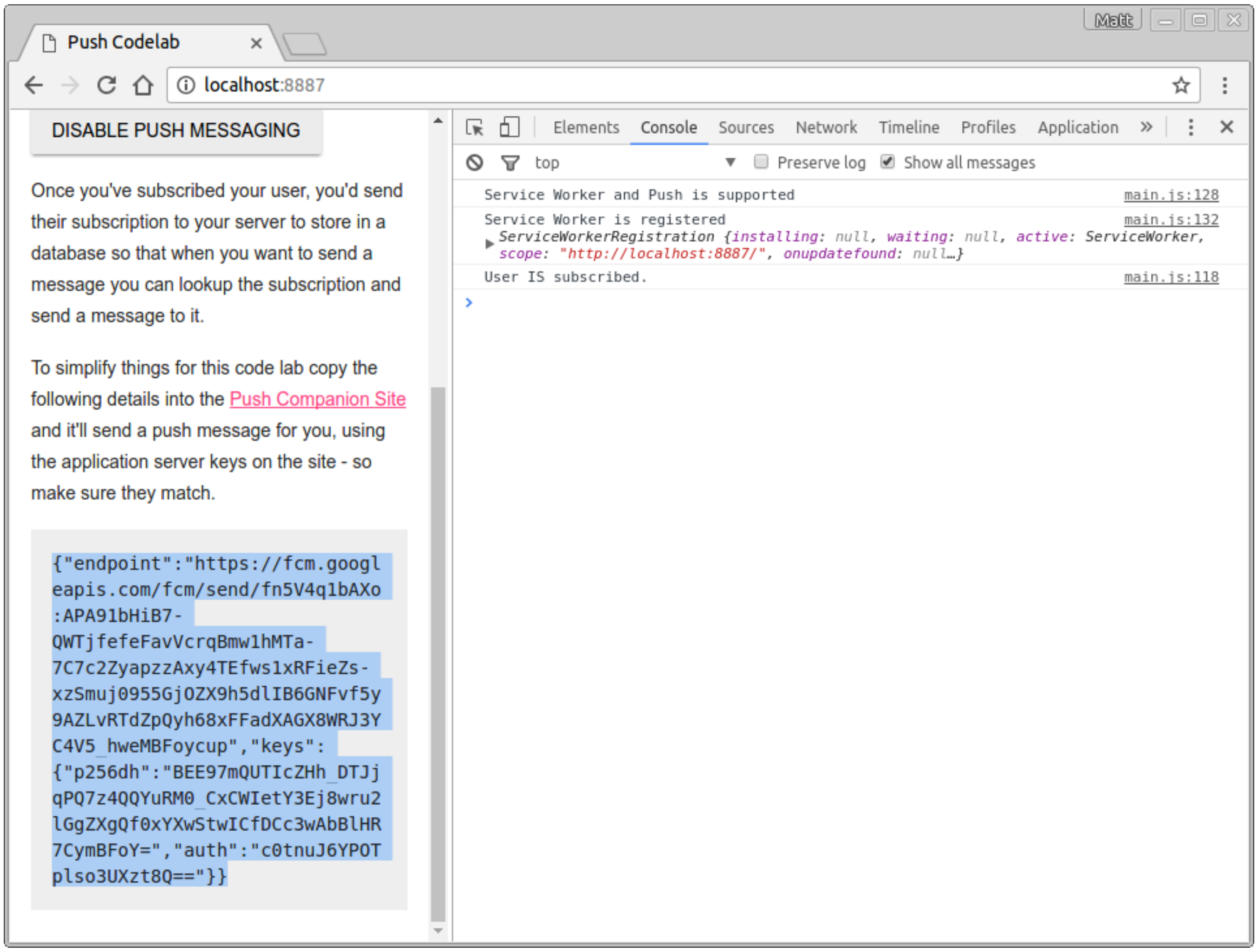
Try triggering a push message in DevTools again and click on the notification. You'll now see the notification close and open a new tab.

Sending push messages

We've seen that our web app is capable of showing a notification using DevTools and looked at how to close the notification of a click. The next step is to send an actual push message.

Normally the process for this would be sending a subscription from a web page to a backend and the backend would then trigger a push message by making an API call to the endpoint in the subscription.

This is out of scope for this codelab, but you can use the companion site (https://web-push-codelab.glitch.me/ (https://web-push-codelab.glitch.me/)) for this codelab to trigger an actual push message. Copy and paste the subscription at the bottom of your page:



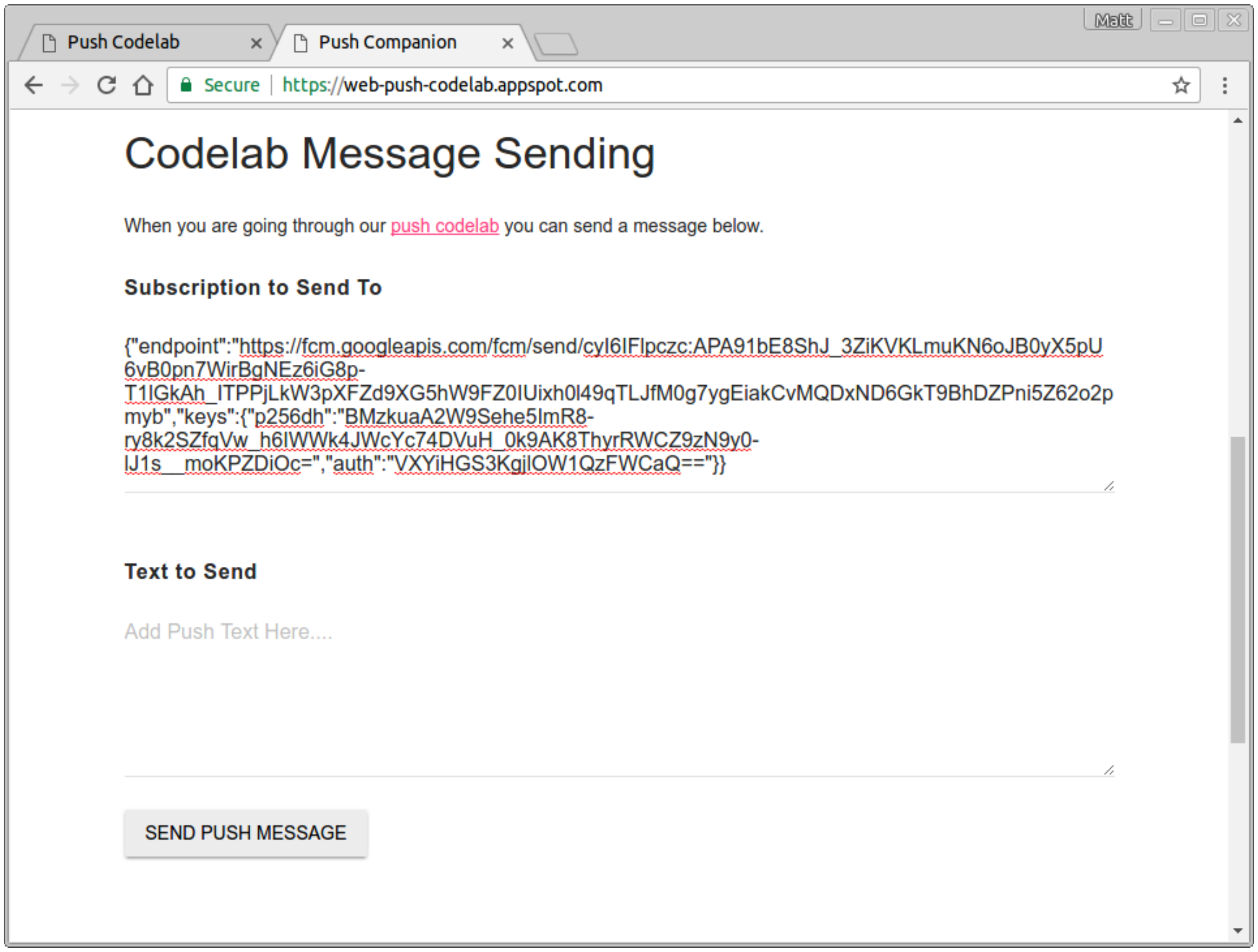
Once you've subscribed your user, you'd send their subscription to your server to store in a database so that when you want to send a message you can lookup the subscription and send a message to it.

To simplify things for this code lab copy the following details into the Push Companion Site and it'll send a push message for you, using the application server keys on the site - so make sure they match.

```
{"endpoint":"https://fcm.googleapis.com/fcm/send/fn5V4q1bAXo:APA91bHiB7-QWTjfefeFavVcrqBmw1hMTa-7C7c2ZyapzzAxy4TEfws1xRFieZs-xzSmuj0955GjOZX9h5dlIB6GNFvf5y9AZLvRTdZpQyh68xFFadXAGX8WRJ3YC4V5_hweMBFoycup","keys":{"p256dh":"BEE97mQUTIcZHh_DTJjqPQ7z4QQYuRM0_CxCWIetY3Ej8wru2lGgZXgQf0xYXwStwICfDCc3wAbBlHR7CymBFoY=","auth":"c0tnuJ6YPOTplso3UXzt8Q=="}}
```

```
Service Worker and Push is supported                                                     main.js:128
Service Worker is registered                                                             main.js:132
▶ ServiceWorkerRegistration {installing: null, waiting: null, active: ServiceWorker,
  scope: "http://localhost:8887/", onupdatefound: null…}
User IS subscribed.                                                                      main.js:118
>
```

Then paste this into the companion site in the *Subscription to Send To* text area:



Codelab Message Sending

When you are going through our push codelab you can send a message below.

Subscription to Send To

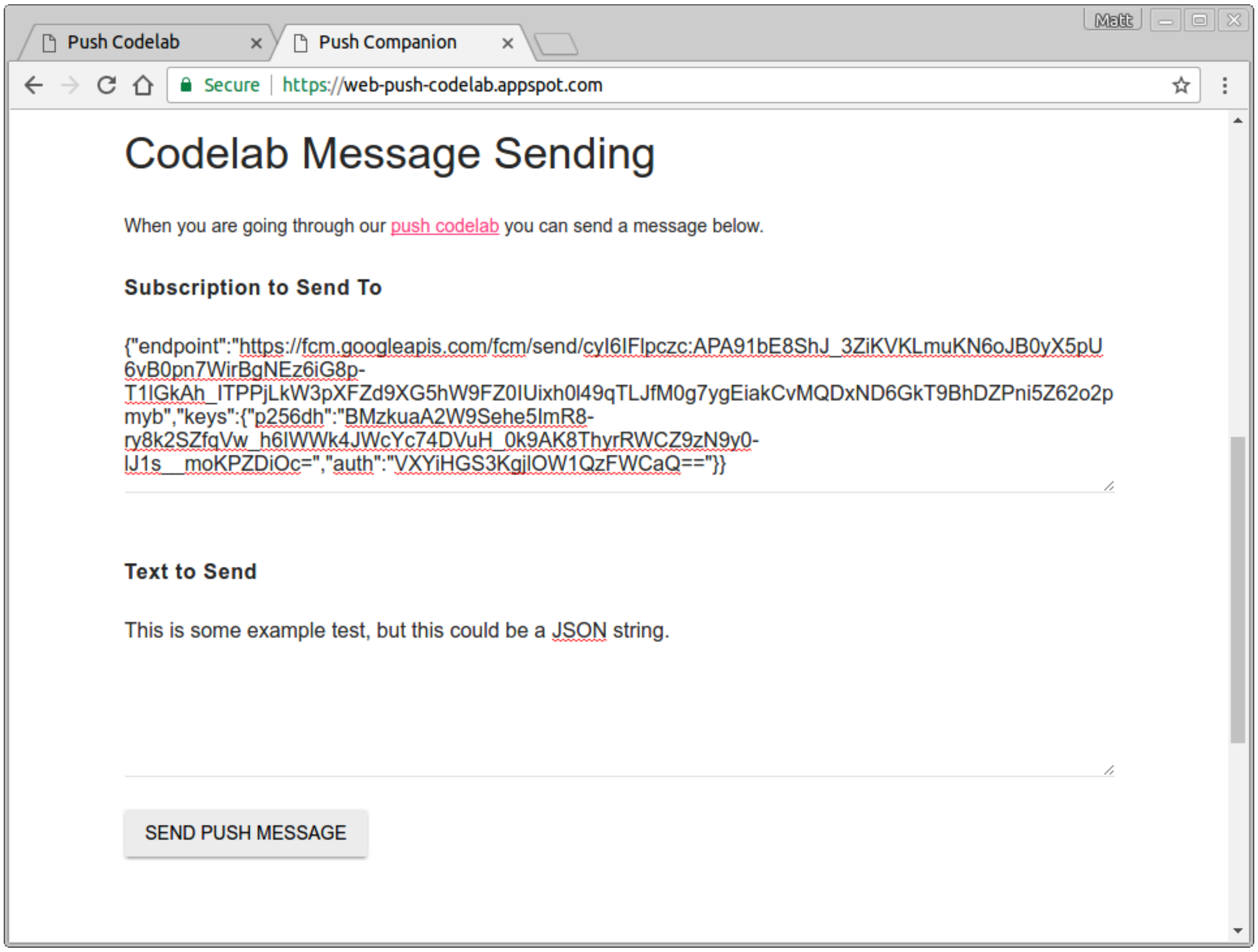
```
{"endpoint":"https://fcm.googleapis.com/fcm/send/cyI6IFlpczc:APA91bE8ShJ_3ZiKVKLmuKN6oJB0yX5pU
6vB0pn7WirBgNEz6iG8p-
T1IGkAh_ITPPjLkW3pXFZd9XG5hW9FZ0IUixh0l49qTLJfM0g7ygEiakCvMQDxND6GkT9BhDZPni5Z62o2p
myb","keys":{"p256dh":"BMzkuaA2W9Sehe5ImR8-
ry8k2SZfqVw_h6IWWk4JWcYc74DVuH_0k9AK8ThyrRWCZ9zN9y0-
lJ1s__moKPZDiOc=","auth":"VXYiHGS3KgjlOW1QzFWCaQ=="}}
```

Text to Send

Add Push Text Here....

SEND PUSH MESSAGE

Then under *Text to Send* you can add any string you want to send with the push message and finally click the *Send Push Message* button.



Codelab Message Sending

When you are going through our push codelab you can send a message below.

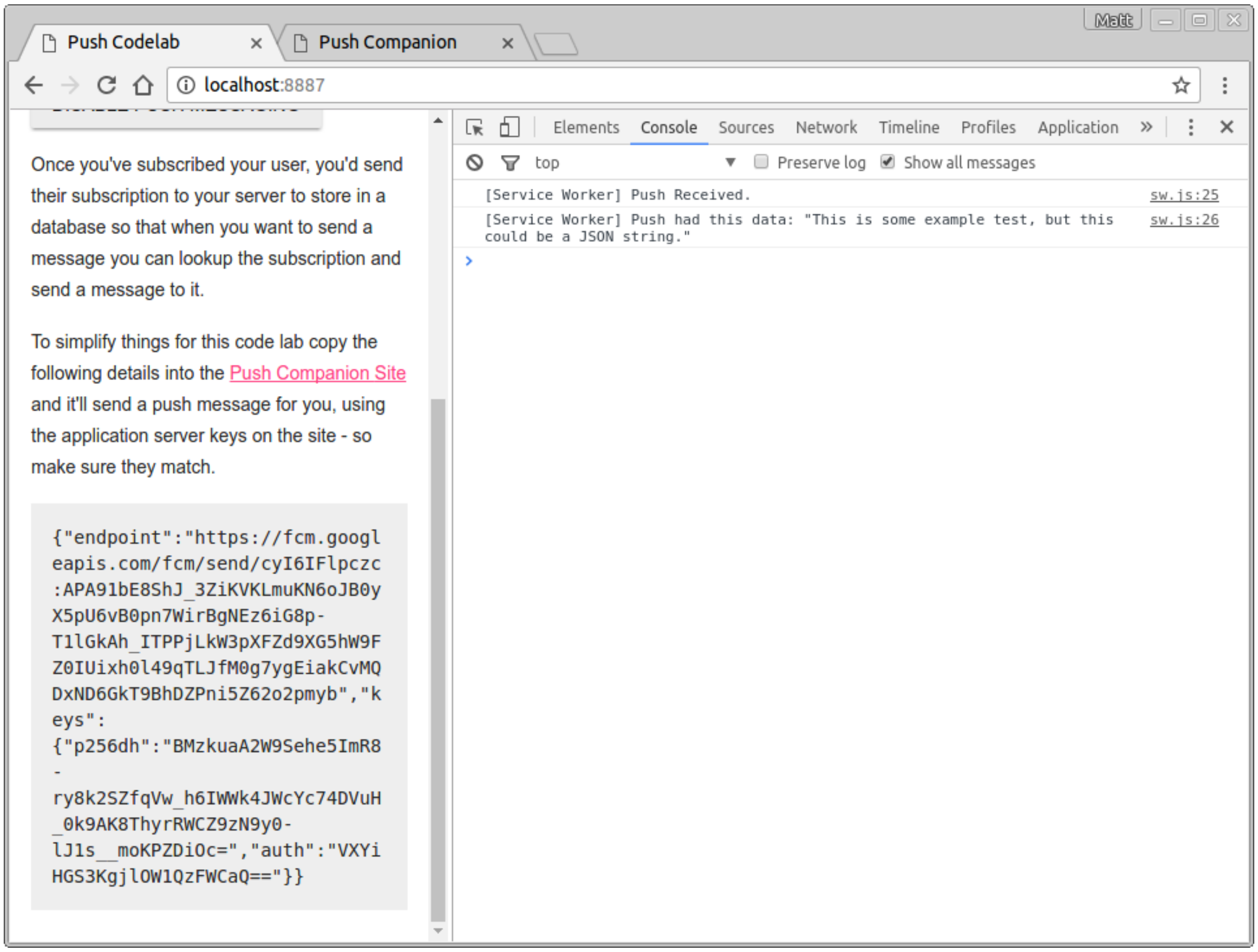
Subscription to Send To

```
{"endpoint":"https://fcm.googleapis.com/fcm/send/cyI6IFlpczc:APA91bE8ShJ_3ZiKVKLmuKN6oJB0yX5pU6vB0pn7WirBgNEz6iG8p-T1IGkAh_ITPPjLkW3pXFZd9XG5hW9FZ0IUixh0l49qTLJfM0g7ygEiakCvMQDxND6GkT9BhDZPni5Z62o2pmyb","keys":{"p256dh":"BMzkuaA2W9Sehe5ImR8-ry8k2SZfqVw_h6IWWk4JWcYc74DVuH_0k9AK8ThyrRWCZ9zN9y0-lJ1s__moKPZDiOc=","auth":"VXYiHGS3KgjlOW1QzFWCaQ=="}}
```

Text to Send

This is some example test, but this could be a JSON string.

You should then receive a push message and the text you included will be printed to the console.



Once you've subscribed your user, you'd send their subscription to your server to store in a database so that when you want to send a message you can lookup the subscription and send a message to it.

To simplify things for this code lab copy the following details into the Push Companion Site and it'll send a push message for you, using the application server keys on the site - so make sure they match.

```
{"endpoint":"https://fcm.googleapis.com/fcm/send/cyI6IFlpczc:APA91bE8ShJ_3ZiKVKLmuKN6oJB0yX5pU6vB0pn7WirBgNEz6iG8p-T1lGkAh_ITPPjLkW3pXFZd9XG5hW9FZ0IUixh0l49qTLJfM0g7ygEiakCvMQDxND6GkT9BhDZPni5Z62o2pmyb","keys":{"p256dh":"BMzkuaA2W9Sehe5ImR8-ry8k2SZfqVw_h6IWWk4JWcYc74DVuH_0k9AK8ThyrRWCZ9zN9y0-lJ1s__moKPZDiOc=","auth":"VXYiHGS3KgjlOW1QzFWCaQ=="}}
```

```
[Service Worker] Push Received.                                          sw.js:25
[Service Worker] Push had this data: "This is some example test, but this   sw.js:26
could be a JSON string."
```

This should give you a chance to test out sending and receiving data and manipulate notifications as a result.

The companion app is actually just a node server that is using the web-push library (https://github.com/web-push-libs/web-push) to send messages. It's worthwhile checking out the web-push-libs org on Github (https://github.com/web-push-libs/) to see what libraries are available to send push messages for you (this handles a lot of the nitty gritty details to trigger push messages).

You can see all the code for the companion site here (https://glitch.com/edit/#!/web-push-codelab).

Unsubscribe the user

The one thing we are missing is the ability to unsubscribe the user from push. To do this we need to call `unsubscribe()` on a `PushSubscription`.

Back in our `scripts/main.js` file, change the `pushButton`'s click listener in `initializeUI()` to the following:

```
pushButton.addEventListener('click', function() {
  pushButton.disabled = true;
  if (isSubscribed) {
    unsubscribeUser();
  } else {
    subscribeUser();
  }
});
```

Notice we are now going to call a new function `unsubscribeUser()`. In this method we'll get the current subscription and called unsubscribe on it. Add the following code to `scripts/main.js`:

```
function unsubscribeUser() {
  swRegistration.pushManager.getSubscription()
  .then(function(subscription) {
    if (subscription) {
      return subscription.unsubscribe();
    }
  })
  .catch(function(error) {
    console.log('Error unsubscribing', error);
  })
  .then(function() {
    updateSubscriptionOnServer(null);

    console.log('User is unsubscribed.');
    isSubscribed = false;

    updateBtn();
  });
}
```

Let's step through this function.

First we get the current subscription by calling `getSubscription()`:

```
swRegistration.pushManager.getSubscription()
```

This returns a promise that resolves with a `PushSubscription` if one exists, otherwise it returns `null`. If there is a subscription, we call `unsubscribe()` on it, which makes the `PushSubscription` invalid.

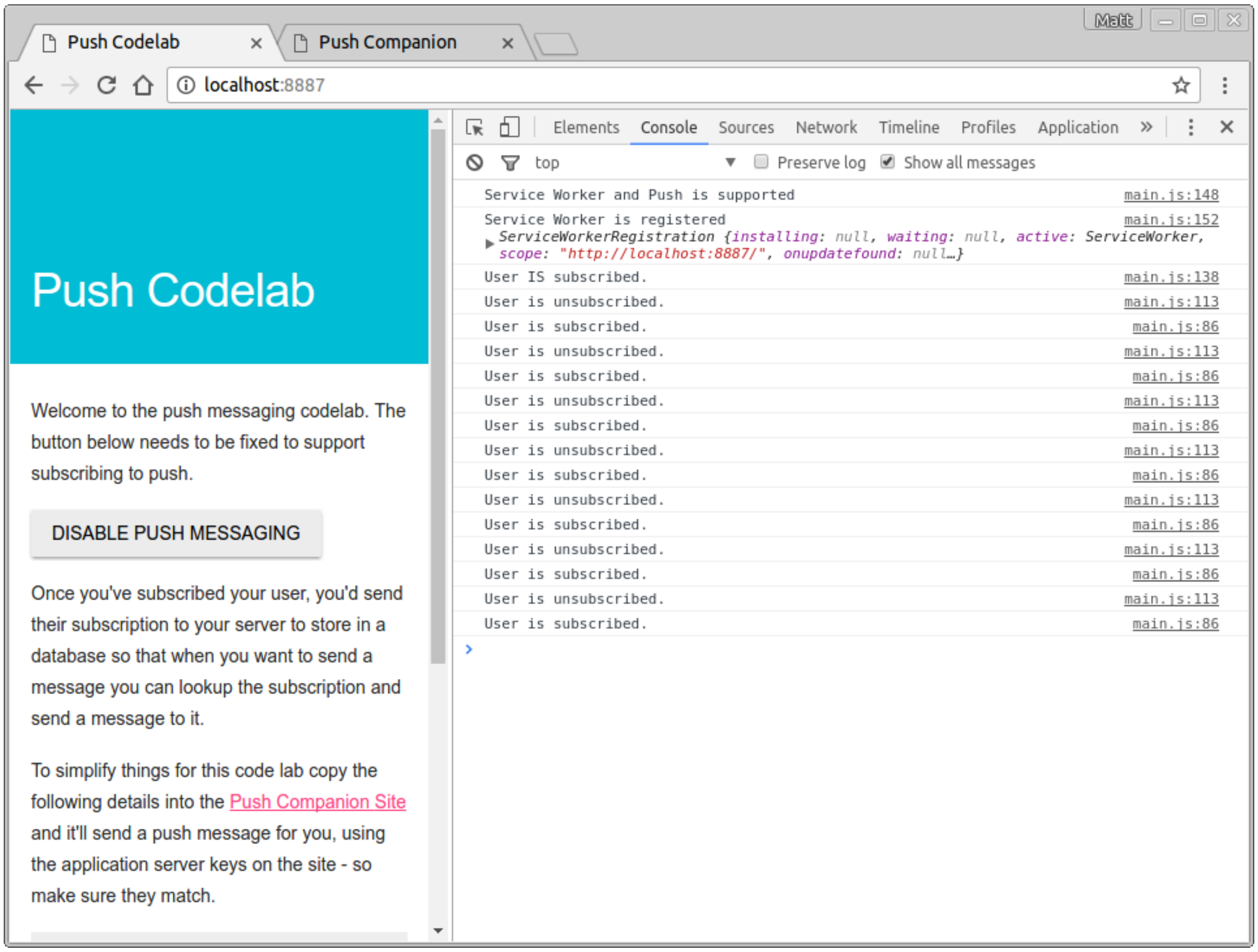
```
swRegistration.pushManager.getSubscription()
.then(function(subscription) {
  if (subscription) {
    // TODO: Tell application server to delete subscription
    return subscription.unsubscribe();
  }
})
.catch(function(error) {
  console.log('Error unsubscribing', error);
})
```

Calling `unsubscribe()` returns a promise as it can take some time to complete, so we return that promise so the next `then()` in the chain waits for `unsubscribe()` to finish. We also add a catch handler in case calling `unsubscribe()` results in an error. After this we can update our UI.

```
.then(function() {
  updateSubscriptionOnServer(null);

  console.log('User is unsubscribed.');
  isSubscribed = false;

  updateBtn();
})
```

Try it out

You should be able to press the *Enable Push Messaging* / *Disable Push Messaging* in your web app and the logs will show the user being subscribed and unsubscribed.



Push Codelab

Welcome to the push messaging codelab. The button below needs to be fixed to support subscribing to push.

Once you've subscribed your user, you'd send their subscription to your server to store in a database so that when you want to send a message you can lookup the subscription and send a message to it.

To simplify things for this code lab copy the following details into the Push Companion Site and it'll send a push message for you, using the application server keys on the site - so make sure they match.

```
Service Worker and Push is supported                                    main.js:148
Service Worker is registered                                            main.js:152
▸ServiceWorkerRegistration {installing: null, waiting: null, active: ServiceWorker,
 scope: "http://localhost:8887/", onupdatefound: null…}
User IS subscribed.                                                     main.js:138
User is unsubscribed.                                                   main.js:113
User is subscribed.                                                      main.js:86
User is unsubscribed.                                                   main.js:113
User is subscribed.                                                      main.js:86
User is unsubscribed.                                                   main.js:113
User is subscribed.                                                      main.js:86
User is unsubscribed.                                                   main.js:113
User is subscribed.                                                      main.js:86
User is unsubscribed.                                                   main.js:113
User is subscribed.                                                      main.js:86
User is unsubscribed.                                                   main.js:113
User is subscribed.                                                      main.js:86
User is unsubscribed.                                                   main.js:113
User is subscribed.                                                      main.js:86
>
```

Finished

Congratulations on completing this codelab!

This code lab has shown you how to get up and running with adding push to your web app. If you want to learn more about what web notifications can do, check out theses docs (https://developers.google.com/web/fundamentals/push-notifications).

If you are looking to deploy push on your site, you may be interested in adding support for older / non-standards compliant browsers which use GCM, learn more here (https://web-push-book.gauntface.com/chapter-06/01-non-standards-browsers/).

Further Reading

- Web Push Notification (https://developers.google.com/web/fundamentals/push-notifications) documentation on Web**Fundamentals**.
- Web Push Libraries (https://github.com/web-push-libs/) - Web Push libraries including Node.js, PHP, Java and Python.

Relevant blog posts

- Web Push Payload Encryption (https://developers.google.com/web/updates/2016/03/web-push-encryption)
- Application Server Keys and Web Push (https://developers.google.com/web/updates/2016/07/web-push-interop-wins)
- Notification Actions (https://developers.google.com/web/updates/2016/01/notification-actions)
- Icons, Close Events, Renotify Preferences and Timestamps (https://developers.google.com/web/updates/2016/03/notifications)

Found an issue, or have feedback?

Help us make our code labs better by submitting an issue (https://github.com/GoogleChromeLabs/web-push-codelab/issues) today. And thanks!

Chromium Blog
The latest news on the Chromium blog.

GitHub
Fork our code samples and other open-source projects.

Twitter
Connect with @ChromiumDev on Twitter.

Videos
Check out our videos.

Events
Attend a developer event and get hacking.