ADD PUSH NOTIFICATIONS TO YOUR WEB APP WITH NODE.JS AND SERVICE WORKERS

Ayooluwa Isaiah

February 5th, 2019

You will need Node 6+ installed on your machine.

In this tutorial, you'll learn how to use service workers and the web-push library to provide push notifications in the browser. You'll learn how to subscribe the user for push messaging and how to deliver rich notifications in the browser.

Push notifications have a place in many web apps to day. They help re-engage users and draw their attention to new activity that occurs in your web application.

Prerequisites

You need basic knowledge of HTML, CSS, JavaScript and Node. You also need a modern web browser that supports service workers (the latest editions of Chrome or Firefox will suffice). Lastly, you need to have Node.js (version 6 or later) and npm installed on your computer. You can view installation instructions here.

Getting started

The first thing you need to do is launch the terminal program on your computer and create a new directory for this project. Then run `npm init -y` from within your project to initialize a new Node project.

The next step is to install the dependencies which we'll be making use of for this project. Run the command below to install them all in one step:

```
npm install dotenv body-parser express web-push -S
```

Here's what each one does:

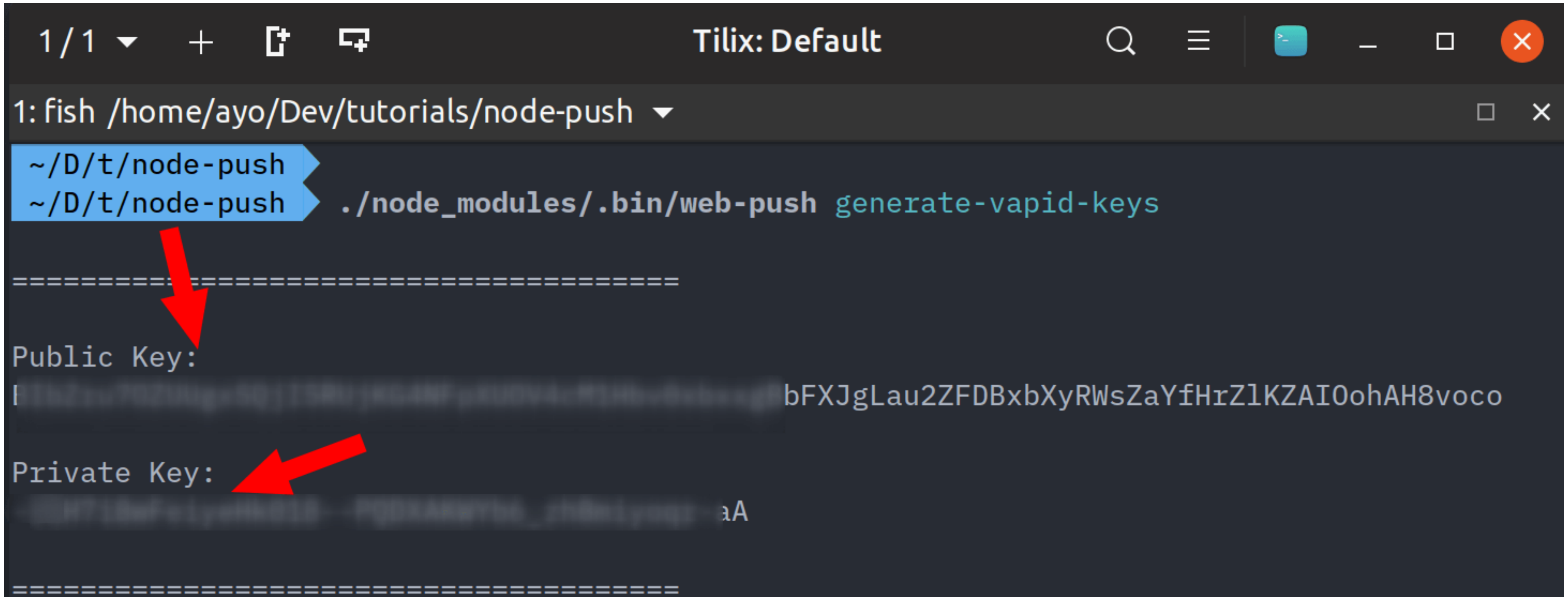
- dotenv: Loads environmental variables from a `.env` file to `process.env`.
- express: Web application framework for Node.
- body-parser: Middleware for parsing incoming request bodies.
- web-push: Web Push library for Node.js.

Generate VAPID Keys

The `web-push` library relies on a set of VAPID keys to work. VAPID keys are a pair of public and private keys which is used to restrict the validity of a push subscription to a specific application server, and also to identify the server that is sending the push notifications.

You can generate the VAPID key pair by running the command below from the root of your project directory:

```
./node_modules/.bin/web-push generate-vapid-keys
```



```
~/D/t/node-push
~/D/t/node-push  ./node_modules/.bin/web-push generate-vapid-keys

=======================================

Public Key:
I[redacted]bFXJgLau2ZFDBxbXyRWsZaYfHrZlKZAIOohAH8voco

Private Key:
[redacted]aA

=======================================
```

```
~/D/t/node-push
```

Copy the public and private key and paste them into a new `variables.env` file in the root of your project directory as shown below:

```
// variables.env
PORT=5000
PUBLIC_VAPID_KEY=<your public key>
PRIVATE_VAPID_KEY=<your private key>
```

Set up the server

Create a new `server.js` file in your project directory. Open it up in your text editor and paste the following code into it:

```
// server.js
require('dotenv').config({ path: 'variables.env' });

const express = require('express');
const webPush = require('web-push');
const bodyParser = require('body-parser');
const path = require('path');

const app = express();
```

```
app.use(bodyParser.json());

app.use(express.static(path.join(__dirname, 'client')));

const publicVapidKey = process.env.PUBLIC_VAPID_KEY;
const privateVapidKey = process.env.PRIVATE_VAPID_KEY;

webPush.setVapidDetails('mailto:test@example.com', publicVapidKey, privateVapidKey);

app.post('/subscribe', (req, res) => {
  const subscription = req.body

  res.status(201).json({});

  const payload = JSON.stringify({
    title: 'Push notifications with Service Workers',
  });

  webPush.sendNotification(subscription, payload)
    .catch(error => console.error(error));
});

app.set('port', process.env.PORT || 5000);
const server = app.listen(app.get('port'), () => {
  console.log(`Express running → PORT ${server.address().port}`);
});
```

We are able to access our public and private VAPID keys on `process.env` thanks to our use of the `dotenv` package. We then call the `setVapidDetails()` method which takes an email address for our website as well as the public/private key pair we generated earlier. The `/subscribe` route at the bottom is we're triggering the push notification event to the service worker generated earlier. The `/subscribe` route at the bottom is we're triggering the push notification event to the service worker.

That's all we need to do on the server side. You can start the server by running `node server.js` in the terminal. A message will be printed on the screen informing you that the server was started successfully.

Set up the client

Create a new `client` directory within your project directory. This is where all our static files will be kept. Create the following files from within the `client` directory by running the command below:

```
touch index.html style.css main.js sw.js
```

Open the `index.html` file in your text editor and paste in the following code:

```
// client/index.html

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Push Notifications using Node.js and Web Push</title>
  <link rel="icon" type="image/png" href="https://d2cy1obokpvee9.cloudfront.net/manifest/favicon-196x196.png" si
  <link rel="stylesheet" href="style.css">
</head>
<body>
```

```
  <header>
    <h1>Web Push Notifications Demo</h1>
  </header>

  <div class="buttons">
    <button class="trigger-push">Trigger Push Notification</button>
  </div>

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

Let’s go ahead and add the styles for this page in `style.css`:

```
// client/style.css

html {
  box-sizing: border-box;
}

*, *::before, *::after {
  box-sizing: inherit;
  margin: 0;
  padding: 0;
}

body {
  font-family: "Roboto", "HelveticaNeue", "Helvetica Neue", Helvetica, Arial, sans-serif;
}
```

```
header {
  width: 100%;
  height: 300px;
  display: flex;
  justify-content: center;
  align-items: center;
  background-color: #33154F;
  color: #11D771;
  margin-bottom: 30px;
}

.buttons {
  text-align: center;
}

button {
  display: inline-block;
  height: 38px;
  padding: 0 30px;
  color: #fff;
  text-align: center;
  font-size: 11px;
  font-weight: 600;
  line-height: 38px;
  letter-spacing: .1rem;
  text-transform: uppercase;
  text-decoration: none;
  white-space: nowrap;
  background-color: transparent;
  border-radius: 4px;
  border: 1px solid #bbb;
  cursor: pointer;
```

```
    box-sizing: border-box;
  }

  button:hover {
    outline: 0;
  }

  .trigger-push {
    background-color: #073525;
    border: 1px solid #073525;
  }
```

Now, let's write the main logic for in the `main.js` file. Open it up and paste the following code inside:

```
  // client/main.js

  function urlBase64ToUint8Array(base64String) {
    const padding = '='.repeat((4 - base64String.length % 4) % 4);
    const base64 = (base64String + padding)
      .replace(/-/g, '+')
      .replace(/_/g, '/');

    const rawData = window.atob(base64);
    const outputArray = new Uint8Array(rawData.length);

    for (let i = 0; i < rawData.length; ++i) {
      outputArray[i] = rawData.charCodeAt(i);
    }
    return outputArray;
  }
```

```
const publicVapidKey = '<your public vapid key>';

const triggerPush = document.querySelector('.trigger-push');

async function triggerPushNotification() {
  if ('serviceWorker' in navigator) {
    const register = await navigator.serviceWorker.register('/sw.js', {
      scope: '/'
    });

    const subscription = await register.pushManager.subscribe({
      userVisibleOnly: true,
      applicationServerKey: urlBase64ToUint8Array(publicVapidKey),
    });

    await fetch('/subscribe', {
      method: 'POST',
      body: JSON.stringify(subscription),
      headers: {
        'Content-Type': 'application/json',
      },
    });
  } else {
    console.error('Service workers are not supported in this browser');
  }
}

triggerPush.addEventListener('click', () => {
  triggerPushNotification().catch(error => console.error(error));
});
```

As you can see, the code is straightforward to understand. Once the `triggerPush` button is clicked, we register the service worker file `sw.js`, and create a subscription which then prompts the user to allow notifications for the current page. When using your public VAPID key in your web app, you'll need to convert the URL safe base64 string to a Uint8Array to pass into the subscribe call, which you can do by passing the key to the `urlBase64ToUint8Array()` function as shown above. Don't forget to replace `<your public vapid key>` with the appropriate value from your VAPID credentials.

The request to the `/subscribe` route we created earlier subsequently triggers a new push event. Now, we need to listen for this event on the service worker and show a notification to the user each time this event is triggered.

Open up `sw.js` and change it to look like this:

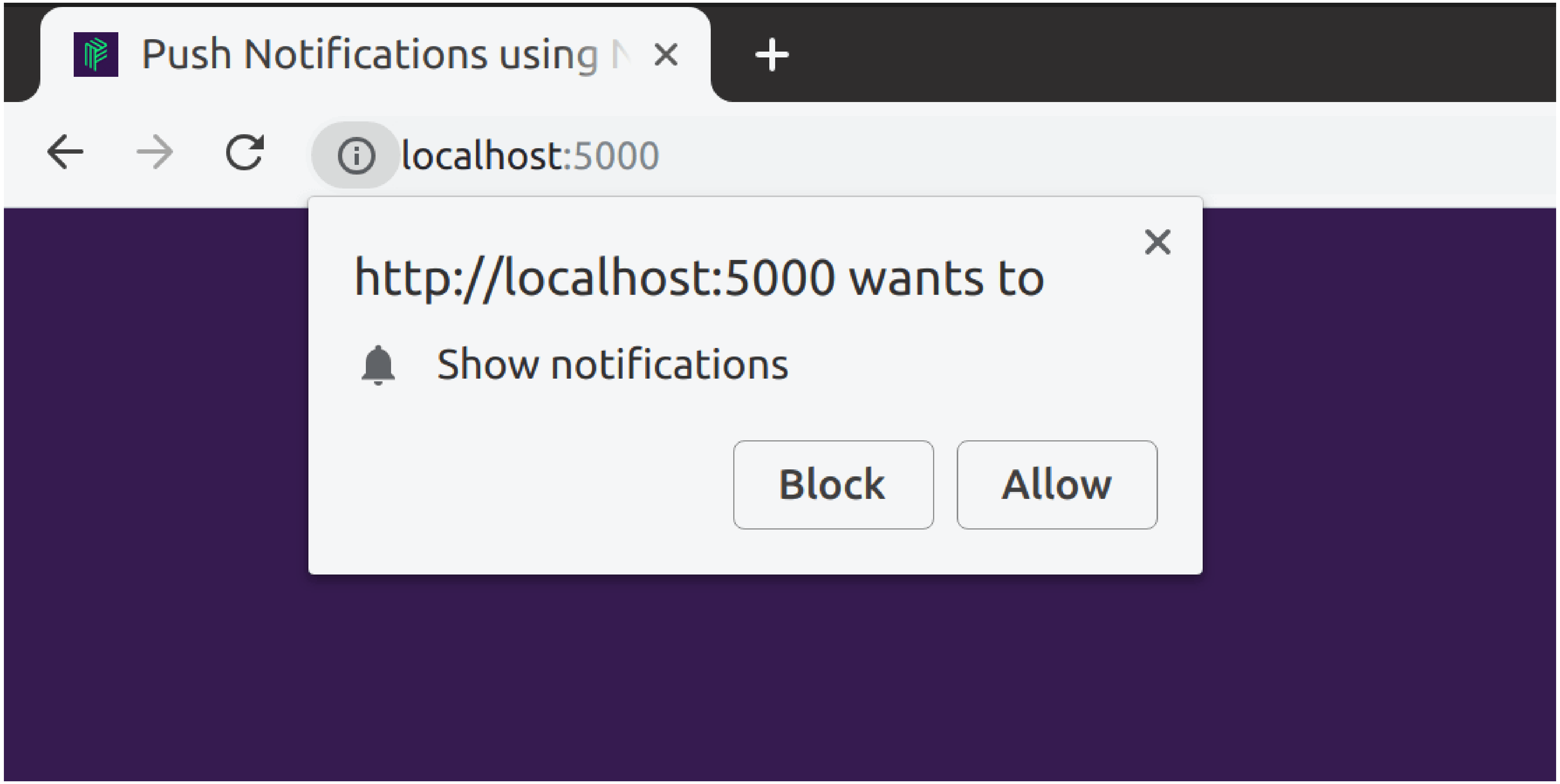
```
// client/sw.js

self.addEventListener('push', event => {
  const data = event.data.json();

  self.registration.showNotification(data.title, {
    body: 'Yay it works!',
  });
});
```

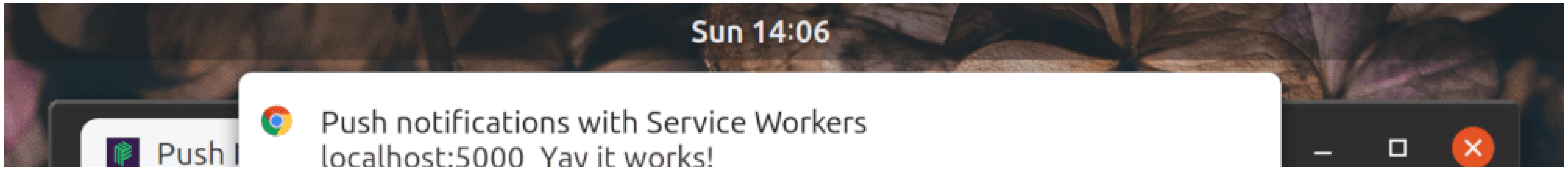
When a push event is received, the event listener above will be invoked, and we create a notification by calling `showNotification()` on our registration. This method expects a title and and options object. The title is grabbed off the event payload, and we've set the body of the notification using the `body` property in the options object.

To test if it's working, open up http://localhost:5000 in your browser and click the button on the page. Once you accept the prompt that allows notifications to be displayed, you should see a new notification somewhere on your screen depending on how your operating system handles notifications.

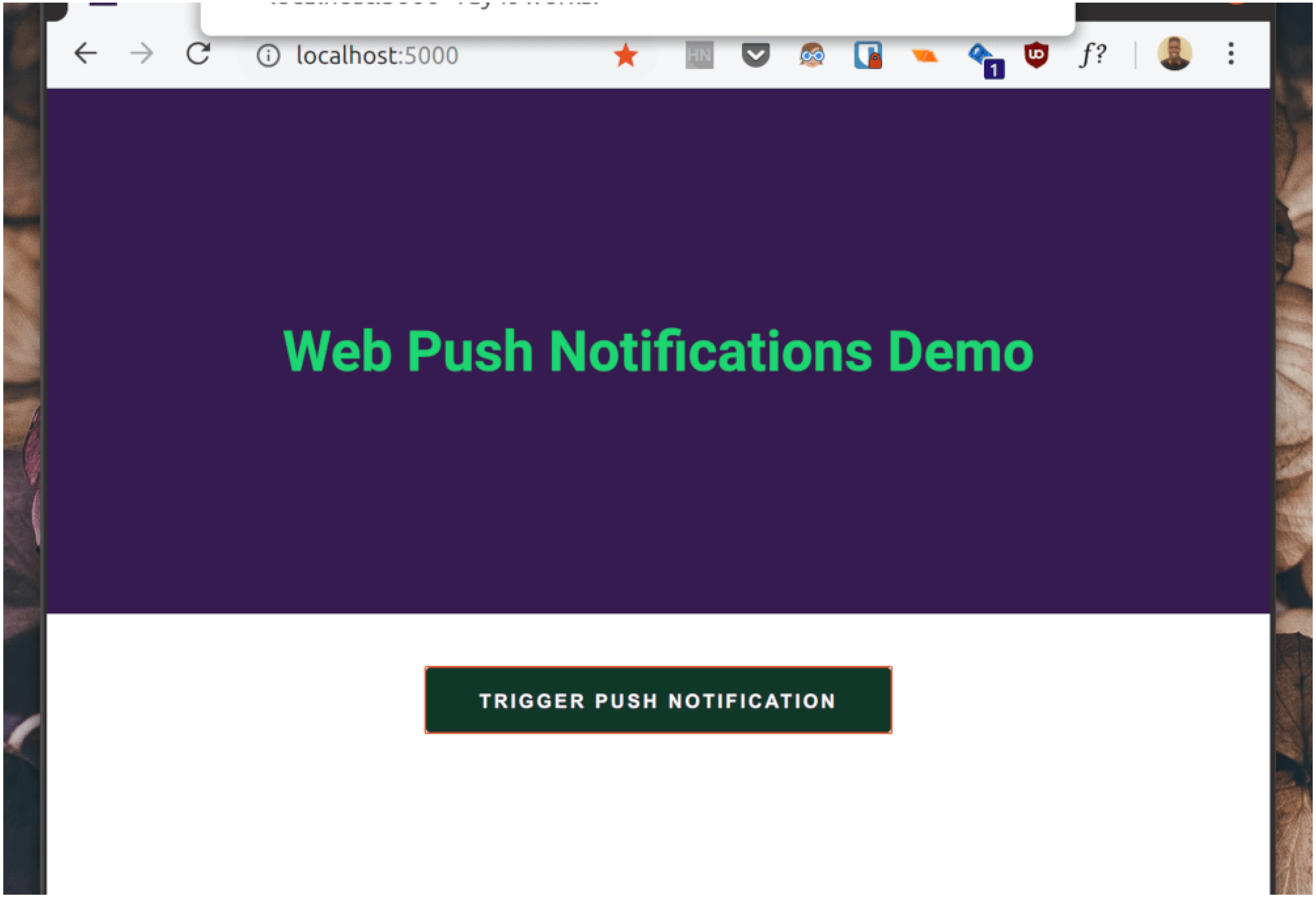


http://localhost:5000 wants to

Show notifications



Push notifications with Service Workers
localhost:5000 Yay it works!


localhost:5000
Web Push Notifications Demo
TRIGGER PUSH NOTIFICATION

Wrap up

This tutorial has shown you how to get up and running with push notifications on the web using service workers and Web Push. You can grab the complete source code in this GitHub repository.

If you're interested in deploying push notifications to an Android or iOS app, you can check out Beams by Pusher which takes the hassle out of managing device tokens and interacting with Apple and Google's messaging services and is built to scale.

NODE.JS CSS HTML JAVASCRIPT

NO PUSHER TECH