Open in app ↗

Search Medium

This member-only story is on us. Upgrade to access all of Medium.

✨ Member-only story

Angular’s PWA: SwPush and SwUpdate

Arjen Brandenburgh · Follow

10 min read · Apr 15, 2019

Listen Share More

Not too long ago, I wrote an article Your Angular app as Progressive Web App where I explained why Progressive Web Apps are the future of web applications. In that article I also briefly touched the Angular SwPush and SwUpdate APIs, where I also mentioned I might write an additional article to go deeper into that subject. A promise is a promise, so here it is! I’ll go a bit deeper into the possibilities these two APIs have to offer and show some examples. SwPush and SwUpdate are both conveniently exposed in the same package `@angular/service-worker` , which we’re obviously going to use.

VAPID keys

So before we start, I have to mention that this article is going to mainly focus on the frontend part of things. That being said, we do need a little bit of background information about the backend to be able to create an entire working system. As you're probably aware, a push message (notification) gets sent from a server somewhere that our web app will act on. The authenticity of these messages can be verified by an encrypted key-pair, making sure that the notifications our app shows are actually coming from us. This key-pair is called VAPID and stands for *The Voluntary Application Server Identification for Web Push*. When we generate this, we will end up with a public- and private key. Generating your own VAPID keys is so easy, you could have done it in less time than you've spent reading this paragraph. You only need two simple commands:

```
npm install web-push -g
web-push generate-vapid-keys
```

SwPush

Not too long ago Angular started supporting push notifications through SwPush, which allowed developers to listen and subscribe to, push notification from the service worker. However, it was lacking the possibility to interact with these notifications. So users were able to receive notifications, but clicking on it simply didn't do anything.

This fortunately changed around Angular version 7.1 where we can now add actions to our notifications. Before we are able to send notifications, users have to opt-in to your website's notifications. We can request this by using the SwPush's `requestSubscription`. To demonstrate this we're going to create a component `NotificationComponent` that shows a button. When a user clicks it, we're going to try to subscribe that user to show notifications. This component will use our own service `WebNotificationService` to actually submit the subscription to our server.

The component we're going to use has a single interaction element: a button, which when clicked will create a subscription and send this to our backend. This button is where our users will start the web-push journey.

```
import { Component } from '@angular/core';
import { SwPush } from '@angular/service-worker';

import { WebNotificationService } from './../services/web-
notification.service';

@Component({
  selector: 'app-notification-button',
  template: `
    <button (click)="submitNotification()">
       Notify me
    </button>
    <p *ngIf="isGranted" class="success">
      Notifications were granted by the user
    </p>
    <p *ngIf="!isEnabled" class="error">
      Notifications are not available in the browser or enabled in
the application
    </p>
  `,
  styles: [`
    .success {
      color: green;
    }
    .error {
```

Resulting in this output which we will use later. However: **do not use these keys and generate your own!**

```
Public Key:
BJdhtb8aRkQIzmi217hck-
EUWO7jIQZR2dlI856wxCAFUgCIqnLY0n254gjSOAMNc9TydEf8aLoSjMMW_1QYBME
Private Key:
zJ6QK5pB6b0kpiYrtTpqu5zZ0GetmB84ob0Q1n6RZp0
```

```
      color: red;
    }
  `]
})
export class NotificationComponent {
  isEnabled = this.swPush.isEnabled;
  isGranted = Notification.permission === 'granted';

constructor(private swPush: SwPush,
              private webNotificationService:
WebNotificationService) {}

submitNotification(): void {
    this.webNotificationService.subscribeToNotification();
  }
}
```

As you probably noticed, we also have two <p> elements. The first one informs the user that he has granted our website permission to send notifications. This check is done by looking at the browser native object `Notification` and checks what the given permission is. The second <p> element tells the user that their browser either (1) does not support service workers, or (2) the service worker is not enabled in the application.

When the user has interacted with our button, we will send the subscription to the server. This logic has been delegated to our `WebNotificationService`, so let's have a peek at that. This is also where our previously discussed VAPID public key comes in!

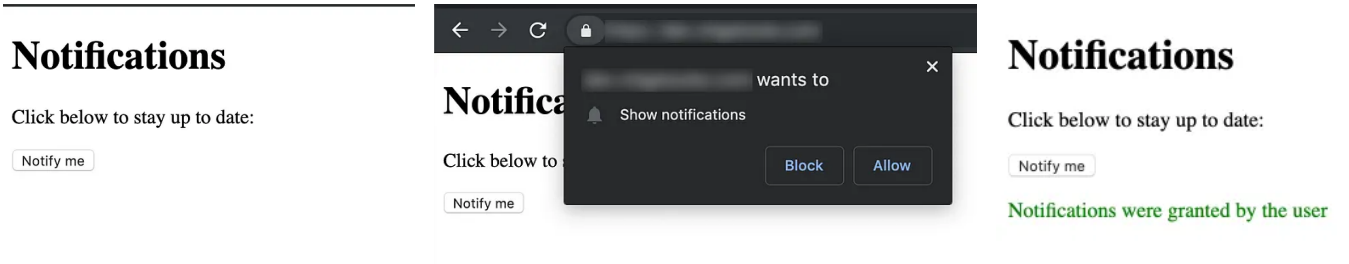
```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { SwPush } from '@angular/service-worker';

@Injectable({
  providedIn: 'root',
})
export class WebNotificationService {
  readonly VAPID_PUBLIC_KEY = '<VAPID-PUBLIC-KEY-HERE>';
  private baseUrl = 'http://localhost:5000/notifications';

  constructor(private http: HttpClient,
              private swPush: SwPush) {}

  subscribeToNotification() {
    this.swPush.requestSubscription({
        serverPublicKey: this.VAPID_PUBLIC_KEY
    })
```

```
    .then(sub => this.sendToServer(sub))
    .catch(err => console.error('Could not subscribe to
notifications', err));
  }

  sendToServer(params: any) {
    this.http.post(this.baseUrl, { notification : params
}).subscribe();
  }
}
```

So here we invoke the `requestSubscription` function from SwPush, passing our VAPID public key. If this succeeds we will receive a subscription object which will be sent to our backend. It contains all we need to actually send notifications to this user. A subscription object looks like this:

```
{
 "endpoint": "https://fcm.googleapis.com/fcm/send
/dUbChzYXFSQ:APA91bEcM0UXzo4GPmKtP6jTXtKeHFrJ-
_qhMgNxE_YXfD4jbefA567Wi5Sb183G3pYWQSTzvV40z5k3mpsdheLKBWhBz184cLM
e6uzQ51Qoy3PwBShQSRQonrni8bylHNZrqACEUvUY",
 "expirationTime": null,
 "keys": {
  "p256dh":
"BDEglRj5umJZ8RjSoYc_pgFAYLwMkQye6rhvATWfFyCVjerVmUjMvuHNu2FakpKKo
_fQpk4o9BATOTKlbxb2rfg",
  "auth": "itBJQ5Vep2mU5-whxt43og"
 }
}
```

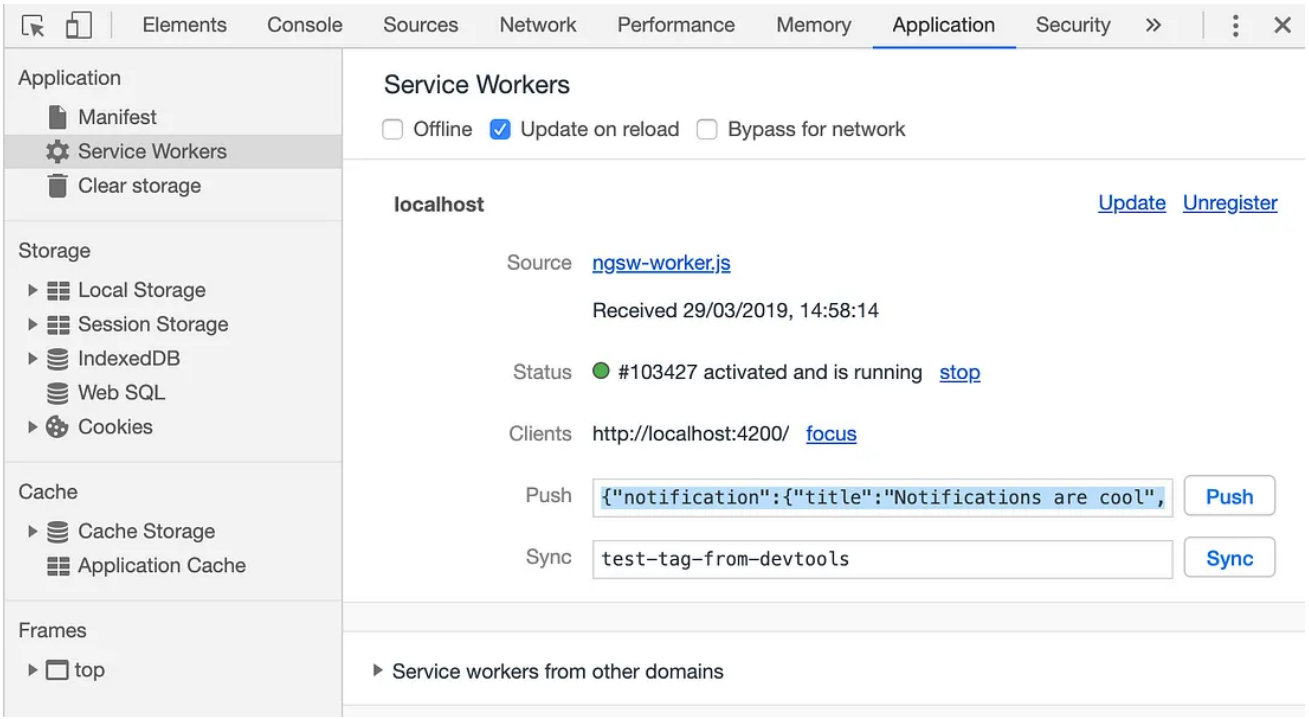
First we have the `endpoint` which our backend should call when sending a push message, followed by the `p256dh` and `auth` keys needed to prove that the user has given permission to receive these messages and we are who we say we are!



User clicks the button, will get a popup asking permission from the browser, and we see the message that permissions were granted

Now that this has been taken care of, and the user has given us his blessing to

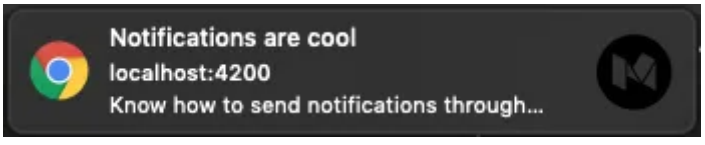
receive our notifications, we're going to send a push message. But as I mentioned earlier in the article, we will mainly focus on the frontend part of things, so how will we test this without a backend? Chrome has already found a solution. In our Development Tools "Application" tab we can inspect the service worker and test from there.



Chrome DevTools "Application" tab

In the "Push" field, we can submit a payload and click the Push button. I've prepared this one:

```
{"notification":{"title":"Notifications are cool","body":"Know how to send notifications through Angular with this article!","icon":"https://www.shareicon.net/data/256x256/2015/10/02/110808_blog_512x512.png","vibrate":[100,50,100],"data":{"url":"https://medium.com/@arjenbrandenburgh/angulars-pwa-swpush-and-swupdate-15a7e5c154ac"}}}
```



Our notification

Hooray! We are receiving our notification. Unfortunately, we are unable to

interact with it: clicking it does nothing. Time for the next step, and hooking into the SwPush `notificationClicks`. We're going to add this snippet to our `app.component.ts`:

```
import { SwPush } from '@angular/service-worker';
(...)
constructor(private swPush: SwPush) {
  this.swPush.notificationClicks.subscribe( event => {
    console.log('Received notification: ', event);
    const url = event.notification.data.url;
    window.open(url, '_blank');
  });
}
```

This time when we click the notification, a page will be opened to our url defined in the push payload.

Great that everything seems to work in our little browser vacuum, but we need a real world solution. Let's connect this to a simple backend! We're going to create a `server.js` file and write our NodeJS server. In my previous article about Progressive Web Apps, I explain why we need to build our frontend and then serve it with `http-server`. We're going to do this now as well, and start our freshly created server (code is slightly more below or on Github).

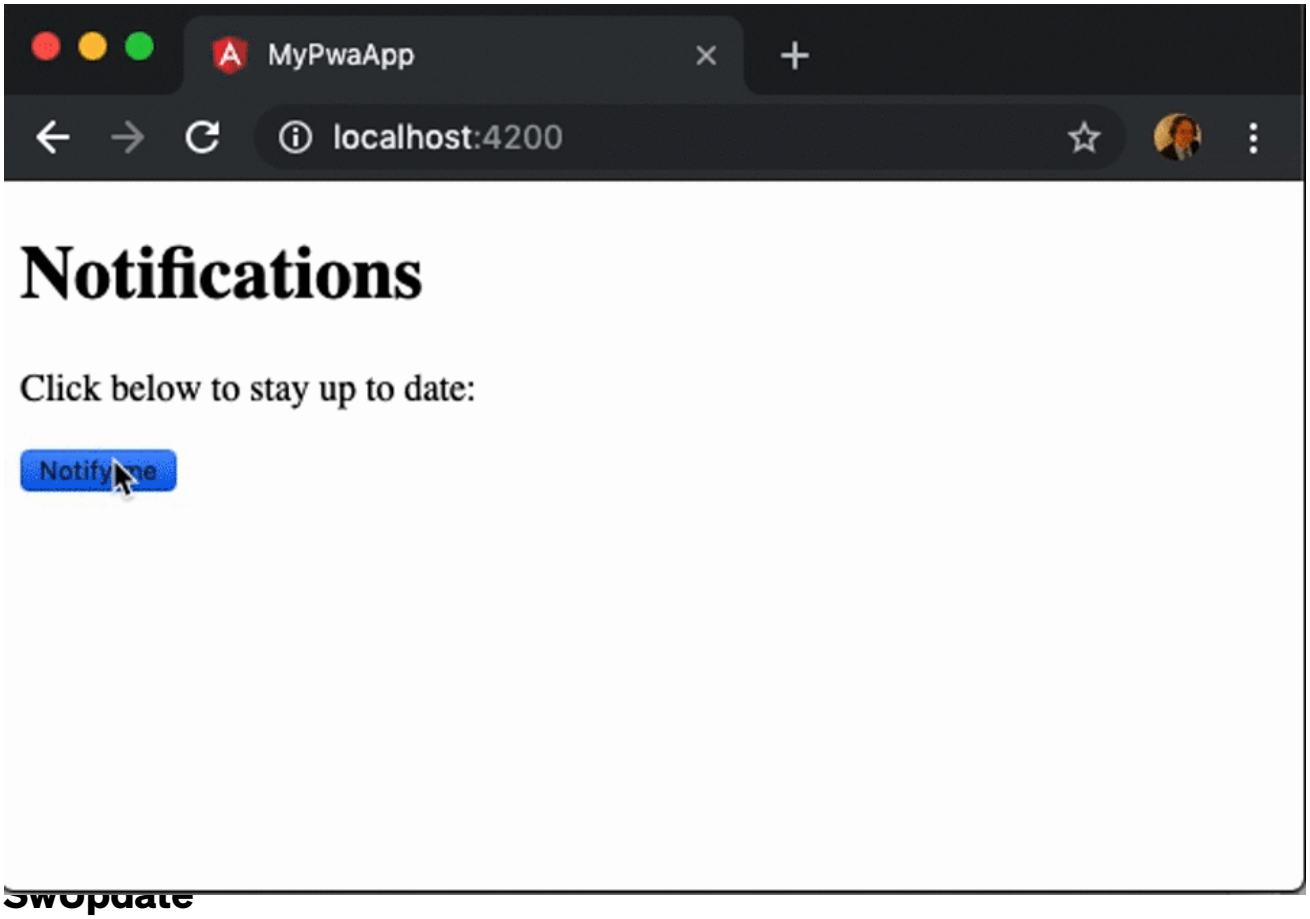
```
npm i -g http-server
ng build --prod
http-server -p4200 -c-1 dist/<name-of-app>

node server/server.js
```

Every time this server receives a new subscription, it will immediately send out a push notification.

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

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

```
const app = express();

app.use(cors());

app.use(bodyParser.json());

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

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

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

console.log(`Subscription received`);

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

const payload = JSON.stringify({
    notification: {
      title: 'Notifications are cool',
      body: 'Know how to send notifications through Angular with
this article!',
      icon: 'https://www.shareicon.net/data/256x256/2015/10
/02/110808_blog_512x512.png',
      vibrate: [100, 50, 100],
      data: {
        url: 'https://medium.com/@arjenbrandenburgh/angulars-pwa-
swpush-and-swupdate-15a7e5c154ac'
      }
    }
  });

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}`);
});
```

In my article about PWAs I've made a small repo on Github showing a basic PWA. I've created a separate branch and pull request to make it easier for everyone to see the changed involved.



Our finished push notification as found on https://github.com/arjenbrandenburgh/medium-pwa-example/blob/feature/swpush

SwUpdate

The SwUpdate can notify us if there's a new version of the service worker available, upon you can take action. We could for example write a component like this:

```
import { Component } from '@angular/core';
import { SwUpdate } from '@angular/service-worker';

@Component({
  selector: 'app-update-component',
  template: `
    <h1>Check for update</h1>
    <p>Click below to check if there's an update available:</p>
    <button (click)="checkForUpdate()">Check for update</button>
  `
})
export class UpdateComponent {
  constructor(private swUpdate: SwUpdate) {
    this.swUpdate.available.subscribe(event => {
      console.log('New update available');
      this.updateToLatest();
    });
  }

  checkForUpdate() {
    if (this.swUpdate.isEnabled) {
```

```
      this.swUpdate.checkForUpdate().then(() => {
          console.log('Checking for updates...');
      }).catch((err) => {
          console.error('Error when checking for update', err);
      });
    }
  }

  updateToLatest(): void {
    console.log('Updating to latest version.');
    this.swUpdate.activateUpdate().then(() =>
document.location.reload());
  }
}
```

What's going on here? We have a button that, when clicked, will tell the service worker to check for a new version in the `checkForUpdate` function. In the constructor we see that we're subscribing to the `swUpdate.available` observable. This observable emits when there's a new update available. When it does, it will invoke `updateTolatest` which will reload the document.

This is not really an elegant solution though. When a user is on your website, it will reload the page by itself and any input will be lost. Also, users shouldn't have to check for updates themselves. This pull request will show you the changes we've made above.

But let's find a better way to let users always have the best experience. On the SwUpdate Angular page we can find a service that will check for an update every 6 hours. Let's use this service!

```
import { ApplicationRef, Injectable } from '@angular/core';
import { SwUpdate } from '@angular/service-worker';
import { concat, interval } from 'rxjs';
import { first } from 'rxjs/operators';

@Injectable()
export class CheckForUpdateService {

  constructor(appRef: ApplicationRef, updates: SwUpdate) {
    // Allow the app to stabilize first, before starting polling
for updates with `interval()`.
    const appIsStable$ = appRef.isStable.pipe(first(isStable =>
isStable === true));
    const everySixHours$ = interval(6 * 60 * 60 * 1000);
    const everySixHoursOnceAppIsStable$ = concat(appIsStable$,
everySixHours$);
```

```
everySixHoursOnceAppIsStable$.subscribe(() =>
updates.checkForUpdate());
  }
}
```

We're going to add a toaster that will notify the user that there's an update, and let them reload themselves whenever they're ready. Our `app.component.ts` now looks like this

```
import { Component } from '@angular/core';
import { SwUpdate } from '@angular/service-worker';
import { CheckForUpdateService } from './services/check-for-
update.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'my-pwa-app';
  updateAvailable = false;

constructor(private updates: SwUpdate,
  private checkForUpdateService: CheckForUpdateService) {
    this.updates.available.subscribe((event) => {
      this.updateAvailable = true;
    });
  }
}
```

The `checkForUpdateService` is loaded, and is checking every 6 hours for an update. In the AppComponent constructor we are listening for an update. If it's available, we're toggling a boolean that will show the toaster, and sure enough:

Our toaster notifying the user that there's an update.

If you want to see the full example, you can check out this Github pull request.

Summary

In this article we have gone over the functionality of SwPush and SwUpdate. These two packages can be leveraged to make our PWA more interactive, and to make sure our users are aware of updates. Our users will always have the most recent and optimal experience on our web applications.

We have come a long way when it comes to websites and web applications. Progressive Web Apps are the next step on this beautiful journey. Not only are we able to quickly, and conveniently, show our web application on our users' devices, but we're gaining more and more tools to make our applications more interactive, intuitive and useful. Being able to present our users with information whenever they need it, is nothing short of a little miracle!

Looking for a job in Amsterdam?

I work for **Sytac** as a Senior Front-end developer and we are looking for medior/senior developers that specialise in Angular, React, Java or Scala. Sytac is a very ambitious consultancy company in the Netherlands that works for a lot of renowned companies in banking, airline, government and retail sectors. You can think of companies like ING, KLM, Deloitte, Ahold Delhaize, ABN AMRO, Flora holland and many more.

From a personal opinion Sytac really sets itself apart with their client portfolio, but also with how they take care of their employees. They do really care about the wellbeing of their employees. Apart from a good salary (50K-75k), you will notice this in regular meetings with the consultant managers but also by the amount of events they organise and all the other perks they offer to keep all employees happy.

If you think you have what it takes to work with the best, send me an email on arjen.brandenburgh@sytac.io and I'll be happy to tell you more.

JavaScript Angular Programming Front End Development

Web Development

Follow

Written by Arjen Brandenburgh

181 Followers

Fullstack Team Lead from the Netherlands @ Techspire

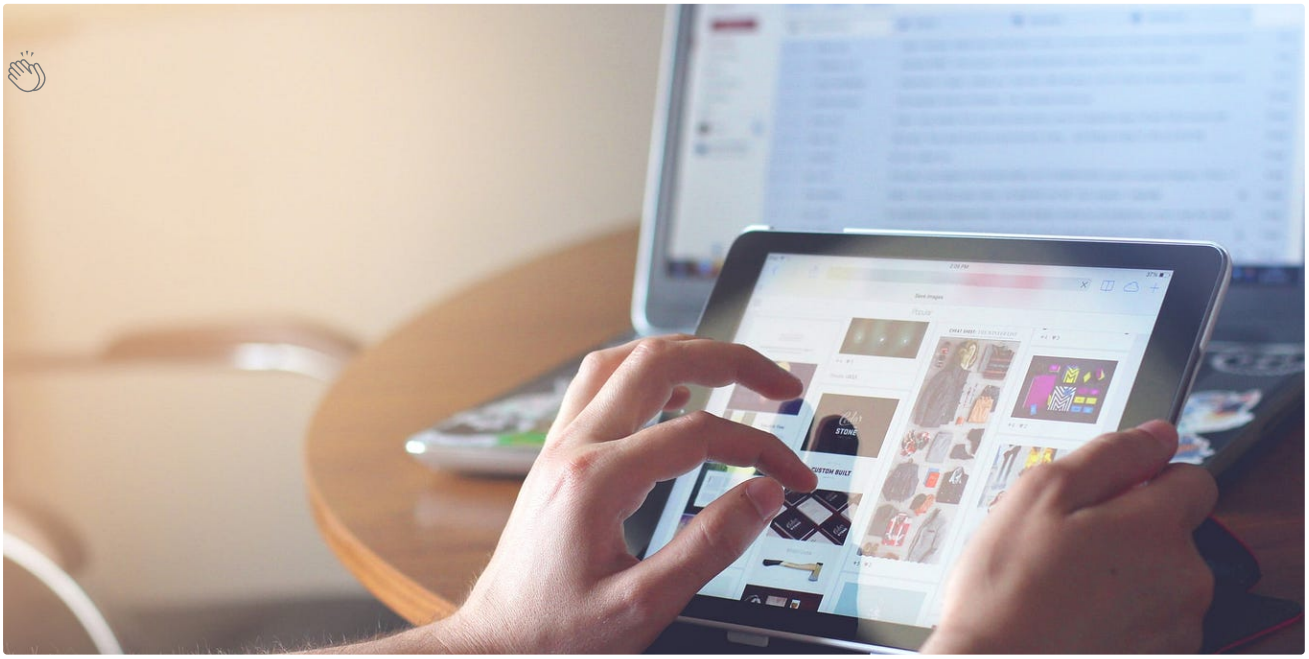
More from Arjen Brandenburgh

Arjen Brandenburgh in Techspiration

Why you shouldn't use Lodash

Javascript has made enormous strides the past few years and functions like find, findIndex, map, filter, and reduce are now standard...

· 5 min read · Jul 21, 2020



Arjen Brandenburgh in Techspiration

Lazy load & split your Angular Material dialogs!

Recently for a client we needed dialog that has some complex user experience flows. It basically boiled down to a wizard with multiple...

· 4 min read · Jul 13, 2020

71

Arjen Brandenburgh

Understanding Angular’s ViewEncapsulation

In your Angular components you’re able to specify a component’s ViewEncapsulation. It defines template and style encapsulation options...

· 6 min read · Apr 1, 2019

243

Arjen Brandenburgh

Your Angular app as Progressive Web App

A Progressive Web App (or PWA) is a web application that provides a set of capabilities to give web sites an app like experience...

Recommended from Medium

8 min read · Mar 22, 2019

Developer Partners

How to Show a Modal Dialog in Angular

Using modal dialogs in web applications is fairly common. Whether you want to edit a record in a table without navigating to a different...

7 min read · 6 days ago

4 1

Ravindra Devrani

Angular pagination with infinite scroll (using ngx-infite-scroll)

Sometimes we get a condition where you don't want to display all the data at once. Initially we display some data, and when we scroll down...

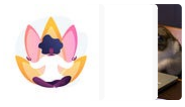
3 min read · Feb 20

Lists

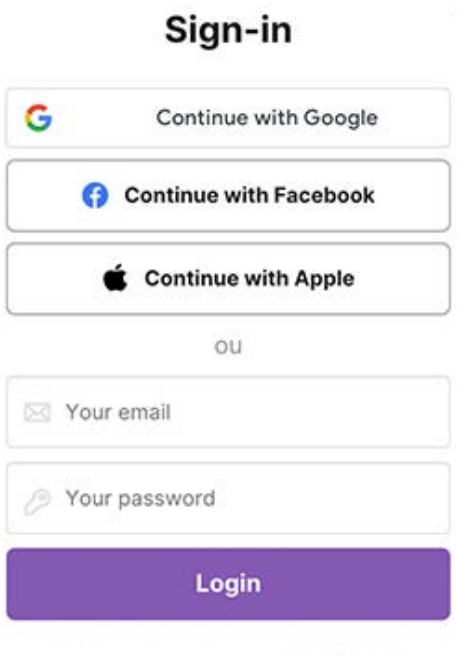
General Coding Knowledge
20 stories · 224 saves

It's never too late or early to start something
15 stories · 81 saves

Coding & Development
11 stories · 114 saves



Stories to Help You Grow as a Software Developer
19 stories · 286 saves



Leonardo Salles

A guide to custom Google Sign-in button

Recently Google has deprecated the Sign-in for Web. Now we have to add login using Google Identity Series(GSI).

3 min read · Mar 24

126 8

Nourhan Elsherif

Part 1—Exploring Firebase for Frontend Projects

If you're a frontend developer aiming to create a full-stack project without getting your hands dirty with servers and backend, Firebase...

2 min read · Jul 23

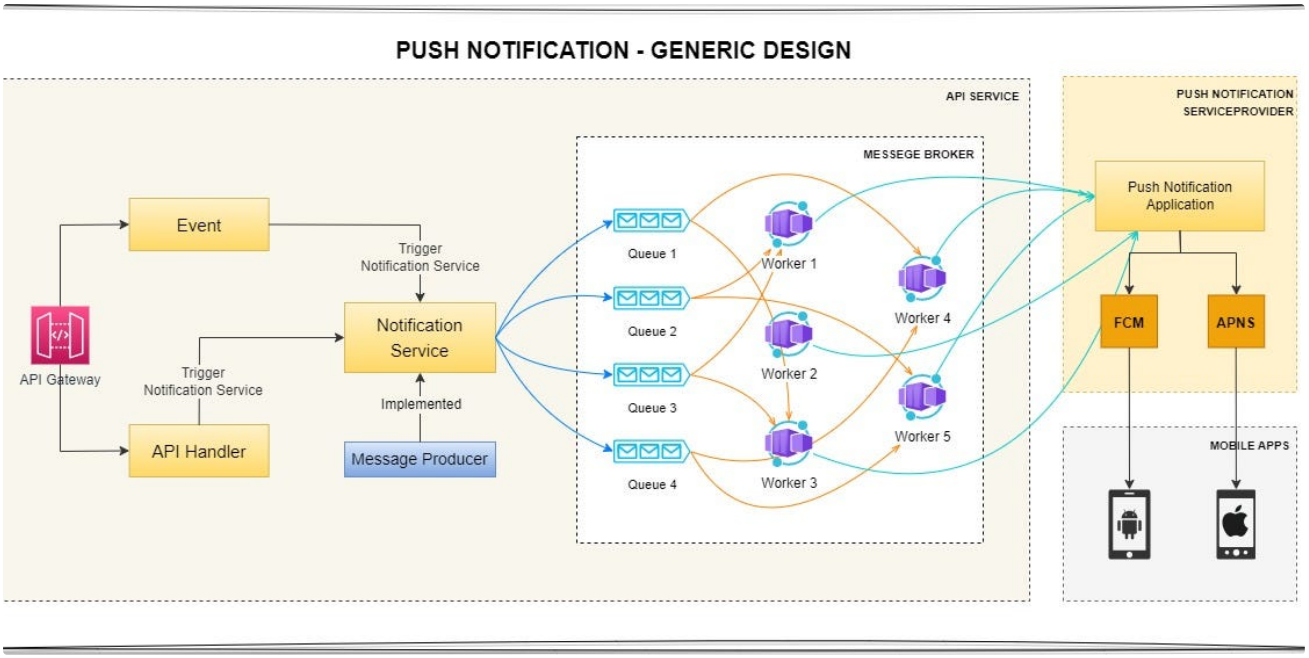
Emre Ertuğ

How To Create Dynamic Forms With Angular

Most of the time we create forms from scratch. Each time we spend unnecessary amount of time creating and styling these forms. Only if...

3 min read · Aug 9

36 1



Er. Udhaya

Push Notifications — Generic design and Java implementation

Mobile notifications, also known as push notifications, have become an essential component of the smartphone user experience. They provide...

10 min read · Feb 22

21

See more recommendations