4 OCTOBER 2018 / ANGULAR

Push Notifications + Angular + .NET Core

I'm sure 100% of you have already heard about what a push notification is, that's what apps and several web sites do to get your attention even if might seem a little bit of magic for some. In this post, I'm going to cover how to implement push notifications in an Angular application and also how to handle these notifications differently when you have the app open.

> *As always, the source code covered in this post is available here*

Push notifications in a web site is now possible due to service workers, which is something that's going to run in the background even if you don't have the website open. If you haven't heard about what service workers are, check out this post.

To get started, I'm going to use the .NET Core + Angular SPA template (if you want to know more about this template, check this post). Here are the very simple commands:

```
mkdir push-notification-angular-dotnet-core
cd push-notification-angular-dotnet-core
dotnet new angular
```

Here's our plan, the stuff we need to put together:

1. A service worker in place to handle push events when the site is not open

3. Create Public and Private keys
4. A way to subscribe and unsubscribe to push events on demand. Basically, ask permission from the user to send push events.
5. Handling push events
6. Handling subscriptions in the server
7. Send push events from the server

#1 Create our service worker

For now, we're just going to create a `sw.js` file in the assets folder. This is going to be our service worker.

#2 Register our service worker

Since we're using Angular, I want to do this in the neatest way possible, so I'm going to create a service for this using the Angular Cli:

```
cd ClientApp\src\app
ng g s core/notificationMiddleware
```

the state of push notification of the browser instance.

```
public pushNotificationStatus = {
    isSubscribed: false,
    isSupported: false
  };
```

An `init` method which I'm going to call from the `app.component` as soon as the app starts which is going to check if the current browser supports service workers and push notifications, if it does then we're going to automatically register the service worker (`sw.js`) we added to our assets folder. Also a `checkSubscription` method to update the `isSubscribed` flag.

```
init() {
  if ('serviceWorker' in navigator && 'PushManager' in window) {
    navigator.serviceWorker.register('/assets/sw.js')
      .then(swReg => {
        console.log('Service Worker is registered', swReg);

        this.swRegistration = swReg;
        this.checkSubscription();
      })
      .catch(error => {
        console.error('Service Worker Error', error);
      });
    this.pushNotificationStatus.isSupported = true;
  } else {
    this.pushNotificationStatus.isSupported = false;
```

```
checkSubscription() {
  this.swRegistration.pushManager.getSubscription()
    .then(subscription => {
      console.log(subscription);
      console.log(JSON.stringify(subscription));
      this.pushNotificationStatus.isSubscribed = !(subscription === null);
    });
}
```

Now in the `app.component.ts` , we're going to inject the `NotificationMiddlewareService` and call the `init` method:

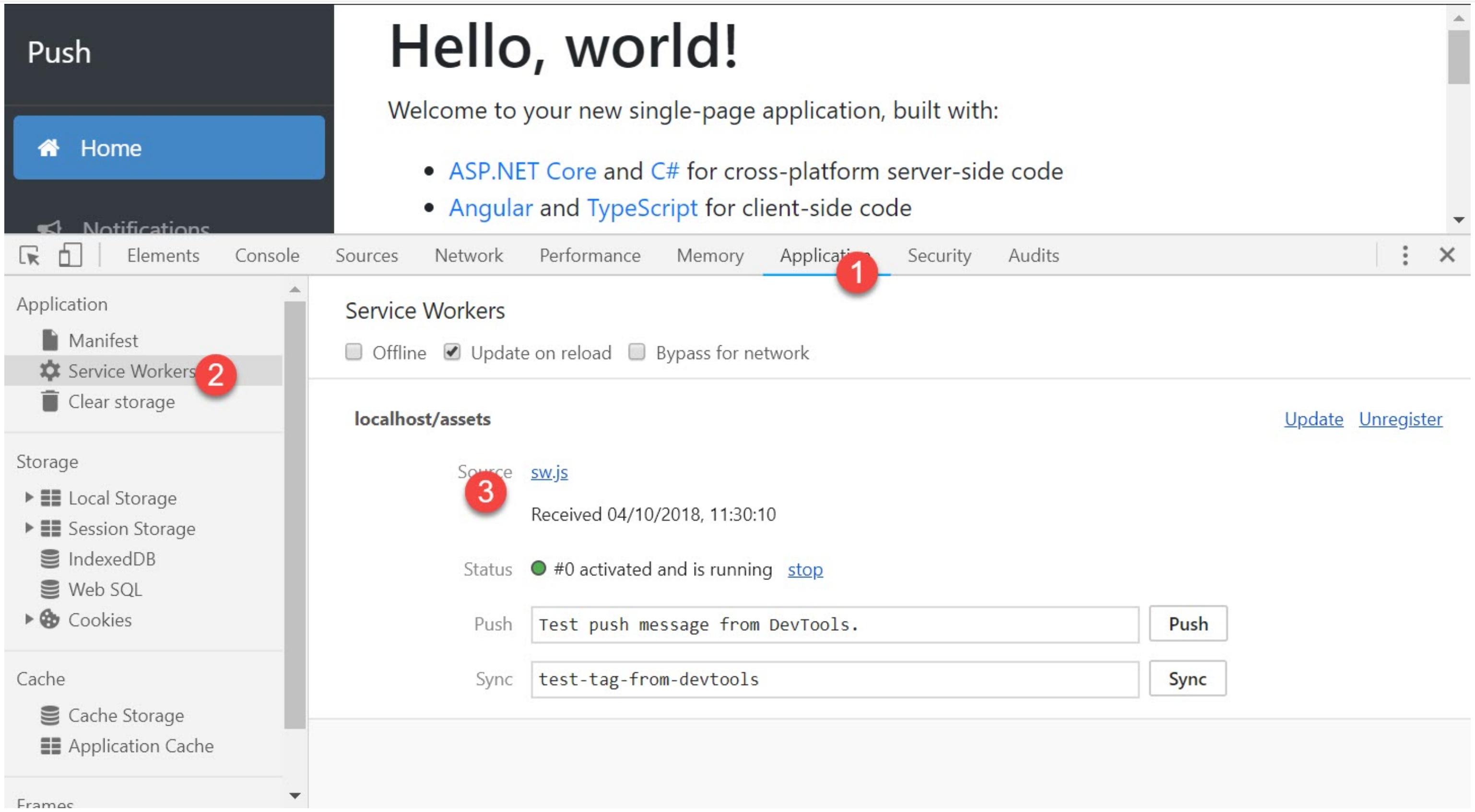
```
export class AppComponent implements OnInit {
  title = 'app';

  constructor(private notificationMiddleware: NotificationMiddlewareService) {
  }

  ngOnInit(){
    this.notificationMiddleware.init();
  }
}
```

If we run `ng serve` in the `ClientApp` folder and go to the browser now, we should see the service worker installed by going to the *Dev Tool bar > Application > Service Workers*.

> *While in development stage, keep the flag* `Update on reload` *checked, so every time*


Push
Home
Notifications
Hello, world!
Welcome to your new single-page application, built with:
• ASP.NET Core and C# for cross-platform server-side code
• Angular and TypeScript for client-side code
Elements Console Sources Network Performance Memory Application Security Audits
1
Application
Manifest
Service Workers
2
Clear storage
Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies
Cache
Cache Storage
Application Cache
Frames
Service Workers
Offline Update on reload Bypass for network
localhost/assets
Update Unregister
Source sw.js
3
Received 04/10/2018, 11:30:10
Status #0 activated and is running stop
Push Test push message from DevTools. Push
Sync test-tag-from-devtools Sync

#3 Create Public and Private keys

Before we get to the next step which, we need a public and private keys so all the communication between the sender and the receiver is encrypted. There are several ways to do it, but the simpler way I found was via the website https://web-push-codelab.glitch.me. You can just go there and copy both the **Public key** and **Private key**. We're going to add our **Public key** to the `environment.ts` file:

```
  production: false,
  applicationServerPublicKey: `[PUBLIC KEY]`
};
```

In the server side, we're going to add them to the `appsettings.json` .

```
{
  "VapidDetails": {
    "Subject": "[ORIGIN URL, EX:https://localhost:4201]",
    "PublicKey": "[PUBLIC KEY]",
    "PrivateKey": "[PRIVATE KEY]"
  }
}
```

#4 Subscribe and Unsubscribe

Instead of asking the user to trust the site as soon as they land on the site, it's a best practice that we ask for permission on demand so the user knows what he's agreeing to. So we're going to create `subscribe` and `unsubscribe` methods in the just created service and call this from the UI.

The `subcribe` method encrypts the public key and uses the `pushManager` to subscribe the user:

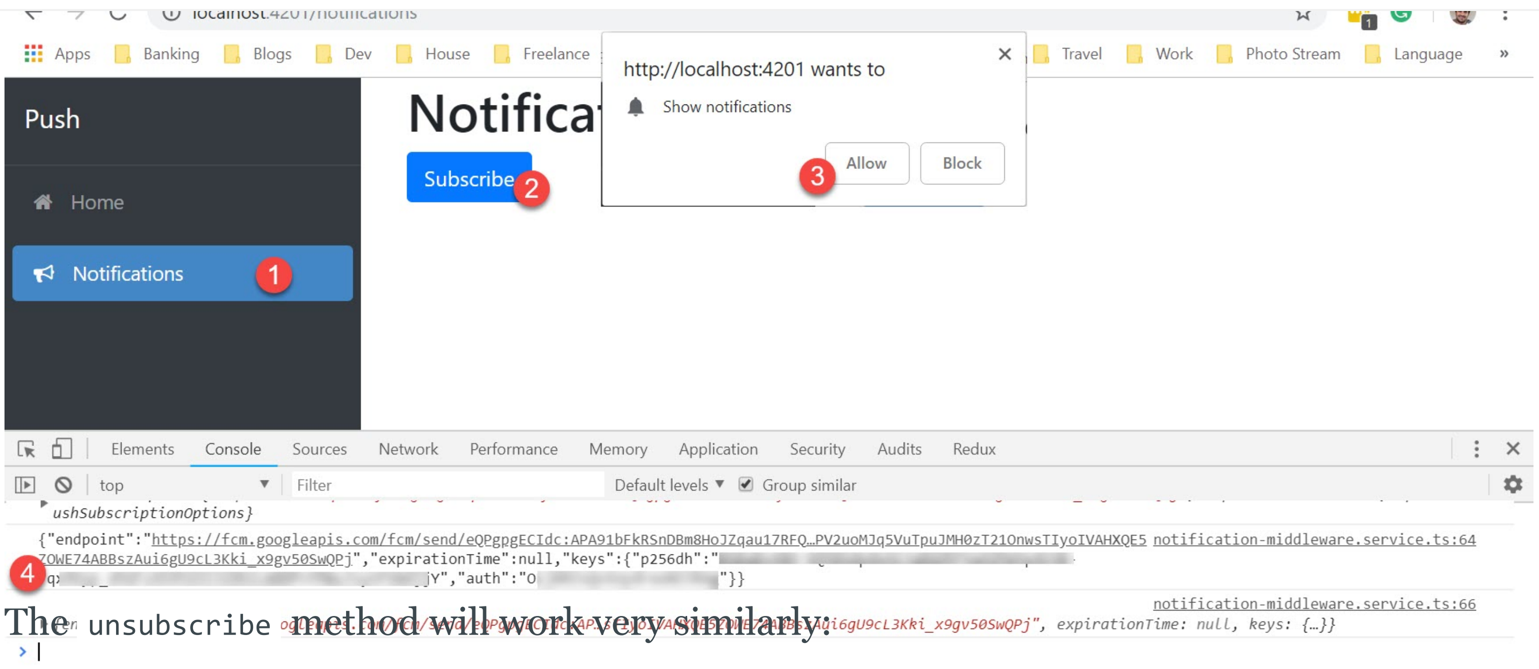
```
subscribe() {
    const applicationServerKey = this.urlB64ToUint8Array(environment.applicationServerPublicKey);
    this.swRegistration.pushManager.subscribe({
      userVisibleOnly: true,
      applicationServerKey: applicationServerKey
    })
      .then(subscription => {
        console.log(JSON.parse(JSON.stringify(subscription)));
        this.pushNotificationStatus.isSubscribed = true;
      })
      .catch(err => {
        console.log('Failed to subscribe the user: ', err);
      })
      .then(() => {
        this.pushNotificationStatus.isInProgress = false;
      });
  }
```

I've created a notifications component so I can add a button to call the subscribe method and as soon as I call it, the user will be prompted to either allow or block as per screenshot below. Once they allow, you should have the subscription which will have everything you need to send push notifications.

> *You might want to copy the subscription from the logs to test the push notifications later on*

Thiago Passos

Share this



```
ushSubscriptionOptions}
{"endpoint":"https://fcm.googleapis.com/fcm/send/eQPgpgECIdc:APA91bFkRSnDBm8HoJZqau17RFQ…PV2uoMJq5VuTpuJMH0zT21OnwsTIyoIVAHXQE5 notification-middleware.service.ts:64
ZOWE74ABBszAui6gU9cL3Kki_x9gv50SwQPj","expirationTime":null,"keys":{"p256dh":"[redacted]
q[redacted]jY","auth":"O[redacted]"}}
                                                                             notification-middleware.service.ts:66
oogleapis.com/fcm/send/eQPgpgECIdc:AP…5TIyoIVAHXQE5ZOWE74ABBszAui6gU9cL3Kki_x9gv50SwQPj", expirationTime: null, keys: {…}}
```

The `unsubscribe` method will work very similarly:

```
unsubscribe() {
  this.pushNotificationStatus.isInProgress = true;
  this.swRegistration.pushManager.getSubscription()
    .then(function (subscription) {
      if (subscription) {
        return subscription.unsubscribe();
      }
    })
    .catch(function (error) {
      console.log('Error unsubscribing', error);
    })
    .then(() => {
      this.pushNotificationStatus.isSubscribed = false;
      this.pushNotificationStatus.isInProgress = false;
```

In order to make my life easier, I've created a `toggleSubscription` method, since we already have a flag that recognizes if the user is subscribed or not:

```
toggleSubscription() {
  if (this.pushNotificationStatus.isSubscribed) {
    this.unsubscribe();
  } else {
    this.subscribe();
  }
}
```

#5 Handling push events

Now that we have a subscription, it's time to start handling push events. That's where the service worker is going to shine. Let's go now to the `sw.js` file.

> *WARNING: The service worker is javascript file, not typescript.*

We're going to add an event listener for the push event.

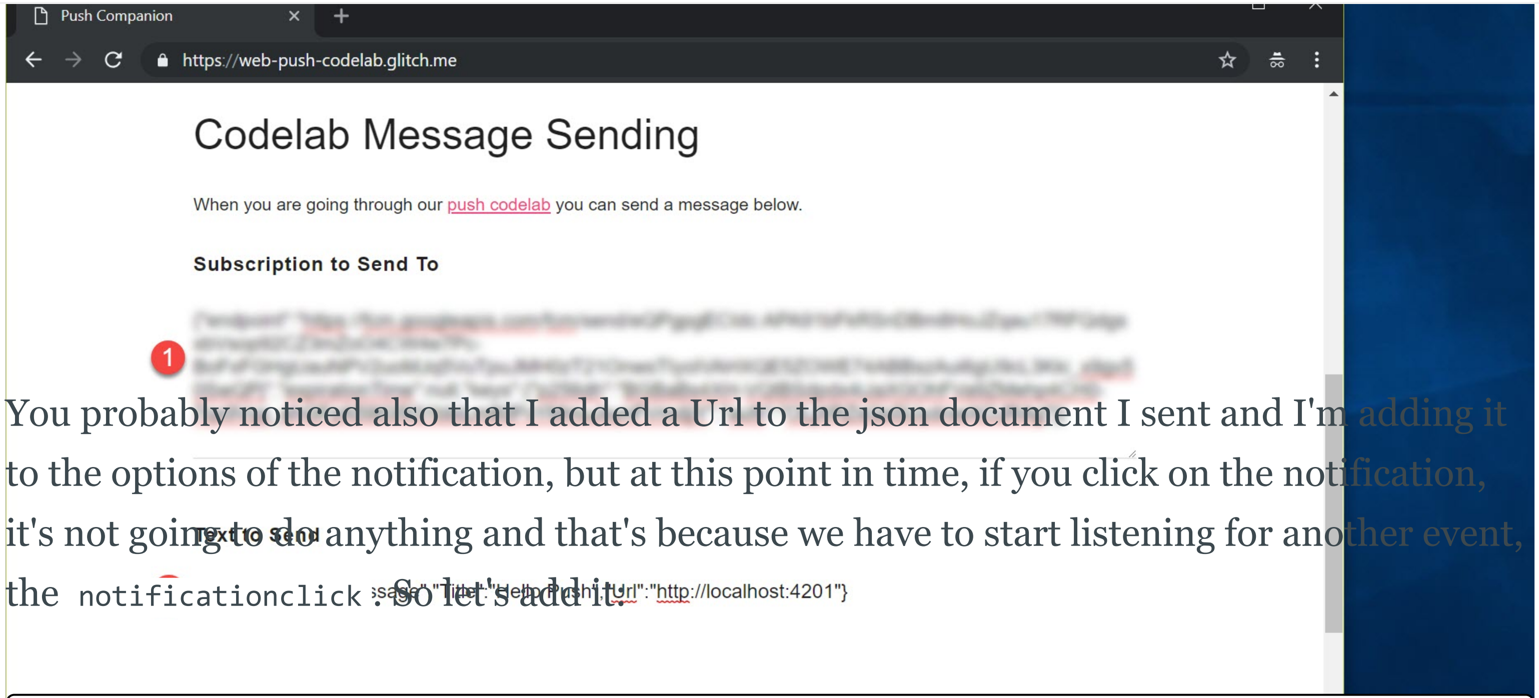
```
'use strict';

self.addEventListener('push', function (event) {
    const title = data.Title;
            const options = {
                body: data.Message,
                icon: 'assets/push.png',
                badge: 'assets/push.png',
                data: data.Url
            };

            const promiseChain = self.registration.showNotification(title, options);

    event.waitUntil(promiseChain);
});
```

To test it out, let's go back to https://web-push-codelab.glitch.me and paste the subscription we got from the previous step. Notice that I'm sending a json document in the



Codelab Message Sending

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

Subscription to Send To

[redacted]

Text to Send

{"Message":"Title":"Hello Push","Url":"http://localhost:4201"}

You probably noticed also that I added a Url to the json document I sent and I'm adding it to the options of the notification, but at this point in time, if you click on the notification, it's not going to do anything and that's because we have to start listening for another event, the `notificationclick`. So let's add it:

```
self.addEventListener('notificationclick', function (event) {
    const urlToOpen = new URL(event.notification.data, self.location.origin).href;

    event.notification.close();

    event.waitUntil(clients.openWindow(urlToOpen));
});
```

Great, now if you click on the notification, it will open the Url we sent.

Okay, but what if I already have a tab open of that site, or if I am on the site itself right now? It's going to be a pain in the bum to start getting all those push notifications and opening always a new instance. There's a solution for that, this post talks about all different but I'm not going to cover here as this post is already getting too long.

#6 Handle subscriptions in the server

For this section, we're going to need both the public and private keys we created in the step #3, also we're going to need to modify the `subscribe` and `unsubscribe` methods so it calls the API endpoints to manage the subscriptions.

> *For testing purposes, I'm going to save all the subscription into memory, but if you're going to implement this somewhere, be sure to persist the subscriptions somewhere else.*

To send the push notifications, I'm going to use a nuget package called WebPush. I've also configured in the startup to inject the `VapidDetails` which are the details we need to send the push notification: Subject (Origin URL), Public and Private keys.

I've created a `Notification` controller to handle 3 things: **Subscribe**, **Unsubscribe** and **Broadcast**:

```
[ApiController]
[Route("api/[controller]")]
public class NotificationController : Controller
{
    public static List<PushSubscription> Subscriptions { get; set; } = new List<PushSubscription>
    [ProducesResponseType((int)HttpStatusCode.OK)]
    public void Subscribe([FromBody] PushSubscription sub)
    {
        Subscriptions.Add(sub);
    }

    [HttpPost("unsubscribe")]
    [ProducesResponseType((int)HttpStatusCode.OK)]
    public void Unsubscribe([FromBody] PushSubscription sub)
    {
        var item = Subscriptions.FirstOrDefault(s => s.Endpoint == sub.Endpoint);
        if (item != null)
        {
            Subscriptions.Remove(item);
        }
    }

    [HttpPost("broadcast")]
    [ProducesResponseType((int)HttpStatusCode.OK)]
    public void Broadcast([FromBody] NotificationModel message, [FromServices] VapidDetails vapid
    {
        var client = new WebPushClient();
        var serializedMessage = JsonConvert.SerializeObject(message);
        foreach (var pushSubscription in Subscriptions)
        {
            client.SendNotification(pushSubscription, serializedMessage, vapidDetails);
        }

    }
}
```

`unsubscribe` methods to call the server side endpoints.

> *In order to talk to the server, I've implemented OpenApi with code generation, to know more about it, check this post. My NSWAG is also in the source control if you need to re-generate the typescript file.*

So now my `subscribe` method is calling

```
this.notificationService.subscribe(<PushSubscription>{
  auth: newSub.keys.auth,
  p256Dh: newSub.keys.p256dh,
  endPoint: newSub.endpoint
}).subscribe(s => {
  this.pushNotificationStatus.isSubscribed = true;
})
```

And the `unsubscribe` is calling

```
this.notificationService.unsubscribe(<PushSubscription>{
  auth: sub.keys.auth,
  p256Dh: sub.keys.p256dh,
  endPoint: sub.endpoint
}).subscribe(() => {
  this.pushNotificationStatus.isSubscribed = false;
  this.pushNotificationStatus.isInProgress = false;
});
```

Alright, now to test it all out, I've created a broadcast component to call the broadcast endpoint in the notification controller. Let's see it in action.

1. On the right hand side I've got Chrome which I'm going to broadcast the message from, on the left hand side Microsoft Edge.
2. When I send the message, it broadcasts to everyone who's subscribed.
3. I get a notification for Edge
4. I get an in app notification for chrome since I have it focused.

Thiago Passos

Share this

Cheers.

Buy me a coffee

Thiago Passos

I'm Thiago Passos, a Solution Architect working for SSW sharing what I've been working with and learning. Love technology, dancing and I get unfriendly when I'm hungry.

Read More

What do you think?

9 Responses

Upvote Funny Love Surprised Angry Sad

2 Comments **passos.com.au**

1 Login

Recommend Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Farouq • a month ago

how can i call a method located in the middleware.service from within the sw.js file ???

i am asking this cuz i am handling the broadcast from the backend without the need of the front end broadcast button.

count.

Reply • Share ›

Ahmed Mohamed • 7 months ago

great work , I download demo but it is n't working & Raise this error
XHR finished loading: POST "http://localhost:5000/api/Notification/subscribe".
ERROR Error: An unexpected server error occurred.
at new SwaggerException (generated.ts:192)

Reply • Share ›

ALSO ON **PASSOS.COM.AU**

My Console Windows Service

6 comments • 2 years ago

Avatar **Manav Joshi** — Logs.. I am having a real hard time solving this one. Could it be because I have Windows 10 creator's update OS? https://uploads.disquscdn.c...

Azure Superpowers Tour

1 comment • 2 years ago

Avatar **Duncan Hunter** — Thanks for the awesome Azure SuperPowers day in Brisbane I really enjoyed attending it!

TryGhost v1.11 Deployed To Azure

4 comments • 2 years ago

Avatar **Thiago Sobral** — Hi RiskyB, I haven't updated yet, but I've got a pull request from a friend with it just didn't have the time yet to do it. You can check his post about it https://blog.gabrielgeorge....

Getting Started: EF Core + Cosmos Db

4 comments • a year ago

Avatar **Quinntyne Brown** — Hi Thiago, Thanks for the awesome post. I'm trying it out and getting the following error. It's for a simple "var books = _context.Books.ToList();"One or more errors occurred. (Cross partition query is required but disabled. Please set x-ms-documentdb-query-enablecrosspartition to true, specify x-ms-documentdb-partitionkey, or revise your query to avoid this exception., Windows/10.0.17134 documentdb-netcore-sdk/1.7.1)Any ideas as to what the issue is?

angular

Rich Text Editor for Angular

YouTube: Angular In 5 Minutes

Getting Started With SignalR, .NET Core 2.1 And Angular

See all 8 posts →

ANGULAR

Rich Text Editor for Angular

Okay, so I've published my very first npm package. Let's start from the beginning. We've got an internal MVC application that uses the Kendo UI from Progress, the rich text editor to be

THIAGO PASSOS

NDC

Thiago Passos Share this

In the last 2 years, I went to NDC Sydney and it's always been a great event, possibly the best conference I've been, but this time it would be different, this time I

THIAGO PASSOS

Latest Posts Ghost