****

**Ypto**

SNCB Information Technology

**Rapport de Stage**

Notifications Push, technologies et POC

**Le Guernic François, 4 sep 2019**

Contents

[I. Contexte et projet 3](#_Toc18448834)

[II. Description fonctionnelle 3](#_Toc18448835)

[1. Intérêt des notifications Push 3](#_Toc18448836)

[2. Choix d’une application Web 4](#_Toc18448837)

[3. Fonctionnement des notifications Web Push 6](#_Toc18448838)

[4. Objectifs initiaux 6](#_Toc18448839)

[III. Architecture du projet 8](#_Toc18448840)

[IV. Description technique du rendu 10](#_Toc18448841)

[1. Etapes de formation du POC 10](#_Toc18448842)

[2. Aperçu de certaines parties du code 12](#_Toc18448843)

[3. Outils externes utilisés 13](#_Toc18448844)

[4. Difficultés rencontrées 15](#_Toc18448845)

[V. Etapes suivantes 16](#_Toc18448846)

[VI. Rétropsective sur le stage 16](#_Toc18448847)

# Contexte et projet

En tant que filiale de la SNCB, Ypto fournit les services informatiques à l'opérateur ferroviaire.

J’ai réalisé mon stage dans l’équipe “Traffic Management” sous la responsabilité de Toon Scheirlinckx.

L’équipe offres des solutions et des services aux départements SNBC en charge de l’exécution opérationnel du trafic ferroviaire.

Dans plusieurs processus les employés opérationnels doivent être averti en temps réel sur différents équipements en forme digital.

* Déviation d’un train et nécessité d’informer les trains liés
* Retard d’un train
* Incidents avec des collègues
* Un liaison entre deux trains qui vient d’être annulé
* …

Le projet proposé porte sur la réalisation d’un “Proof of Concept” (POC) d’une application utilisant des notifications de type “Push”.

L’idée étant de faire une veille technologique sur les différentes technologies possibles pour ce projet, ainsi que de réaliser une application la plus aboutie possible.

# Description fonctionnelle

## Intérêt des notifications Push

Une notification Push est un message d’alerte envoyé à l’utilisateur pour :

* Envoyer une information importante
* Rediriger l’utilisateur vers une page en particulier

Les notifications push présentent l’avantage intéressant de garder un contact avec l’utilisateur même lorsque celui-ci n’est pas sur son navigateur.

C’est une technologie qui s’apparente aux SMS, eux-mêmes pouvant servir à informer l’utilisateur lorsqu’il n'est pas connecté à l’application. Toutefois les notifications Push ont notamment l’avantage d’utiliser une connexion de données mobile, et d’être plus “customisables “ que les SMS *(image, effets de retard…)*



Notifcations “Push” via le web

## Choix d’une application Web

Il y a deux types de notifications :

* Locales *(quand elles proviennent de l’app en elle-même)*
* Push *(quand elles proviennent d’un server )*

Une des premières interrogations a été la comparaison entre une application native *(applications à installer sur le mobile)* et une application web.

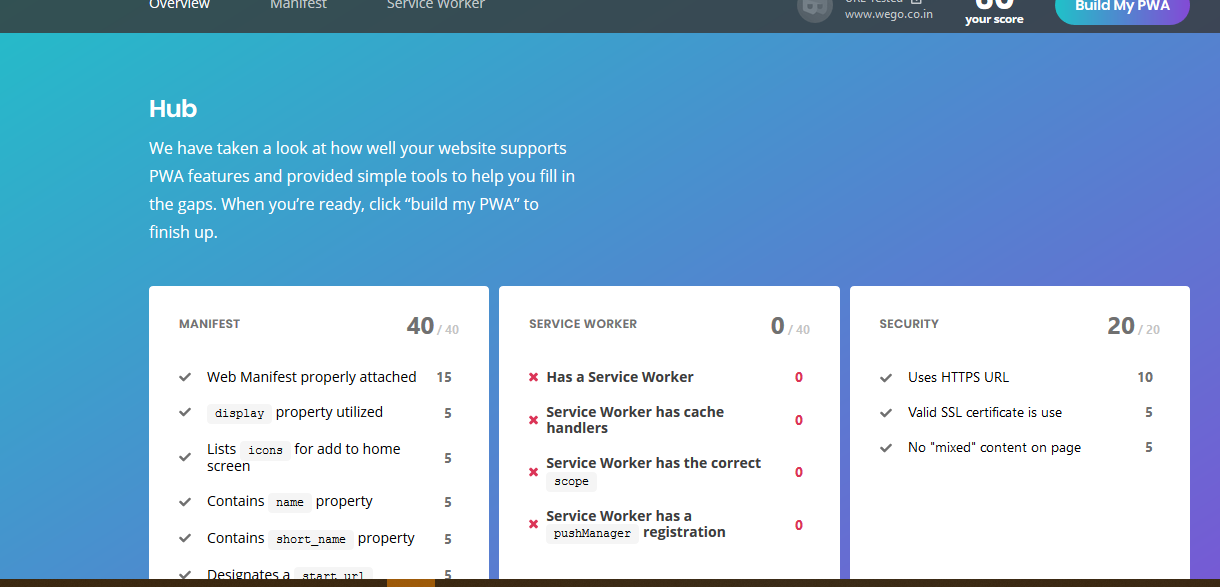
Réaliser une application web dans le cadre du projet a présenté différents avantages :

|  |  |
| --- | --- |
| Native App Notifications | Web Push Notifications |
| Envoi de notifications quand l’utilisateur n’est pas sur l’application | Envoi de notifications quand l’utilisateur n’est pas sur le site / n’a pas le navigateur ouvert |
| Uniquement pour mobile | Pour toutes les plateformes |
| Nécessite des updates | Ne nécessite pas d’updates |
| Code propre à un OS | Code multiplatforme |

Une application web qui répond à un certain nombre de critères est appelée une **Progressive Web App.**

Expérience utilisateur : les PWA peuvent même , si c’est écrit dans leur « **manifest.json** », créer une icône à installer dans l’écran d’accueil *(proposé automatiquement sur Android)*.

Il existe des outils gratuits pour vérifier qu’une application répond à la majorité des critères des PWA : par exemple PWABuilder.com :



Ces différents points ont donc amené à étudier plus en profondeur l’utilisation de notifications Push via le Web .

Toutefois, réaliser des notifications Push avec le Web présente actuellement un désavantage :

La Push API n’est pas compatible avec IOS. En d’autres termes, il n’est pas possible d’envoyer des notifications Push sur IPhone. Dans le cadre du projet de stage, ce n’est pas si dérangeant, car les employés de l’entreprise utilisent des appareils sur Android.

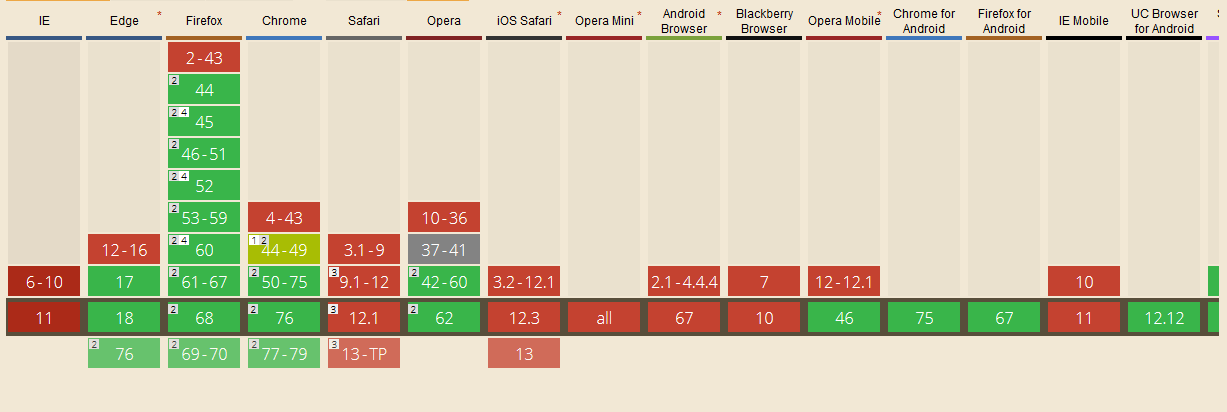
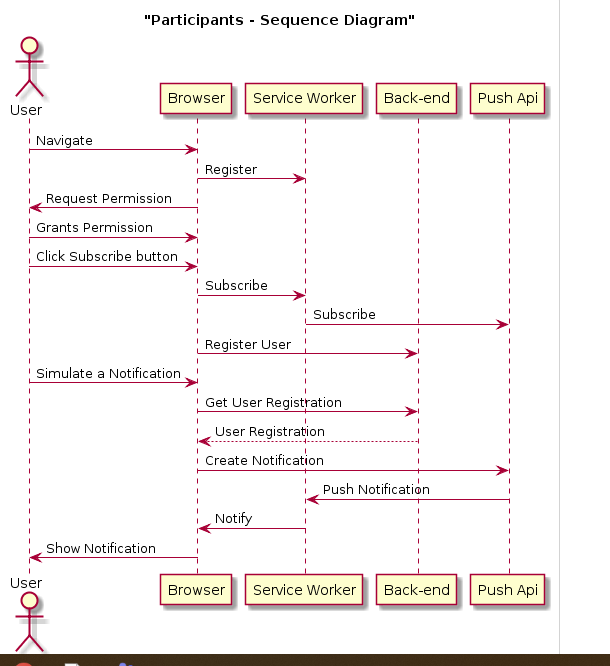


Table de compatibilité de la Push API

*CanIuse.com*

## Fonctionnement des notifications Web Push

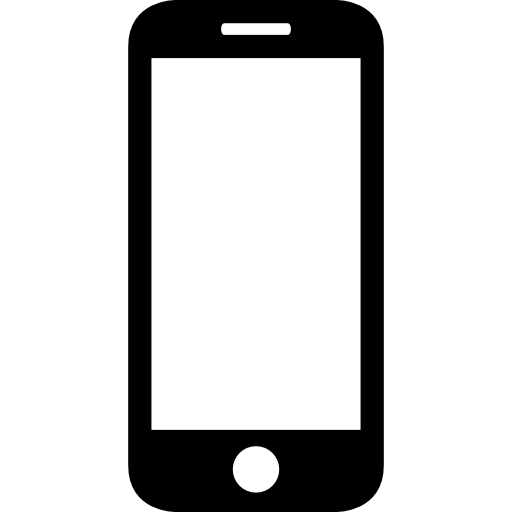


## Objectifs initiaux

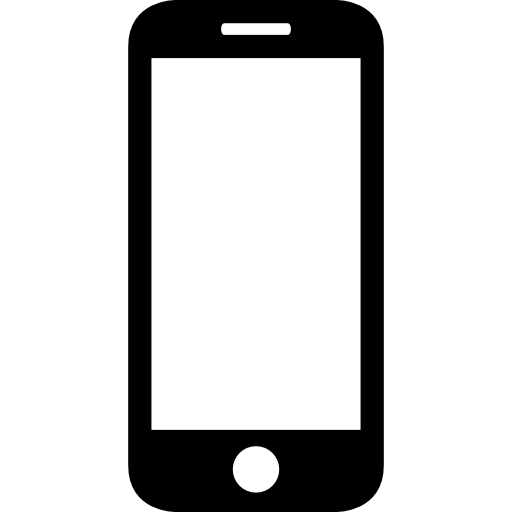
Au début du projet, les sous-objectifs suivants ont été définis :

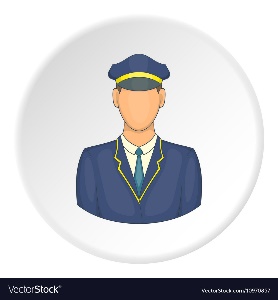
* Comparaison Native Apps /Web Apps : choix du mode d’implémentation.
* Réalisation du POC : idéalement il s’agirait de modéliser les communications entre deux employés de postes différents *(par exemple accompagnateur de train et conducteur de train)* à l’aide de notifications push.

Communication par notifications push

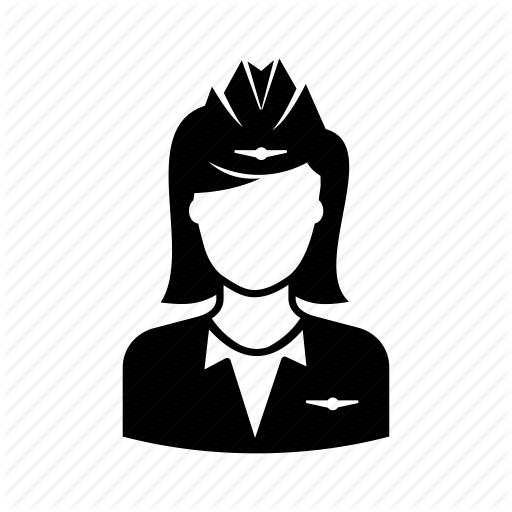


Send Notification





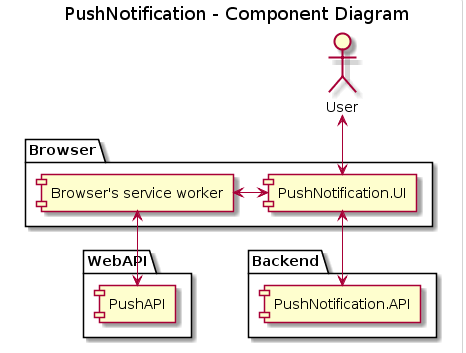
Show Notification



Dans la mesure où le projet aurait bien avancé, des objectifs secondaires ont été évoqués :

* Intégration sur Azure: il s’agirait éventuellement de faire tourner l’application sur un serveur Azure, voire d’utiliser le service Azure Notification Hubs
* Stratégie de Cache/Rédaction du Manifest: répondre aux critères d’une **Progressive Web App :** choisir quels fichier mettre dans le cache pour pallier aux problèmes de réseau, rédiger la manifest.json pour que le système propose d’ajouter l’application à l’écran d’accueil.

# Architecture du projet



Architecture du projet

Le FrontEnd **(PushNotification.UI)** : il s’agit de simuler simplement le choix du poste d’un employé, afin d’envoyer des notifications. Les notifications peuvent être envoyées depuis cette interface. Cette interface a été réalisée en React.JS

Le composant charnier autour duquel les notifications Push fonctionnent est le **Service Worker.** Il s’agit d’un script javascript associé à l’origine ( ou url) , et tournant en arrière-fond. Ceci-lui permet de réagir aux évènements « push » même lorsque le navigateur n’est pas ouvert.



Interface basique

L’interface dispose simplement de quelques boutons pour l’envoi de notifications push :

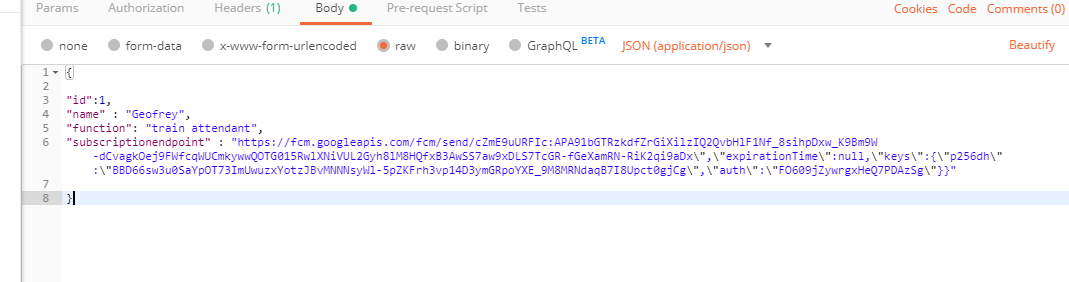
Un bouton pour autoriser l’envoie de notification sur le port en question

Un bouton pour enregistrer l’utilisateur à l’aide de la push API : ceci génère un “endpoint”, par lequel passera toutes les requêtes de notifications destinées au port en question

Un bouton pour changer de function ( conducteur de train/ accompagnateur de train )

Un bouton pour envoyer une notification

Le BackEnd ( **PushNotification.API)** est une Rest API simple réalisée avec asp.net core, sur Visual Studio , pour stocker quelques données sur l’utilisateur : nom, poste, endpoint de la “PushSubscription”. La Rest API a été réalisée en **.net core.** Les objets sont stockés de la manière suivante :



Idéalement, cette reste API contiendrait toutes les infos nécessaires à l’envoie de notifications aux employés.

# Description technique du rendu

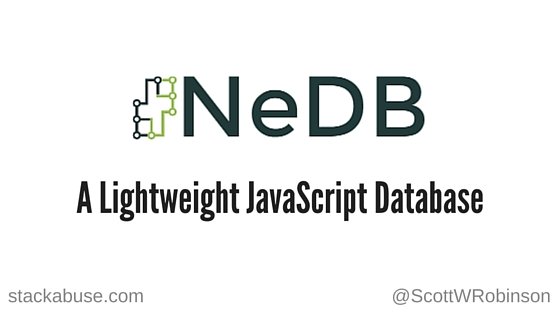
## Etapes de formation du POC

### La début du POC a été l’apprentissage du fonctionnement des services workers , ainsi que le fonctionnement des API utiles, notamment à l’aide du tutoriel Google\* , sur les Progressive Web Apps. Il s’agissait au début simplement d’afficher des notifications push à l’écran.

### Une fois cette étape passée, il a été possible de réaliser un projet basique, sans framework particulier, pour pouvoir envoyer des notifications à des utilisateurs particuliers. Les infos relatives aux utilisateurs étaient simplement stockées dans un fichier, et récupérées à l’aide de la librairie NeDB\*\*. Il s’agit d’une librairie de gestion de base de données sans setup particulier, et donc simple à utiliser.

*\** [*https://developers.google.com/web/ilt/pwa/*](https://developers.google.com/web/ilt/pwa/)

*\*\** [*https://github.com/louischatriot/nedb*](https://github.com/louischatriot/nedb)



Avec cette étape il était possible d’envoyer des notifications depuis un script distant :



Pop-up de notifications Push

Le principal problème est que rien n’était automatisé dans ce premier projet: il fallait renseigner manuellement le poste, le nom, faire un appel manuel au serveur.

L’idée pour la suite était de réaliser un projet s’apparentant le plus possible à veritable application, où les appels à la librairie push seraient réalisés en arrière fond : l’utilisateur devra uniquement decider de l’envoi de notification *(et si possible à qui elle est destinée)*.

### Enfin, pour plus de réalisme, le projet a ensuite été séparé entre un frontend recevant les notifications et un backend de type Rest API.

Pour le frontend, le framework MVC **React.js** a paru être un choix judicieux, car : *(extrait des docs React)*

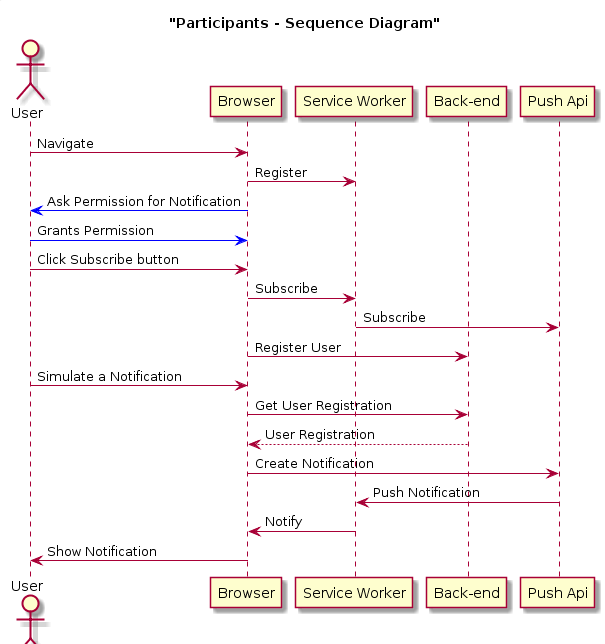
Il permet d’afficher des views qui s’actualisent rapidement lorsque leur état est modifié. C’est utile dans notre cas, notamment pour la manipulation des endpoints

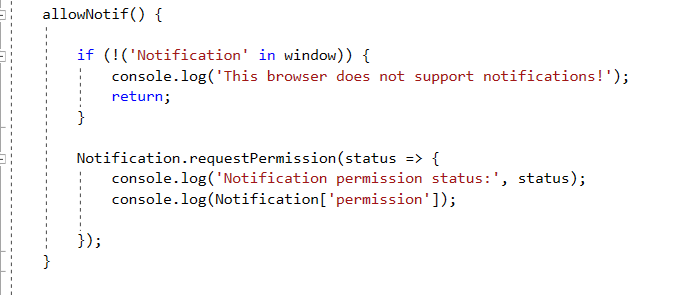
Les composants peuvent être encapsulés les uns dans les autres , ce qui permet de réaliser des interfaces avec plusieurs composants liés les uns aux autres sans trop de difficultés

Pour le backend, l’idée était de stocker les informations relatives, à un utilisateur dans un premier temps , nécessaires à l’envoie d’une notification Push personnalisée *(nom, poste, endpoint)*. Celui-ci a été réalisé avec **asp.net core,** dans l’IDE **Visual Studio,** notamment parce qu’un développeur expérimenté pouvait m’aider avec ce framework.

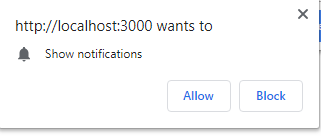
## Aperçu de certaines parties du code

### Demander l’autorisation à l’utilisateur de recevoir des notifications:

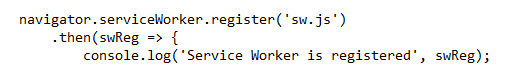


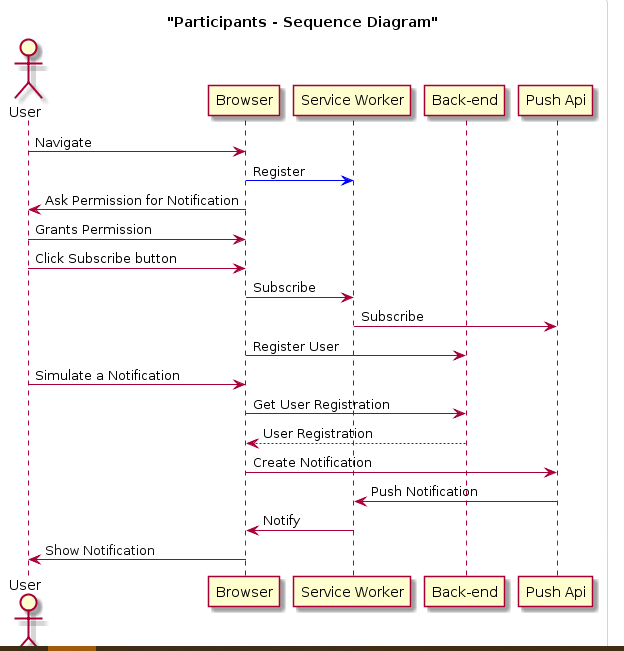


Un pop-up comme le suivant apparaît sur la page:

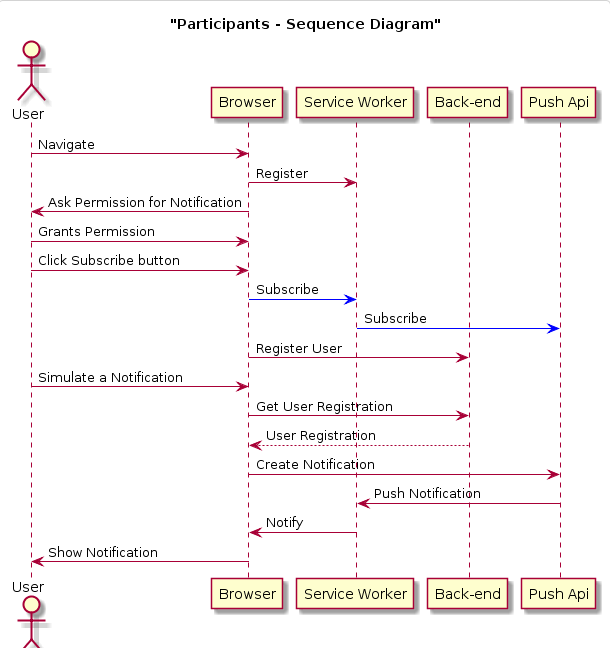


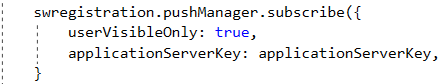
### Enregistrer le service worker: c’est ce fichier qui permettre au browser de réagir à un évènement push, même lorsque le navigateur est offline:



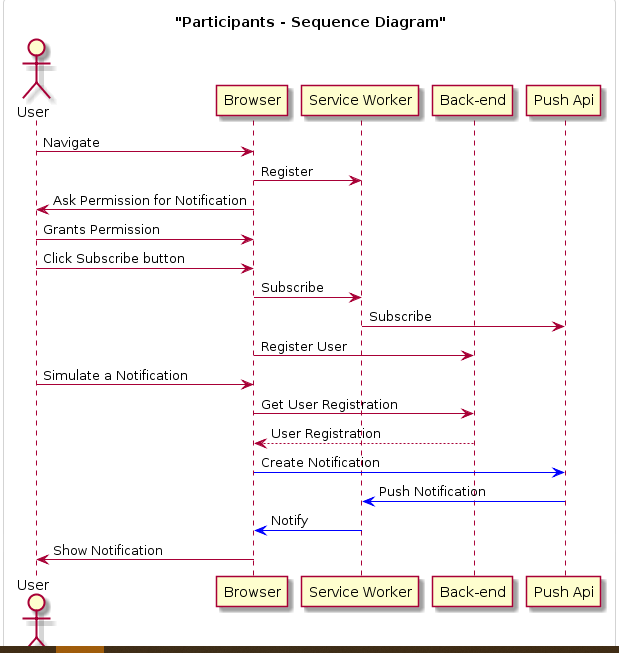


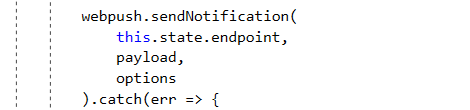
### S’enregistrer à un “service push”: selon le navigateur dans lequel c’est réalisé, l”endpoint” résultant, qui est en quelque sorte l’adresse virtuelle de l’appareil, sera différent. C’est cet “endpoint” qui va permettre au serveur push d’envoyer une notification.





### Envoyer la notification

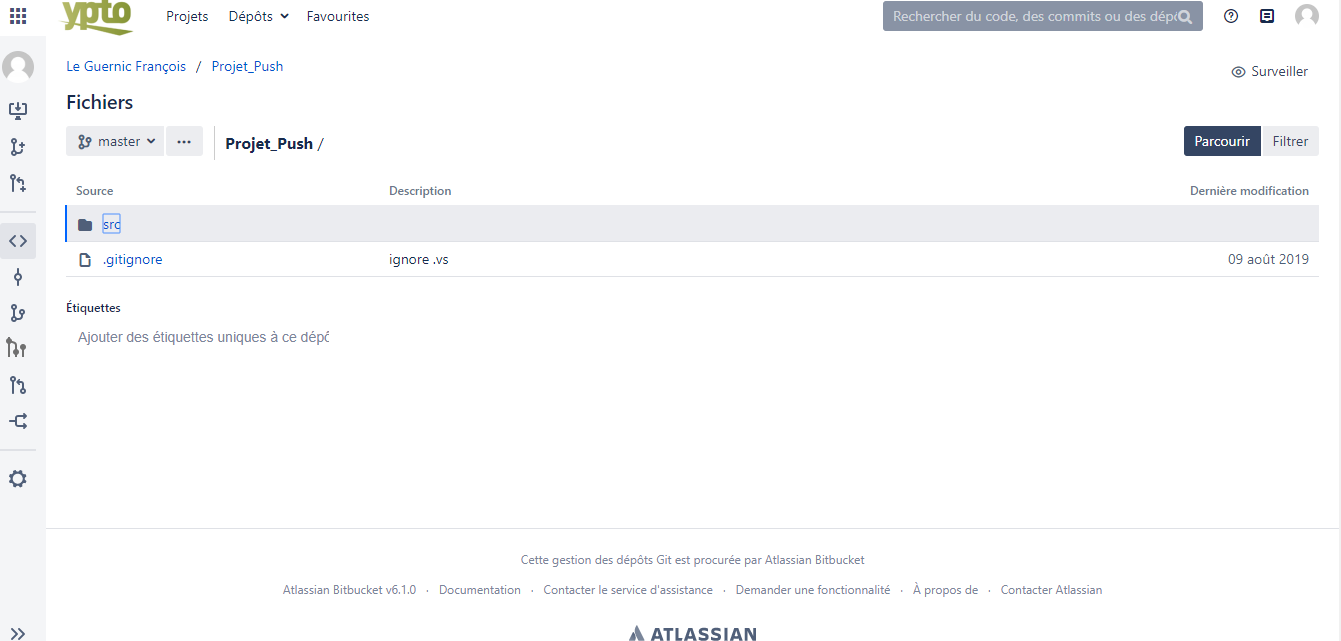




## Outils externes utilisés

### L’outil Git pour historiser et sauvegarder le code.

Le code source du projet en deux parties a été sauvegardé sur un repository sur le Bitbucket interne de l’entreprise

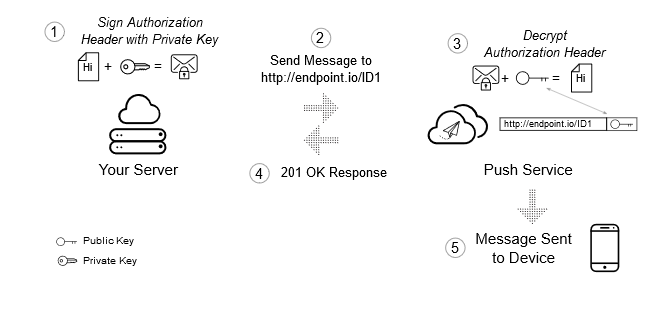


### Utilisation d’une librairie Web Push\*

Il s’agit d’une librairie **Node**, ce qui a donc paru le choix judicieux afin d’intégrer l’envoie de notification au front-end.

L’intérêt d’utiliser une librairie externe ( telle que la librairie Node web-push ) est de simplifier le code car la librairie se charge de de signer les messages ( ajouter un header seulement déchiffrable avec la clé publique envoyée à l’endpoint ) , et de les encrypter, dans e cadre du protocole web-push :

\*<https://github.com/web-push-libs/web-push>

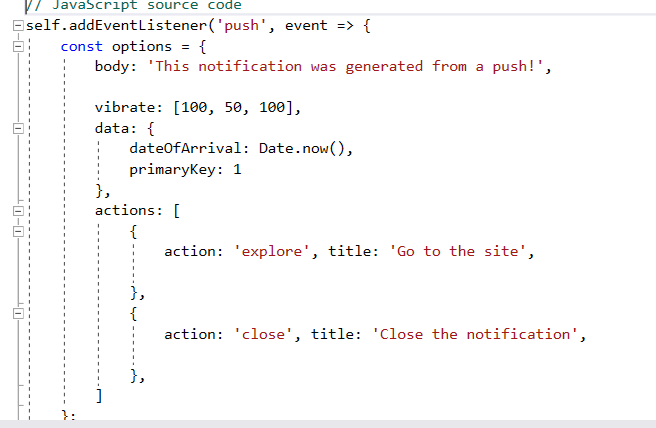
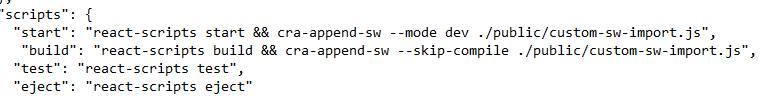


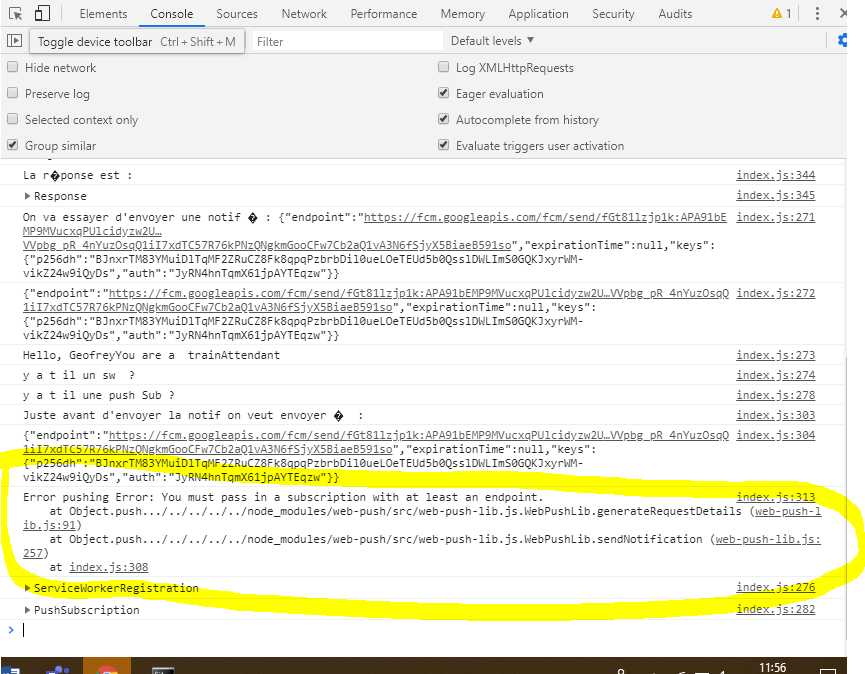
Les étapes de l’envoie d’un message push depuis un serveur sont:

* Le serveur envoie un push message , avec une private key servant à “signer” le message
* L’information signée est envoyée au push service en tant que header dans une requête POST
* Le push service utilise la public key générée avec le **pushManager.subscribe()** pour vérifier que l’information signée l’est avec la private key en relation avec la public key
* Si l’information est valide, le push message est envoyée au service worker de l’utilisateur

L’information signée envoyée est un JSON Web Token. La librairie se charge d’encrypter ce JWT.

## Difficultés rencontrées

* La prise en main des outils nécessaires a pris beaucoup de temps car il y avait beaucoup de choses nouvelles à assimiler:  
    
  Utilisation des différentes Web Api Utilisées sur le Web:  
  Service Worker API : pour l’enregistrement des services Workers  
  Fetch API : pour faire communiquer différentes API entre elles  
    
  Il a fallu assimiler la syntaxe de react et les basiques de C# pour asp.net core  
    
  Beaucoup de probmèmes techniques ont retardé grandement l’avancée du projet:  
  Ex:
  + Personnaliser le service Worker avec Create React App a posé problème:  
      
    Pour que le service worker puisse affiche rune notification Push :  
      
    Le problème est que la configuration de **“Create-React-App”**  fait qu’il est compliqué d’ajouter des fonctionnalités au service worker.  
    La recherche d’une librairie npm qui permet d’ajouter ce petit bout de code Durant le build process m’a retardé pendant un peu plus de 2 semaines.  
    
  + Respecter la Cors Policy  
    La **“Cross Origins Ressource Sharing”** sécurise les échanges de données entre origins différentes.  
      
    Pour permettre la communication entre le frontend et le backend, il a ainsi fallu specifier dans le code C# que ces échanges étaient autorisés.  
    Pour des raisons encore inexpliquées, il a aussi fallu modifier la structure des données dans la Rest API
  + Problèmes avec la librairie web-push  
      
    Le projet React peut recevoir des notifications envoyées depuis un serveur externe à l’application, mais il ne peut recevoir des notifications provenant du projet lui-même , ce qui est problématique car l’idée était de faire tourner le projet sur différents ports, pour mimer les communications de deux utilisateurs par l’intermédiaire de notifications Push.



Endpoint non reconnu par la librairie

# Etapes suivantes

Si je devais continuer mon projet , les étapes suivantes seraient :

Faire marcher les notifications push, uniquement de manière interne à l’application. Ainsi, il serait possible, ultérieurement, de lancer sur un port l’application en tant que conducteur de train, et sur un autre port en tant qu’accompagnateur de train ; puis les faire communiquer à l’aide de notifications push

Réaliser une progressive Web Apps

Faire tourner l’application sur un server Azure

Tester les notifications push sur téléphone

# Rétropsective sur le stage

Ce stage , bien qu’un peu intimidant au début par , m’a permis de me familiarizer avec de nombreux standards du web qui me seront sûrement utiles par la suite :

Cors policy

Fetch API

Progressive Web Apps

Push Notifications

Service Workers

Aussi j’ai pu apprendre les rudiments de React.

De manière assez contradictoire, si ce stage m’a permis de m’intéresser à de nombreuses choses, c’est surement aussi l’ampleur des choses à maîtriser qui m’a retardé, et me fait partir avec la deception de n’avoir pas pu mener mon projet à terme.

**Bibliographie**

<https://developers.google.com/web/ilt/pwa/> : le tutorial de google sur les Progressive Web Apps , c’est là que j’ai trouvé la plus grande partie des informations

<https://web-push-book.gauntface.com/chapter-01/01-introduction/> : le Web Push Book de Gaunt Face

<https://github.com/web-push-libs/web-push> : documentation de la librairie Web-Push

Mozilla Development Network : de nombreuses infos utiles sur les différentes API utilisées ( Service Worker API, Push API, Notificaton API )

La doc React

La doc .Net