

Joutes  
CPNV

**Auteurs**

Kevin Jordil

Benjamin Delacombaz

Table des matières

[Introduction 3](#_Toc3554347)

[Application 4](#_Toc3554348)

[Difficultés rencontrées 4](#_Toc3554349)

[Connexion 4](#_Toc3554350)

[Détection de domaine 4](#_Toc3554351)

[Asynchrone 4](#_Toc3554352)

[Améliorations apportées 4](#_Toc3554353)

[Gestion d’erreurs 4](#_Toc3554354)

[Gestion des pains grillés (toast) 4](#_Toc3554355)

[Gestion des routes 4](#_Toc3554356)

[Gestion des points d’accès 4](#_Toc3554357)

[Améliorations de l’expérience utilisateur 4](#_Toc3554358)

[Gestion de projet 5](#_Toc3554359)

[Choix de l’outils 5](#_Toc3554360)

[Git project 5](#_Toc3554361)

[Documentation techniques 5](#_Toc3554362)

[Api Requirements 5](#_Toc3554363)

[Mobile Store 5](#_Toc3554364)

[Build App 5](#_Toc3554365)

[Create release 5](#_Toc3554366)

[Custom toast 5](#_Toc3554367)

[Error Custom 5](#_Toc3554368)

[Routes Provider 5](#_Toc3554369)

[Update dependencies 5](#_Toc3554370)

[Continuation du projet 5](#_Toc3554371)

[Notifications 5](#_Toc3554372)

[Amélioration des providers 5](#_Toc3554373)

[Migration vers Ionic 4 5](#_Toc3554374)

[Amélioration du mode hors-ligne 5](#_Toc3554375)

[Conclusion 5](#_Toc3554376)

# Introduction

Lors de l’attribution des projets pour MAW 2.1, nous avons reçu le projet Joutes avec comme chef de projet Monsieur Chevillat. Cependant, Jérome Chevillat n’était pas le chef de projet de Joutes pour les autres années.

Nous avons commencé par lire la documentation des anciens élèves ayant travaillés sur le projet, elle explique bien les points exécutés pendant le projet et comment il fonctionne. Cependant il manquait une partie assez importante, l’état du projet, car aucun outil de gestion de projet n’était mentionné. Certes, après quelques configurations standard nous avons réussi lancer le projet et constater de nous-même l’état.

# Application

Evènements

# Difficultés rencontrées

## Connexion

Une fonctionnalité de connexion au joutes était en train d’être développée. Hors pour se connecter au joutes, il faut utiliser SAML.

SAML est une technologie qui permet de centraliser l’authentification sur un service au lieu d’avoir une authentification différente sur chaque application.

Dans notre cas, nous ne pouvons donc pas créer notre propre système de login, il a donc fallu utiliser un navigateur intégré à l’application pour ouvrir la page de login SAML de l’intranet. Hors à ce moment-là une erreur survient. Il est impossible d’ouvrir la page d’authentification en mode iframe.

Nous cherchons donc à trouver un autre moyen de se logguer, par exemple d’utiliser le navigateur de l’appareil et non celui intégrer dans notre application. Mais là, nous sommes identifiés dans le navigateur et pas dans l’application Joutes.

La seule solution trouvée pour se logué fut d’enlever le header « X-Frame-Options :SAMEORIGIN » dans les fichiers de SimpleSamlPHP de l’intranet.

Sans ce paramètre, il est désormais possible d’ouvrir la page d’authentification directement avec le navigateur intégré de l’application.

## Détection de domaine

Inappbrowser ne fonctionne pas avec ionic serve

## Asynchrone

L’asynchronisme du JavaScript est très puissant mais peut également poser certains problèmes. En effet, nous avons été confronté à un soucis lorsque nous avons voulu ‘’lié’’ notre gestion des points d’accès avec les providers / services que les groupes précédents avaient mis en place. Le système de récupération des données essayait de récupérer les données avant que notre gestion des points d’accès ait fini de charger, le service n’avait donc pas de point d’accès pour récupérer les données. Nous devions donc faire attendre le constructeur du service principal que notre provider de points d’accès soit prêt. Dommage, il n’est pas possible de faire de l’asynchrone dans les constructeurs[to be continued]

# Améliorations apportées

## Gestion d’erreurs

Le message d’erreur récupéré lors de l’échec d’une requête http n’était pas assez explicite.

bad_error_msg

Nous avons donc décidé de modifier ce message d’erreur par le suivant qui, quant à lui, est bien plus explicite :

good_error_msg

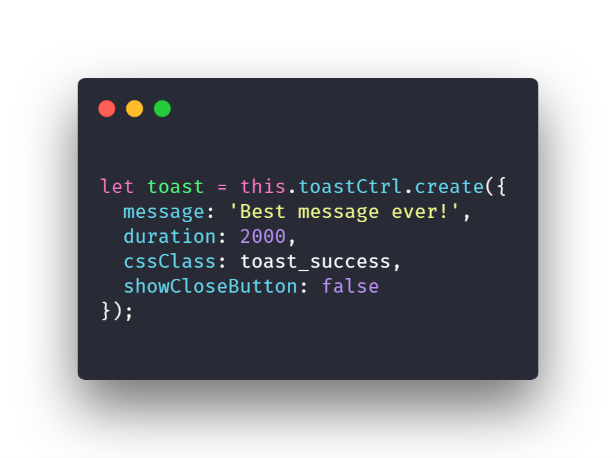
Nous nous sommes dit que si un cas similaire ce produirait, il serait utile d’avoir un outil permettant de gérer les messages d’erreur. C’est pourquoi nous avons créé le provider ‘’ErrorCustomProvider’’ cette classe nous permet d’utiliser la fonction statique ‘’getBetterMessage’’ qui, selon le nom de l’erreur, renvoie le message que l’on a défini. Si par hasard le message d’erreur que vous voulez modifier n’est pas traité, le message d’erreur de base sera affiché.

Voici un cas concret d’utilisation de cet outil :



## Gestion des pains grillés (toast)

La création d’un toast est assez simple, il suffit déclarer une variable en utilisant le toast controleur.



Le problème survient lorsque nous voulons changer les informations de ce toast nous devrions accéder à chaque attribut pour pouvoir les modifier.

C’est pourquoi nous avons créé le composant ‘’ToastCustom’’ qui nous permet de faire abstraction de la déclaration du toast dans les pages. Il nous suffit d’utiliser la méthode ‘’showToast’’ de notre composant.



## Gestion des routes

L’application a été pensée pour peut être utilisées par plusieurs écoles, dans cette optique-là, nous avons pensé qu’il faudrait des critères d’API pour être sûr que l’application fonctionne correctement. Cependant, après réflexion, nous nous sommes dit qu’il serait plus judicieux de faire un tableau JSON avec le nom des routes et l’école en question renvoie ses routes. Exemple :

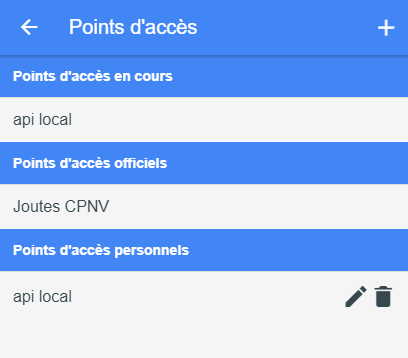


Ensuite il a fallu créer un service qui permet de renvoyer l’url de l’application en fonction de la route de l’application courante.

Lors de la méthode de demande d’url sur le service, on doit fournit en paramètre les différents identifiants ({event}, {notification}) pour avoir une url utilisable.

## Gestion des points d’accès

Lorsque nous avons repris le projet, la gestion des points d’accès se faisait ‘’en dur’’ dans le code. Si nous avions besoin de nous connecter à un autre point d’accès, l’application devait être redéployer. Nous avons donc développé la gestion des points d’accès qui permet d’ajouter des points accès personnel ce qui est assez pratique pour le développement et pour gérer différentes versions d’API.



Cette liste est stockée dans le stockage local du navigateur. Le point d’accès officiel ‘’Joutes CPNV’’ est défini ‘’en dur’’ dans le fichier des variables globales (/src/app/app.const.ts) et est ajouté et sélectionné par défaut au premier lancement de l’application. Pour ajouter ou changer de point d’accès, celui-ci doit être disponible et compatible. Pour qu’un point d’accès soit compatible la page de base de l’api doit retourner un numéro de version qui doit être identique à celle qui se trouve dans le fichier des variables globales et également correspondre à la structure que l’on retrouve également dans ce fichier.

## Améliorations de l’expérience utilisateur

Lorsque nous avons reçu le projet, nous étions un peu perdus dans l’application, c’est pourquoi nous avons décidé d’améliorer l’expérience utilisateur en simplifiant et en rendant plus compréhensible les diverses informations. La différence est visible sur les screenshots :

|  |  |  |
| --- | --- | --- |
| **Avant** |  | **Après** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Les informations affichées sont toujours les même exceptés pour le menu. Le but second était aussi d’uniformiser l’application et donc d’avoir les mêmes couleurs et styles partout.

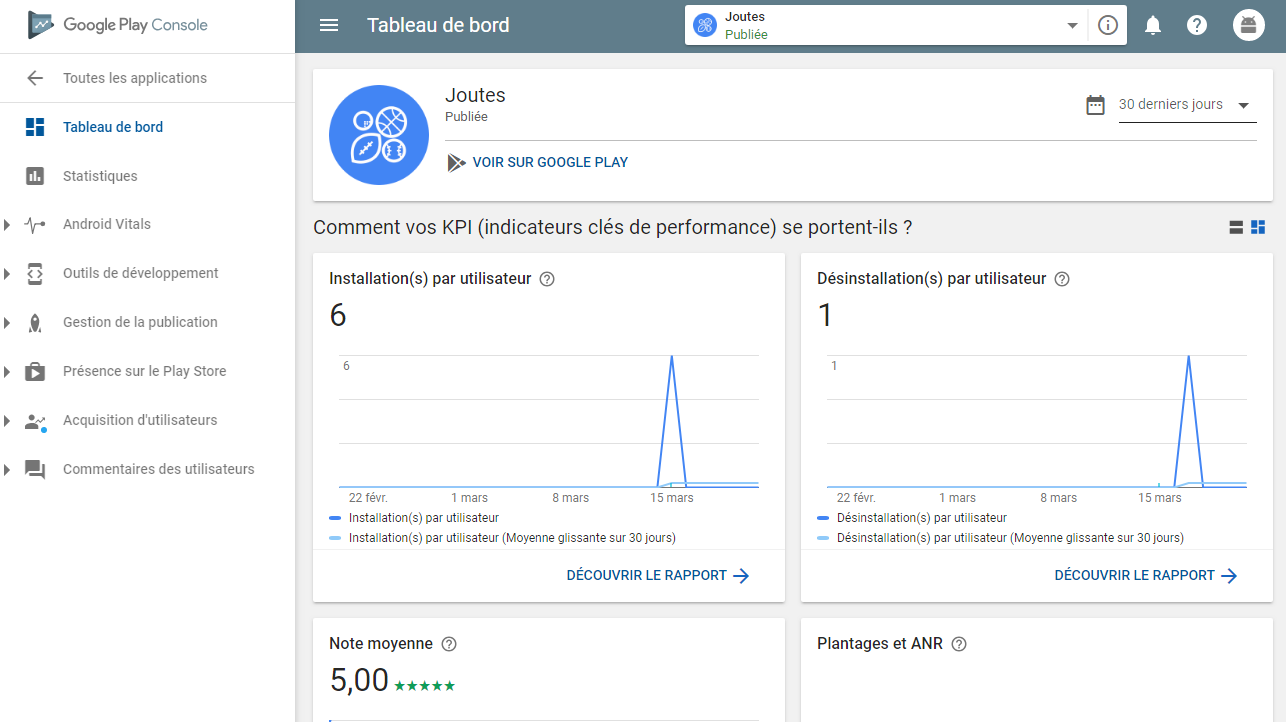
# Déploiement de l’application

## Google Play Store

Avec un compte Google, nous pouvons nous inscrire en tant que développeurs Google, pour cela, il faut régler un montant de 25 dollars pour finaliser l’inscription.

Nous avons directement accès à la console de déploiement d’application. Par la suite nous pouvons ajouter le fichier APK signé et rempli des questionnaires pour définir ce que l’application contenait.

Une fois les contrôles passé l’application est publiée dans les 15 minutes qui suivent sur le Play Store. Sur le dashboard, nous pouvons désormais voir l’état de notre application et différents statistiques sur celle-ci.



## Apple Store

# Gestion de projet

## Client

Au début du projet, Monsieur Chevillat nous informe que le client est Monsieur Marc Dafflon et qu’il lui a envoyé un mail pour lui informer que nous débutions le projet. Après un certain temps sans réponse de Monsieur Dafflon, notre chef de projet envoya un nouveau mail mais aucune réponse à nouveau. La décision de Monsieur Chevillat fut de travailler sans et d’essayer de publier une version fonctionnelle sur le store d’ici la fin du projet.

Les améliorations étaient donc soit proposées par le chef de projet soit par nous-même. Ensuite elles étaient validées par le chef de projet.

## Choix de l’outils

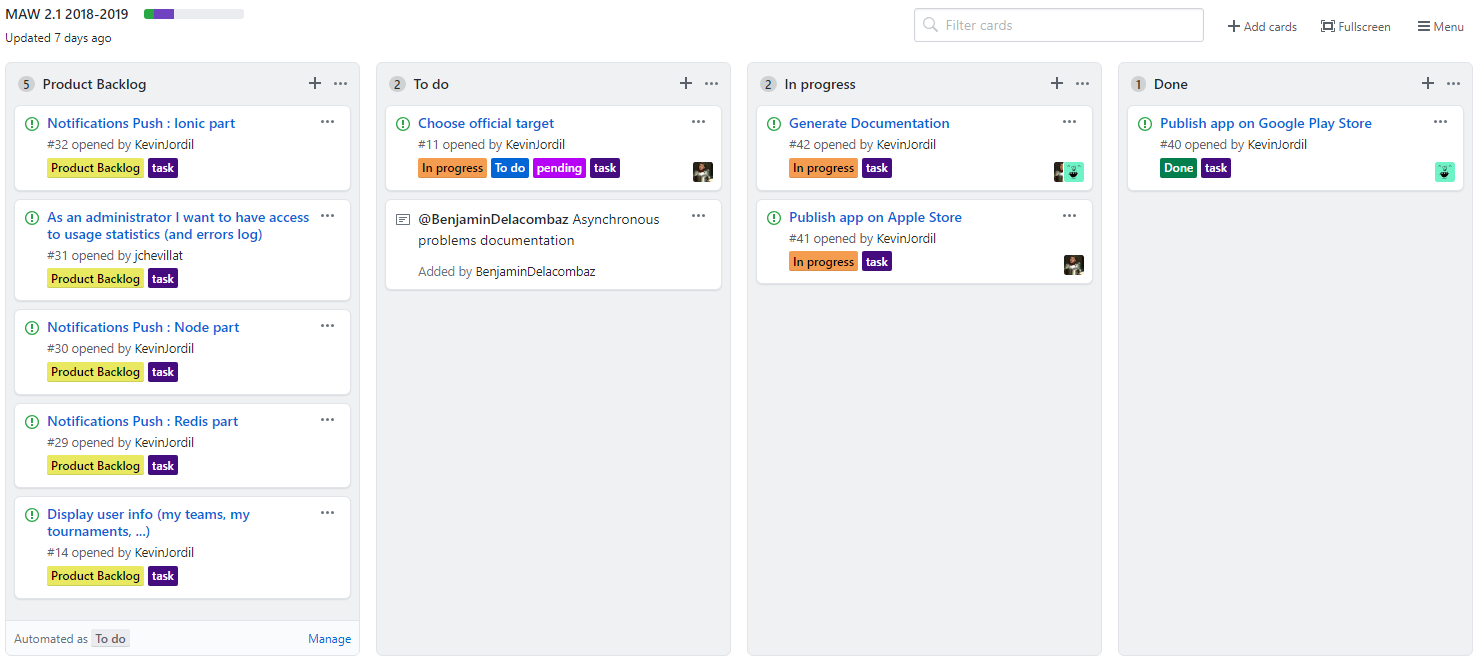
Lors de la réception de notre projet, nous n’avions pas de tableau de bord déjà créer, peut-être, qu’il est inexistant ou alors qu’il est sur Trello. Mais avec un nouveau chef de projet qui ne n’a pas d’expérience sur ce projet et qui ne savait donc pas où en était le projet. Nous avons donc dû créer notre propre gestion de projet.

Suite au problème rencontrés, nous avons trouvé judicieux de choisir git project, car tout est au même endroit, le code source, la gestion de projet et la documentation. De plus les issues créées sont directement intégrées dans la board de projet

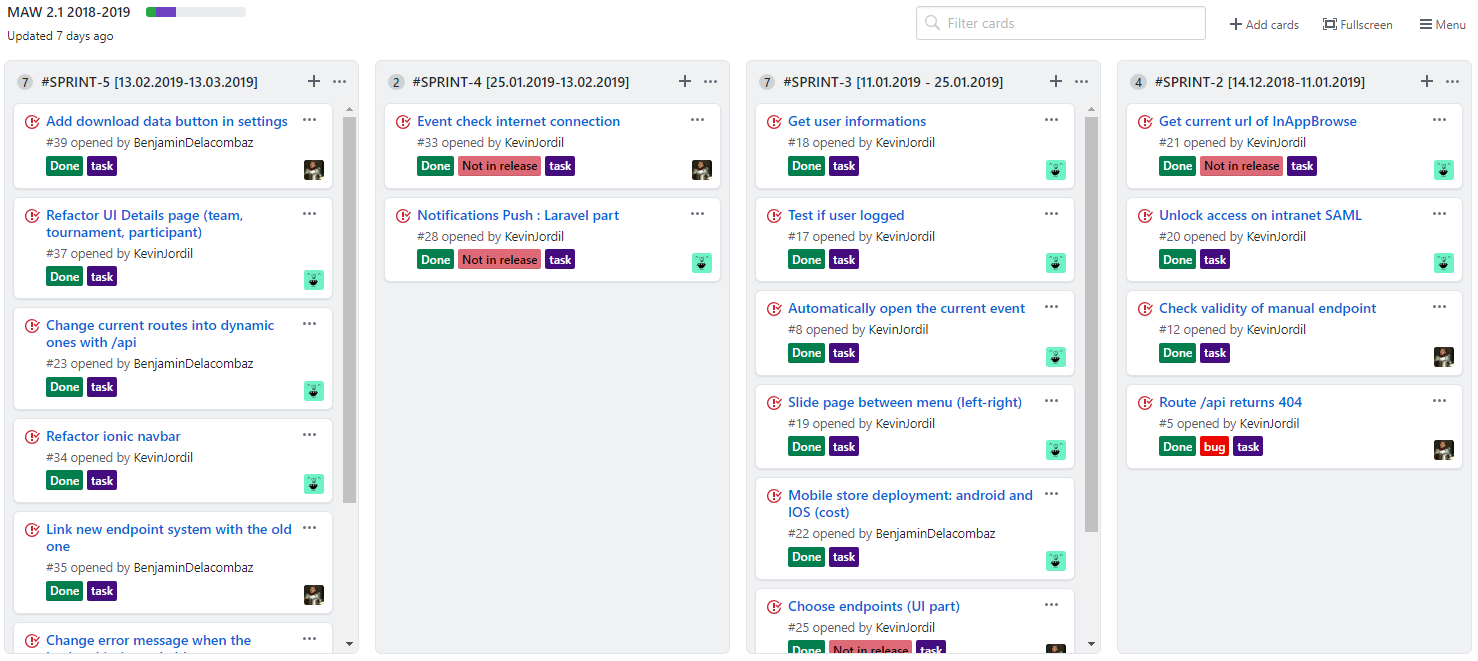
## Git project

En accord avec le chef de projet, nous avons choisi la gestion de projet en mode Kanban simple mais avec des colonnes spécifiques à chaque sprint avec les tâches liées terminées.

Chaque issue est une tâche qui arrive directement dans la colonne Product backlog.



Lors du début d’un sprint les différentes tâches prévues pour celui-ci sont mises dans la colonne To do et sont assignées à la personne si celle-là est déjà définie. En plus de cela un milestone est créé sur Github pour chaque sprint et donc assigné aux tâches à réaliser pour le sprint. Cela permet aussi d’avoir une vue regroupant uniquement les tâches d’un sprint spécifique.



# Documentation techniques

## Api Requirements

Cette documentation contient les prérequis de l’API pour le bon fonctionnement de l’application. [Lien](https://github.com/CPNV-ES/Joutes-Ionic/blob/master/docs/api_requirements.md)

## Mobile Store

Cette documentation contient le prix plus une documentation de mise en production sur chaque plateforme : Android et Apple. [Lien](https://github.com/CPNV-ES/Joutes-Ionic/blob/master/docs/mobile_store.md)

## Build App

Cette documentation contient la procédure pour générer le fichier apk de Joutes. [Lien](https://github.com/CPNV-ES/Joutes-Ionic/blob/master/docs/technical/build_app.md)

## Create release

Cette documentation explique les commandes nécessaires à la création d’une release sur Github. [Lien](https://github.com/CPNV-ES/Joutes-Ionic/blob/master/docs/technical/create_release.md)

## Custom toast

Cette documentation explique le fonctionnement et l’utilisation du composant de notifications personnalisés. [Lien](https://github.com/CPNV-ES/Joutes-Ionic/blob/master/docs/technical/custom_toast.md)

## Error Custom

Cette documentation explique le fonctionnement et l’utilisation du gestionnaire d’erreur personnalisé. [Lien](https://github.com/CPNV-ES/Joutes-Ionic/blob/master/docs/technical/error_custom.md)

## Routes Provider

Cette documentation explique le fonctionnement et l’utilisation du service de routes de l’API. [Lien](https://github.com/CPNV-ES/Joutes-Ionic/blob/master/docs/technical/routes_provider.md)

## Update dependencies

Cette documentation explique comment mettre à jour les dépendances du projet facilement. [Lien](https://github.com/CPNV-ES/Joutes-Ionic/blob/master/docs/technical/update_dependencies.md)

# Continuation du projet

## Notifications

## Amélioration des providers

## Migration vers Ionic 4

## Amélioration du mode hors-ligne

# Conclusion

15 minutes de pres + 5 de démo