Joutes Mobile



20.04.2018

CPNV, Ste-Croix

Ilias Goujgali

Struan Forsyth

Table des matières

[**1. Context**](#_wefc9bfo3h9m) **3**

[1.1. Introduction](#_xsg9dvrwqrmo) 3

[1.2. Durée et déroulement](#_dd0sb8e733e3) 3

[**2. Demande client**](#_fv26i2ugtay7) **4**

[**3. Ce qui a été fait**](#_qtqs308tkgfw) **5**

[3.1. API](#_b8exmxioc6l8) 5

[3.1.1. Modification de l’application mobile](#_fpv4iadvr50b) 5

[3.1.2. Modification de l’API](#_13tvojp7n1vw) 6

[Matchs](#_1mvvuw90srl) 6

[Pools](#_p361gqwfhww3) 6

[Classement](#_re821y68kbzd) 7

[3.2. Recherche globale](#_a70wzvl9uzr5) 7

[3.3. Tris alphabétique](#_2p7yfby3wads) 8

[3.4. Modification de l’ordre des matchs](#_dog6xwh12a2q) 8

[3.5. Offline storage](#_rvqcizoz7ujc) 9

[3.5.1. Cache](#_b6s7y4uabzjj) 9

[3.5.2. Local Forage](#_23u7xvbxok42) 9

[3.5.3. Sqlite](#_wkzbj8hyczex) 9

[3.5.4. Solution retenue](#_qqmtq1xbycjr) 9

[3.5.5. Implémentation](#_7kc9vw3lo5tt) 10

[3.5.6. Problèmes rencontrés](#_w8txx4bfonss) 13

[URI manquante](#_gvf7gxprer4c) 13

[Storage-service stature l’application](#_yqjluv8veye6) 13

[3.6. Mises à jour des technologies](#_c1gsxswlu3mn) 14

[3.6.1. Etapes](#_gnidl59wd3uj) 14

[3.7. Notification (POC)](#_j85lu3fszem) 14

[3.7.1. Implémentation](#_f55db4kgk2yv) 14

[3.7.2. Fonctionnement](#_ekxrswyp9ztk) 15

[3.7.3. Problème rencontré](#_fg4heir96073) 16

[Récupération des notifications](#_uvcpa2nl6q9a) 16

[**4. Technologies**](#_rin51n38gcvw) **16**

[**5. Annexes**](#_hmc3ue7xkw6c) **16**

# 1. Context

## 1.1. Introduction

Dans le cadre de notre cours de MAW 2.1, nous avons repris le projet de l’application mobile des joutes sportives. Les joutes sont organisées par le CPNV chaque fin d’année scolaire, il s’agit d’un ensemble de tournois qui permettent aux élèves, regroupés par équipe, de s’affronter dans différentes disciplines/sports comme par exemple au badminton. Le tout est organisé en poule afin de faire un classement à la fin de l’événement. La plupart des sports disponibles se pratiquent en équipe de 2 à 6 (excepté la marche qui est une activité individuelle)

Le but de cette application est uniquement d’être un moyen de communication vers les participants des joutes: savoir quand est-ce que l’on doit jouer, voir les membres des autres équipes etc. L’application récupère ses données depuis l’API développée par la team joute (2017 - si-t1a). Aucune modification ne peut-être apporter aux données de l’API depuis l’application mobile.

Une grande partie de l’application a déjà été réalisé par l’équipe joutes-app qui nous a précédé. Cependant, il restait des problèmes de compatibilités avec l’API (les projets n’évoluant pas simultanément ). C’est pourquoi nous tâcherons de palier à ces problèmes.

Dans ce document nous traiterons dans un premier temps des demandes qui nous ont été donnés, nous décrirons par la suite ce que nous avons mis en place pour combler les demandes, par quel moyen, les technologies utilisées etc les problèmes rencontrés et les solutions mises en place ou envisagées.

## 1.2. Durée et déroulement

Ce projet a été repris par Struan et Ilias de la SI-T2A. Il a été effectué à raison de 7 périodes par semaine du 13 novembre 2017 au 20 avril 2018 ( 19 semaines ), soit environ 200h (7 périodes \* 19 semaines \* 0.75 heure \* 2 personnes). Pour plus de précision qu’en au temps passé sur les différentes tâches, consultez le journal de bord fourni dans les annexes.

# 

# 2. Demande client

Le projet joute dans sa globalité, autant l’application mobile que le côté web app, est développé pour les maître des gymnastique du CPNV (clients) afin que l’organisation soient simplifier lors des joutes.

Le projet étant en grande partie déjà réalisé par la team joute-app précédente (2017), aucune réelle demande n’a été formulé excepté le fait de pouvoir consulter les données lorsque l’application n’est plus connecté à Internet (un mode offline). Hormis cela, quelques petites demandes d’ergonomies ont également été formulées par M.Daflon comme par exemple le système de recherche global; vous pouvez les consulter dans les PV en annexes.

# 

# 3. Ce qui a été fait

## 3.1. API

### 3.1.1. Modification de l’application mobile

Étant donné que nous avons repris un projet déjà existant, il nous a fallu un peu de temps pour la prise en main, ainsi que la mise en place de l’environnement de travail.

Une fois l’application en marche nous avons dû la mettre à jour pour qu’elle puisse fonctionner avec la dernière version de l’API développée en été 2017 par les anciens première année.

Les problèmes étaient petits et rapidement corrigés. La plupart des problèmes étaient du fait que l’API ne retournait pas exactement ce qui était attendu. Dans l’exemple ci-dessous on peut voir que la différence est mineur mais elle a un grand impact sur le fonctionnement de l’application.

|  |  |
| --- | --- |
| ***Retour de l’API*** | ***Retour attendu de l’API*** |
|  |  |

### 3.1.2. Modification de l’API

Pendant le développement de l’application mobile, nous avons remarqué que l’API ne retournait pas toutes les informations nécessaires pour le bon fonctionnement de l’application. Pour remédier cela, nous avons apporté plusieurs modifications à l’API.

#### Matches

Une partie importante de l’application est la visualisation des prochains matches ou des scores finals des matchs. Malheureusement, l’API était hardcodée pour nous retourner une tableau vide.

Pour résoudre ce problème, nous avons apporté quelques modifications à des fichiers existant pour ajouter des relations et inclure les matches dans la réponse de l’API. Nous avons aussi dû créer un transformeur pour gérer la structure de la valeur de retour pour un matche. En plus des valeurs d’un match, nous avons aussi inclus l’id de du pool dans laquelle le match se trouve.

#### Pools

En ce qui concerne les pools, tout comme les matches, l’API était hardcodée pour nous retourner une tableau vide de pool lorsqu’on faisait une requête pour avoir les informations d’un tournoi. Il manquait aussi tout le système pour nous retourner les informations d'un pool unique. C'est-à-dire, route et transformeur.

Pour la valeur hardcodée nous avons modifié le transformeur correspondant pour inclure la liste des pools associées à ce tournoi. Un transformeur pour gérer la structure de retour d’un pool a aussi été créé.

Pour que l’API nous retourne les informations d’un pool unique, il a fallu ajouter la route correspondante. Une fonction pour récupérer un pool d’un tournoi a aussi été ajoutée au modèle du tournois. Pour finir, un contrôleur et deux transformeurs ont été créés. Le contrôleur nous retourne le pool d’un tournois. Un des transformeurs gère la structure de retour d’un match pour le pool.

Par la suite nous avons remarqué qu’il y avait un problème avec les pools qui n’avaient pas encore d’équipe enregistrée. Dans ce cas le serveur nous retournait une erreur 500 et notre application s'arrêtait subitement.

Nous avons ouvert une issue sur le Github de l’API (<https://github.com/CPNV-ES/Joutes/issues/1>) demandant la correction de cette erreur. L’équipe concerné a résolu le problème le 16.03.2018.

#### 

#### Classement

Pour le classement, comme pour les deux au-dessus, l’API était hardcodée pour nous retourner une table vide par rang (premier, deuxième, troisième). Ceci était le cas pour deux requêtes vers l’API. Celle pour avoir les tournois et une autre pour avoir le détail d’un tournois.

Pour remédier ce problème nous avons dû créer une fonction dans le fichier modèle pour un tournois. Pour finir, deux transformeur ont été modifié pour gérer la structure de retour et inclure le classement.

## 3.2. Recherche globale

Une des demandes du client était un moyen de faire une recherche globale sur les équipes, participants et tournois en même temps. Avec l'ancienne application, il était possible de faire une recherche sur ces trois éléments, mais pas en même temps.

Pour réaliser cette demande, une page supplémentaire a été créée. Grâce à une petite icon de recherche dans la bar d’outils, cette page est accessible depuis toutes les page lorsqu'un événement est sélectionné.



## 

## 3.3. Tris alphabétique

Pour faciliter la lecture et la visualisation des listes, il a été décidé de les trier par ordre alphabétique. Pour cela nous avons simplement utilisé la fonction Javascript “sort” et “localeCompare”.

|  |
| --- |
| ***Tris des participants*** |
|  |

## 3.4. Modification de l’ordre des matchs

Pour que l’utilisateur puisse rapidement voir quel sont ses prochain matches, il a fallu modifier leurs ordre d’affichage. En premier, nous affichons les matches qui n’ont pas encore été terminés, suivi des ceux terminés.

|  |
| --- |
| ***Liste des matchs*** |
|  |

Pour nous aider à faire ce tri nous avons utilisé la réponse “Generic, Powerful Answer” dans ce post Stackoverflow:

<https://stackoverflow.com/questions/10123953/sort-javascript-object-array-by-date>

## 

## 3.5. Offline storage

Auparavant un smartphone n’ayant pas de connexion internet aurait été dans l’incapacité de consulter les informations sur les joutes. Et cela même s’il s’était connecté auparavant à l’API. C’est pourquoi, une des fonctionnalités qui nous a été demandées est le mode offline de l’application. L’objectif était simplement de pouvoir continuer de consulter les informations au travers l’application si l’appareil mobile se trouvait couper d’internet.

Comme l’avait renseigné l’équipe du projet l’année passée, 3 solutions s’offre à nous:

### 3.5.1. Cache

Le cache est la solution la plus simple et rapide à mettre: à chaque requête le système conserve le contenu renvoyé par le serveur. Le problème que nous avons perçu à cette solution est le fait de récupérer le contenu uniquement lorsqu’une requête est faite à l’API autrement dit vous ne pourrez pas consulter une page sur laquelle vous n’êtes pas aller auparavant .

### 3.5.2. Local Forage

Local Forage est une bibliothèque simple et rapide qui permet de stocker des informations, elle est très similaire à Local Storage, ces deux méthodes de stockage utilisent le principe de Key-Value, elle a l’avantage contrairement à une base de donnée SQL conventionnelle de stocker les données telles qu’elles sont affichées. Dans notre cas on utiliserait l’URI de la requête comme clé et le JSON retournée comme valeur à stocker.

### 3.5.3. Sqlite

La troisième option que nous avons considéré est l’utilisation d’une base de données SQL. Nous avons alors pensé à Sqlite. Le problème est que cette solution aurait impliquer pas mal de code notamment pour faire notamment la façon dont sont récupérées les données une fois hors-ligne.

### 3.5.4. Solution retenue

Nous avons finalement retenu la solution Local Forage car elle est moins compliqué à mettre en place d'une base de données SQlite et nous offre plus de flexibilité que du cache.

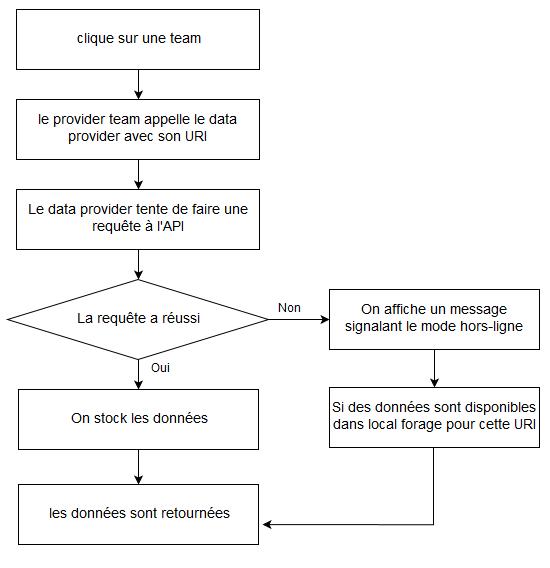
### 3.5.5. Implémentation

Pour ce faire nous avons commencé par modifier le code existant afin que chaque réponse de l’API soit enregistrée également.

Voici un diagramme qui explique grossièrement comment un requête était faite à l’API (Dans ce cas, on demande à récupérer les informations d’une team spécifique)

### 

Enfin voici, le diagramme une fois modifié :

:

on peut observer les deux opérations supplémentaires “on stock les données” et “si les données sont disponibles dans local forage pour cette URI”

De plus, nous avons implémenter un service: Storage-service, ce dernier avait pour but d’aller à interval régulier questionner l’API pour récupérer les données que nous avions jugé comme essentiel c’est-à-dire:

* Evénement
* Pools
* Tournoi
* Participants
* Équipe

Dans cette idée, nous avons créer un arbre qui décrivait les URLs à requêter.



Pour chaque noeud de l’arbre, 3 propriétés sont présentes

* URL: correspond à l’URI a allé questionner, il est possible de mettre un id entre accolade - {idEvent},
* Key, correspond à clé de la réponse à utiliser dans la réponse
* Children, un array des mêmes éléments décrits au dessus

Ce service devait être ensuite lancé tous les x temps (10 minutes par exemple). Afin de garder une version à jour des données. Nous avons cependant rencontrés des problèmes ces derniers sont décrits dans la section suivante.

### 3.5.6. Problèmes rencontrés

Nous avons rencontrés deux problèmes lors de la réalisation de ce service.

#### URI manquante

Il n’existait pas d’ URI permettant d’afficher une liste de pools pour un tournoi donné. ce qui a été un peu handicapant pour la création de notre service.

Solution

Nous avons réussi à trouver un code de contournement et avons fait un issue sur le repository du backend des joutes.

De plus, nous avons ouvert une issue sur le Github de l’API (<https://github.com/CPNV-ES/Joutes/issues/2>) demandant l’ajout de cette URI. L’équipe concerné a résolu le problème le 14.03.2018.

#### Storage-service stature l’application

Le problème majeur est que le storage-service envoie beaucoup de requêtes d’un coup (environ 500~ avec les données de tests actuelles) ce qui a pour effet de paralyser la navigation dans l’application. En effet, lorsqu’on navigue sur une page la requête nécessaire à l’affichage de la dite page se retrouve en-bas de la queue, elle doit attendre donc la fin des autres requêtes pour être envoyée.

Solution provisoire

Ayant remarqué ce problème trop tard, il nous a été impossible de changer notre code pour une solution plus viable. Nous avons décidé d’enlever la mise à jour régulière des données du local-forage et l’avons remplacée par un bouton qui, une fois cliqué, effectue l’opération de téléchargement du contenu en avertissant l’utilisateur que cette opération peut prendre un certain temps et qu’elle paralysera l’application.

Solutions envisageables

La solution provisoire est bien évidemment pas idéal et devrait être remplacer. Nous avons pour cela imaginer plusieurs possibilités :

1. Créer 3 - 4 URIs sur l’API permettant de récupérer tout le contenu d’un tournoi.
2. Trouver un moyen de prioriser les requêtes: les requêtes venant d’un changement de page serait exécuté avant toutes les autres.
3. Créer un système Publish-subscribe avec le serveur, ce dernier se chargerait d’envoyer les modifications aux différents devices qui se seraient connecté à l’API au préalable.

## 3.6. Mises à jour des technologies

Pour cette mise à jour, nous sommes passés d’Angular 2.1 et Ionic 2.0 vers Angular 5.0 et Ionic 3.9. La décision de mettre à jour l’application a été prise pour pouvoir implémenter le module d'extension de notification.

Pour nous aider dans ce processus, nous avons trouvé une marche à suivre avec les différentes étapes à effectuer pour que tout se déroule avec le moins de problèmes.

### 3.6.1. Etapes

1. Mise à jour du “package.json”
2. Suivi du guide pour la mises à jour d’Angular
3. Mise à jour du “app.module.ts”
4. Mise à jour vers “Ionic Native 3”

Pour plus de détails sur les étaps voici un lien vers le guide utilisé: <https://forum.ionicframework.com/t/guide-how-to-update-to-ionic-3-x/87516>

## 3.7. Notification (POC)

Nos prédécesseurs avaient pensé à une implémentation de notification afin de pouvoir avertir un utilisateur lorsqu’un de ses matches allait commencer. Cependant, ce choix nous a semblé peu judicieux dans la mesure où une équipe attend rarement plus de 30 minutes pour jouer un match.

En revanche, nous avons implémenté un “proof of concept” d’un système de notification plus générique qui aurait pour but d’avertir les utilisateurs d’un changement imprévu comme par exemple, un membre d’une équipe introuvable.

### 3.7.1. Implémentation

Celui-ci permet à un administrateur de l’application web de créer des notifications pour une équipe. Pour implémenter ce système, nous avons ajouté une nouvelle table “notifications” dans la base de données qui contient les colonnes suivantes: id, title, description, participant\_id et viewed.

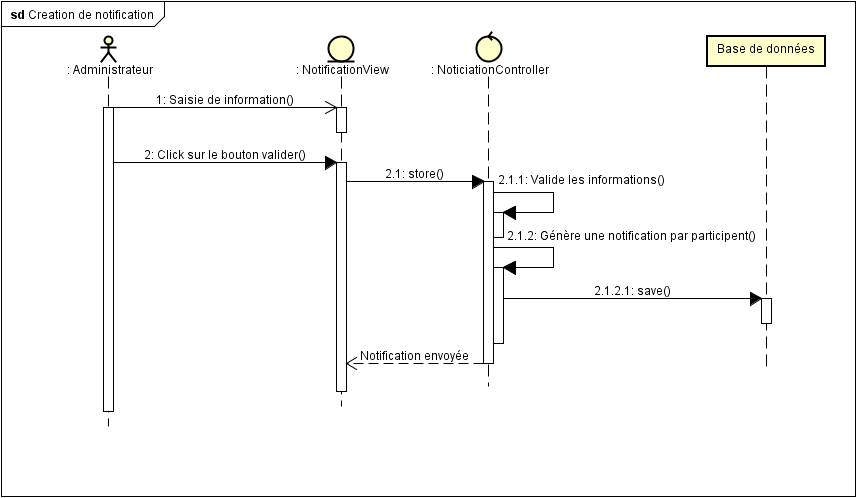
Pour créer une notification, nous avons ajouté une page sur le site web qui nous permet de saisir le titre, une description et de choisir l’équipe à laquelle nous voulons envoyer la notification.

La branche sur laquelle a été réalisée cette fonctionnalité est : POC-Notifications, elle n’a pas été fusionnées avec la branch master. Du côté back-end, vous pouvez consulter la page de création de notification: http://adresse\_serveur/notification/create

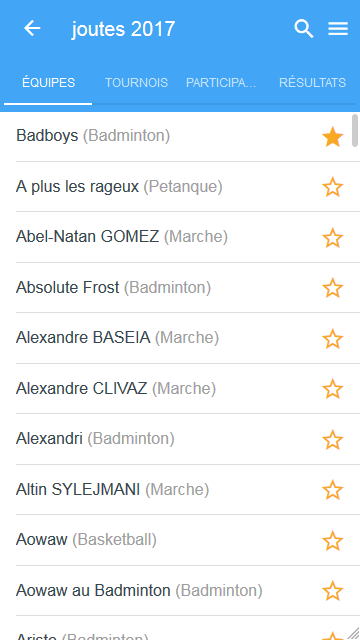
### 

### 3.7.2. Fonctionnement

Voici un schéma pour illustrer le fonctionnement de la création d’une notification pour une équipe.



Une fois les notifications créées l’application mobile pourra les récupérer, pour autant que l’utilisateur ait défini au moins une équipe préférée. Une équipe est définie comme préférée s’il y a une étoile remplie à côté du nom. Dans l’exemple ci-dessous nous pouvons voir que “Badboys” est une équipe préférée mais “A plus les rageux” ne l’est pas.



Lorsque la notification est récupérée par l’application de l’utilisateur, une requête est envoyée à l'API pour marquer la notification comme “vue” dans la base de données.

### 

### 3.7.3. Problème rencontré

#### Récupération des notifications

Nous avons constaté un problème avec la méthode que nous utilisons pour récupérer les notifications ou les marquer comme vue.

En ce moment, dans l’application nous identifiant une équipe préférée et non un participant. Donc, lors de la récupération des notifications, l’application récupère toutes les notifications pour une équipe et en conséquence les marques toutes comme vues.

Solution envisagée

De notre point de vue, la manière la plus correcte de résoudre ce problème serait d'implémenter un système d’authentification via l’intranet (ou autre) comme proposé par la première équipe de développement.

Cela nous permettrait de récupérer les notifications pour une personne uniquement et d’envoyer une notification à une seule et unique personne.

# 4. Prise en main du projet

L’application mobile est disponbile dans l’état actuel sur le <https://github.com/CPNV-ES/Joutes-Ionic> avec le “tag” **v0.9.5.** Les informations relatives à l’installation et au lancement de l’application y sont également présentes.

Étant donné que notre application est dépendante de l’API. Nous avons créé un “Fork” sur Github avec les modifications pour que les notifications puissent fonctionner. La version qui fonctionne est celle avec le “tag” ***v1.1.***

Voici un lien vers la page Github: <https://github.com/LogLauncher/Joutes/releases/tag/v1.1>

Technologies utilisées dans l’application mobile:

* Ionic 3.9
* Angular 5.0

# 5. Annexes

* Journal de bord
* PVs