ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

TPI – Shoot4Stats

« SHOOT4STATS » APPLICATION WEB DE GESTION DE STATISTIQUES POUR LE TIR À L'ARC

ENAC-IT2
EPFL | Bonjour Mickael

Contents

1 Analyse préliminaire

Je vais ici décomposer le cahier des charges dans son ensemble et présenter les buts du projet

1.1 Introduction

Mon travail de fin d'apprentissage a pour but de permettre aux archers, une fois authentifiés à l'aide Facebook, d'enregistrer des données sur leurs Shoots (c'est-à-dire des *Ends* et des *Arrows*). Elle permettra de visionner ces données par des informations graphiques ou chiffrées.

Ce travail permet de fortifier mes connaissances en développement WEB et d'apprendre et voir comment se déroule le développement d'une application WEB de manière professionnelle et AGILE.

Le choix de cette application a été fait en corrélation avec le chef de projet à ma demande, pour répondre à des besoins personnels et peut-être même aux besoins des archers en règle générale.

1.2 Objectifs

Développer une interface WEB « Frontend » et « mobile first » puis adapter le « BackEnd » si besoin afin d'effectuer des tâches pour un utilisateur authentifié ou non.

1.3 Contexte

1.3.1 Structure d'un *Shoot*

Un *Shoot* est défini comme suit :

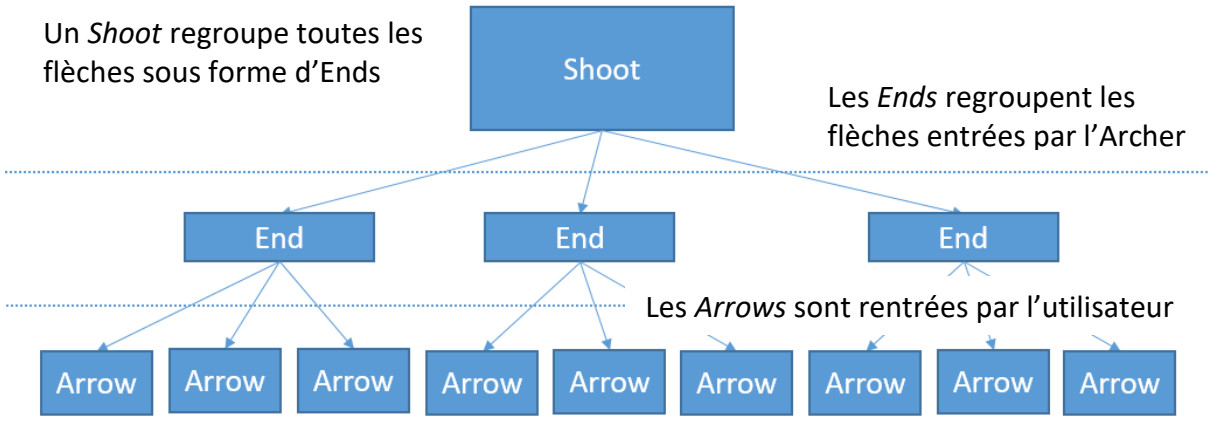


Figure 1 Schéma d'un Shoot

1.3.2 Remplissage d'un *Shoot*

La séquence de tir au tir à l'arc se passe comme suit ;

- L'on tire n flèches (*Arrows*) pour un **Training**. 3 *Arrows* pour les compétitions **Indoor** et 6 pour les compétitions **Outdoor**
- Ces n flèches se « regroupent » et forment une volée « *End* »
- Et après n volées (20 pour **Indoor**, 12 pour **Outdoor**) Le Shoot se termine.

1.3.3 Points au tir à l'arc

Voici l'image d'un blason de tir à l'arc officiel :

Figure 2 FITA Official Target face

Comme vous pouvez le voir, il y a une paire de cercle pour chaque couleur, et chaque cercle représente un point. Le dernier (le plus au centre) cependant représente un « Avantage » on appelle ça un « X » mais il vaut tout de même 10 points lors du décompte des points. Cependant dans ma base de données le « X » sera 11 et en frontend on représentera cela comme un X.
Lorsqu'une flèche ne va pas sur la cible l'on appelle ça un « Missed » (ou manqué) et cela vaut 0 point.

2 Analyse

J'analyse ici la difficulté du projet et explique le projet en globalité

2.1 Cahier des charges détaillé

2.1.1 Prérequis

Une partie du travail a déjà été réalisée en collaboration avec le chef de projet, soit :

- Modéliser, implémenter et documenter la base de données
- Mettre en place un environnement de développement
  - IDE (en l'occurrence Visual Studio Code)
  - Gestion de source (GIT)
- Développer la partie « BackEnd » sous la forme d'une API
  - Requêtes principales préalablement développées
  - Authentification externe via Facebook à l'aide de PassportJS

2.1.2 Descriptif du projet

- Les Archers « utilisateurs authentifiés » doivent pouvoir ajouter des *Shoots* complets et voir les détails de ceux-ci. Reprendre les *Shoots* s'ils ne sont pas terminés.
- Aux « administrateurs » (définis au sein de la DB directement) de pouvoir lister les Archers et avoir accès à leurs « Shoots ».

- Aux utilisateurs anonymes (non authentifiés) de voir les statistiques du dernier Shoot d'un utilisateur en ayant l'URL personnelle de l'utilisateur en question.

2.1.3 Logiciels et ressources à disposition

- Un PC :
  - OS : Windows 10 Education
  - IDE : Visual Studio Code
  - Langage : JavaScript
  - Version du serveur : 5.7.9 - MySQL Community Server (GPL)
  - Framework : NodeJS v.7.4.0 (API)
  - Librairies : VueJS (Frontend) / Axios / JQuery / Framework CSS Materialize / ExpressJS (BackEnd) / ORM Sequelize / PaspportJS gère l'authentification / Chart.JS (permet de mettre en place des graphiques simplement)
- Services :
  - Authentification externe via Facebook Login

2.2 Stratégie de test

2.2.1 Stratégie globale

Comme le cahier des charges le spécifie la méthode de développement utilisée doit être itérative (AGILE), voici donc les 4 valeurs fondamentales de la méthode AGILE :

- **L'équipe** ("Personnes et interaction plutôt que processus et outils")
- **L'application** ("Logiciel fonctionnel plutôt que documentation complète")
- **La collaboration** ("Collaboration avec le client plutôt que négociation de contrat")
- **L'acceptation du changement** ("Réagir au changement plutôt que suivre un plan")

L'on remarque que l'accent est mis sur le client, le faire participer un maximum au développement de l'application afin qu'il se sente concerné et que le produit fini lui convienne au maximum au lieu de se baser uniquement sur un cahier des charges fixe.

2.2.2 Procédure

Lors des gros ajouts de fonctionnalités je ferai tester à des personnes externes (archers, coach de tir) dans la limite du possible afin d'avoir un retour externe sur les tests (interface utilisateur et simplicité d'utilisation)

Puis de mon côté lors du développement j'effectuerai des tests tels que :

1. Explication du test
2. Résultats attendus
3. Résultats observés
4. Discussion des résultats / résolution des problèmes

Mes tests s'effectueront sur la validation des formulaires, la restriction relative aux pages de l'application, la récupération des informations sur l'API.

2.3 Etude de faisabilité

2.3.1 Risques techniques

La complexité technique réside dans la modularisation du code afin d'avoir une bonne évolutivité du code pour permettre les améliorations / ajouts de fonctionnalités futures de fonctionner rapidement grâce aux modules présents.

Ayant déjà fait le BackEnd en collaboration avec le chef de projet, ce concept de modularité a été appris à ce moment-là, je vais donc faire en sorte de bien séparer mes Vues et que les données soient bien partagées par composants afin de toujours savoir où l'on en est. Cependant c'est un concept encore assez nouveau pour moi.

2.3.2 Risques sur le planning

Je pense qu'au niveau du planning il ne va pas y avoir des gros problèmes tant que je ne m'éloigne pas de l'objectif principal (c'est-à-dire d'avoir une application fonctionnelle avant tout) et que je me concentre sur les points principaux.

Pour cela j'ai définit des priorités sur les tâches comme on peut le voir dans le Planning Initial, je ferai d'abord la partie « *Archers* », puis *Administrateurs* et enfin les *utilisateurs anonymes*. Cela s'avère vrai si le client n'exige pas un approfondi dans une partie.

Mais lors d'un développement AGILE le planning est bouleversé par les exigences du client au long du projet.

2.4 Planification

Voir en **Annexes** « Planning Initial ».

3 Dossier de conception

Je présente ici les moyens mis en œuvre et la conception du projet, cela comprend ce qui a été fait avant en collaboration avec le chef de projet.

3.1 Processus pour Archers

Le processus est décrit dans le manuel d'utilisation en **Annexes**

3.2 Base de données

La base de données fait partie des prérequis du projet, c'est-à-dire qu'elle a été faite en collaboration avec le chef de projet, soit Stefano Nepa.

3.2.1 MLD

Le MLD a été généré par MySQL WorkBench d'après le diagramme que j'ai conçu.

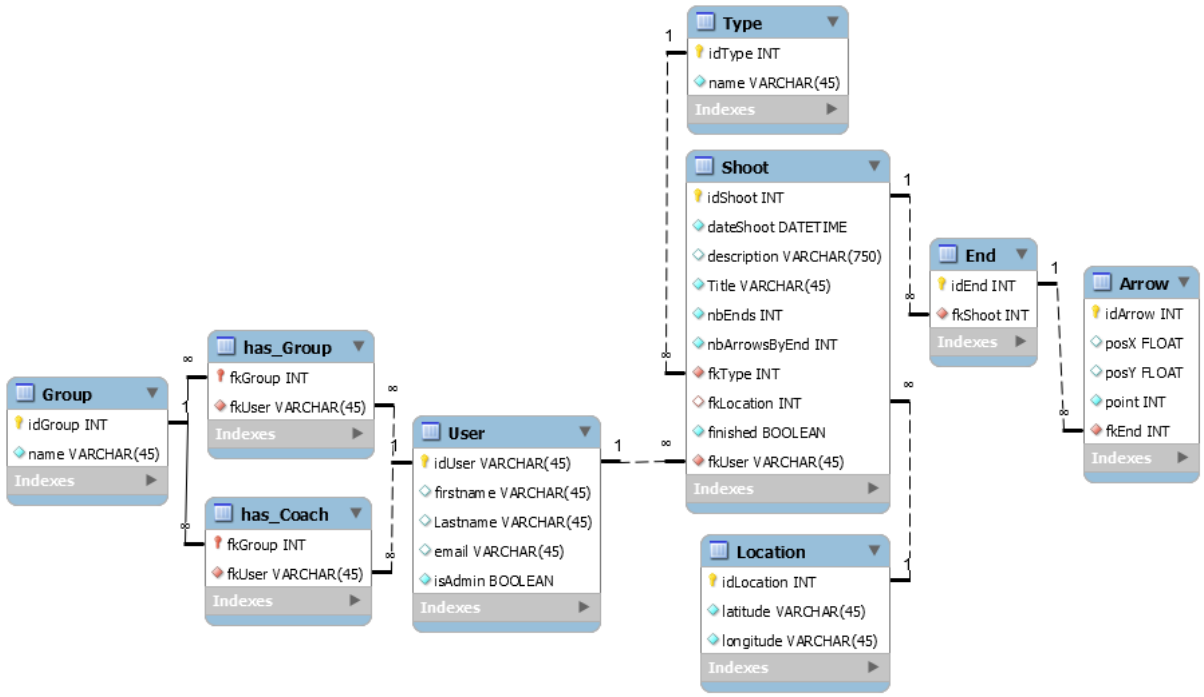


Figure 3 Modèle Logique des données

3.2.2 Dictionnaire de données

Voici la description de la base de données :

- **Shoot**
  Représente un Shoot.
  - **idShoot**
    Id du Shoot, auto incrémentiel
  - **dateShoot, description, title**
    Propriétés du Shoot, définition du Shoot
    DateShoot : valeur par défaut CURRENT_TIMESTAMP
    description et *title* sont rentrés par l'utilisateur
  - **nbEnds, nbArrowsByEnd**
    Le Frontend choisira automatiquement ces valeurs suivant le type du Shoot, sauf si le type est « Training » auquel cas ce sera à l'utilisateur de choisir ces chiffres

  - **fkType**
    Clé étrangère pour la définition du Type du Shoot
  - **fkLocation**
    Clé étrangère qui peut être nulle, définit la *location* du Shoot si l'utilisateur le veut
  - **fkUser**
    Clé étrangère définissant l'Archer qui a tiré le Shoot
  - **finished**
    Propriété qui est définie à « true » par l'API quand le nombre de *Ends* associée avec le Shoot est le même que le nombre de *Ends* mise dans ***nbEnds***

- **Type**
  Représente le Type du Shoot
  - **idType**
    Identifiant du type
  - **name**
    Le nom du Type, dans ce cas il y en a 3 : *Training*, *Indoor* et *Outdoor*
- **Location**
  Permet d'ajouter une localisation au *Shoot*
  - **idLocation**
    Identifiant unique de la localisation (peut-être redondance des données si des utilisateurs tirent au même endroit)
  - **latitude, longitude**
    Description de la localisation, dans le cas de mon application je pensais utiliser ces champs pour *Rue* et *Ville* temporairement
- **End**
  Contient les flèches tirées, appartient au *Shoot*
  - **idEnd**
    Permet l'identification au sein des Arrows, agit uniquement comme une table intermédiaire « Conteneur » Ajoute de la logique à la DB
  - **fkShoot**
    Décrit l'appartenance à un *Shoot*
- **Arrow**
  Décrit la flèche et sa valeur tirée par l'archer
  - idArrow
  - posX, posY, point
    Décrit la position de la flèche sur la cible (hors cadre du TPI mais possibilités d'amélioration), et le point qui est attribué à la flèche (attribut le plus important de la DB)
  - fkEnd
    Détermine sur quel End la flèche va venir se greffer afin de respecter ce principe de séries et de chronologie évoqué plus haut.
- **User**
  Décrit un utilisateur qu'il soit admin ou archer
  - **idUser**

Id unique attribué par Facebook[1] pour l'user utilisant cette application, je m'en sers afin d'identifier les utilisateurs, lors de l'entrée dans la DB l'ID est « *salté* » avec *facebook_*

- **firstname, lastname, email**
  Informations relatives à l'utilisateur, récupérées via Facebook
- **isAdmin**
  Propriété booléenne afin de mettre un utilisateur en « Admin »

- **has_Group, has _Coach, Group**
  Ces tables ne seront pas utilisées dans le cadre de ce TPI mais seront utilisées ensuite afin de définir des rôles bien plus précis aux utilisateurs

3.3 API

3.3.1 Sources

Voir Annexes
https://github.com/mbonjour/Shoot4Stats/blob/master/Documentation/Annexes/API/apiDocumentation.md
Et
https://github.com/mbonjour/Shoot4Stats/blob/master/Documentation/Annexes/API/authDocumentation.md

L'API est bien intégrée dans l'Application, elle est chargée en même temps que le Frontend par ExpressJS.

3.3.2 Structure dossiers

- Documentation
  - Annexes
    - Dans ce dossier se trouve les docs et les avancements fait avant le TPI et les Annexes du rapport
  - Rendu
    - Le dossier rendu pour le dossier final du TPI
- Sources
  - Cli
    - Les tests sur l'API sont présents dans ce dossier
  - Site
    - api
      - dal --> Data Access Layer (Dans ce dossier se trouve les modèles et les méthodes de la DB)
      - middlewares (Dans ce dossier se trouves l'initialisation du module d'authentification et les contrôles effectués)
      - controllers (Chemins différents pour gestion de L'API)
      - config --> config nécessaire pour l'API (connexion DB, ...)
    - client
      - Le dossier désigné pour le build de Vue.js (Ce sera donc majoritairement du code généré automatiquement)

[1] Pour plus de détails : https://developers.facebook.com/docs/graph-api/reference/v2.5/user

3.3.3 Contrôleurs

Les contrôleurs et leurs tâches :

- /api/login
  - GET …/me : récupère les informations de l'utilisateur courant
  - GET …/facebook : route propre à passport permettant de log sur facebook
  - GET …/facebook/return : route propre à passport permettant de log sur facebook
  - GET …/logout : permet de logout l'utilisateur courant
- /api/shoots
  - GET : permet de récupérer la liste des *Shoots*
  - POST : permet d'ajouter un *Shoot*
  - GET …/idShoot : permet de récupérer les détails du *Shoot* voulu
  - GET …/idShoot/finish : permet de terminer un *Shoot*
- /api/ends
  - POST : permet d'ajouter une volée au *Shoot* passé dans l'objet

3.4 Outils et ressources

3.4.1 Vue JS

https://vuejs.org/v2/guide/

VueJS est un Framework qui permet de construire des interfaces utilisateur rapidement et facilement. Grâce à son système de **composants** qui fonctionnent comme des petits modules contenant un mélange de **JavaScript, HTML et CSS**. Voici l'exemple type d'un fichier **.vue** :

```
<template>
  <div class="shootSummary">
    <span>Arrows : {{ shoot.nb_ends*shoot.nb_arrows_by_end }}/{{ shoot.nb_t
    <span class="pushRight">{{ shoot.date }}</span>
  </div>
</template>

<script>
export default {
  name: 'shootSummary',
  props: ['shoot']
}
</script>

<style scoped>
.shootSummary {
  margin: 2px;
}
</style>

```

Figure 4 Exemple fichier .vue

L'on voit ici la partie ***<template> <script> et <style>*** qui regroupe chacun le code HTML, le JS et le CSS.

En créant des modules simplement, Vue nous permet de faire des opérations sur des éléments HTML de base afin d'avoir un site bien dynamique.
Exemple de *binding* sur un élément :

```
<div id="app-2">
  <span v-bind:title="message">
    Hover your mouse over me for a few seconds
    to see my dynamically bound title!
  </span>
</div>
```

Ici le *bind* va récupérer une valeur "message" qui a été déclarée dans la partie ***<script>***

3.4.2 VueX

https://vuex.vuejs.org/en/
Depuis la Doc :
« VueX est un ***state management pattern*** **et une bibliothèque** pour des applications Vue.js. Il sert de store centralisé pour tous les composants dans une application, avec des règles pour s'assurer que l'état ne peut subir des mutations que d'une manière prévisible. »

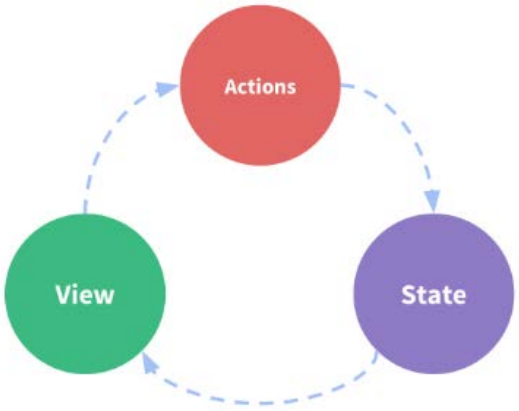


Figure 5 Cercle de vie de VueX

Pour résumer, c'est une « extension » de VueJS qui intègre un système de *store*. En effet, c'est un module directement inspiré de ***flux*** développé par Facebook qui permet d'avoir des états Immutables. Ces états sont ensuite partagés au sein de l'application.

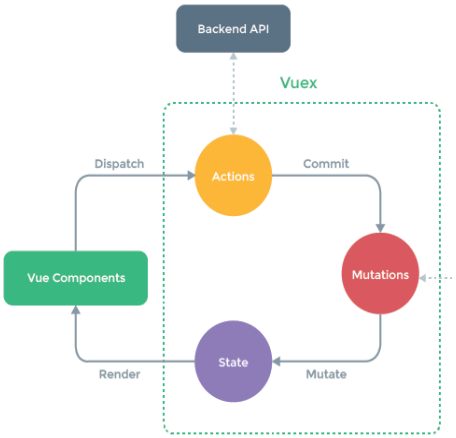


Figure 6 Schéma détaillé VueX

3.4.3 PassportJS

http://passportjs.org/docs

PassportJS est un middleware d'ExpressJS. Un middleware est un petit programme qui va s'effectuer lors du routage, entre la réception de la requête et l'action que cette requête va provoquer. Ici PassportJS va évidemment mettre en place pour nous une stratégie d'authentification toute faite et simple à mettre en place :

```
module.exports = () => {
  passport.use(new FacebookStrategy({
      clientID: config.passportFacebook.clientID,
      clientSecret: config.passportFacebook.clientSecret,
      callbackURL: config.passportFacebook.callback,
      profileFields: ['id', 'first_name', 'last_name', 'photos', 'email']
    },
    function (accessToken, refreshToken, profile, cb) {
      //régler problème si l'user refuse de transmettre son email
      store.repositories.users.getOrCreate(profile, (err, user) => {
        store.repositories.shoots.getLight(user.id, (err, arrayOfIds) => {
          user.shootList = arrayOfIds
          return cb(err, user)
        })
      })
    }))
  passport.serializeUser(function (user, cb) {
    cb(null, user)
  })
  passport.deserializeUser(function (user, cb) {
    cb(null, user)
  })
  return passport
}
```

Figure 8 Exemple d'instanciation de PassportJS

```
router.get('/facebook',(req, res, next) => {
  req.session.url = req.query.url
  next()
}, passport.authenticate('facebook', {scope: ['email']}))

router.get('/facebook/return', passport.authenticate(
  'facebook', {
    failureRedirect: '/#/'
  }
), (req, res) =>{
  res.redirect('/#' + req.session.url || '/#/')
  delete req.session.url
})
```

Figure 7 Exemple des routes à mettre en place

Les routes à mettre en place à droite sont juste 2 routes basiques qui permettent d'authentifier la personne se connectant et de la rediriger sur la page d'où elle est arrivée.

3.5 Historique

Je listerai au mieux les changements effectués, sachant que ce sera une programmation suivant la norme « AGILE » qui consiste à discuter en permanence avec le client et à développer par étapes l'application. Je vais donc répertorier ici les discussions avec le chef de projet / client.

Après une discussion avec le chef de projet l'on a décidé de faire la partie « Archers » le mieux possible (en termes de design et de gestion des données au sein des composants) les parties Administrateurs et anonymes sont donc à prévoir dans une prochaine version de l'application. Ce rapport portera donc sur une seule partie du cahier des charges qui était « Partie Archers ».

Au cours du projet j'ai demandé en permanence l'avis du chef de projet et lui ai proposé quelques possibilités ou alors on les voyait ensemble afin que je puisse avancer selon ces désirs.

4 Dossier de Réalisation

Je vais ici expliquer comment le projet s'est déroulé, les problèmes rencontrés, l'aide que j'ai reçue et comment j'ai conçu l'application.

4.1 Mise en place du projet

Explications : Vue init – Vue CLI
Choix du boiler plate, https://webpack.js.org/concepts/
« Webpack » → c'est un gestionnaire de module et très grossièrement
« « « Compilateur » » » il va transformer le code compréhensible que j'écris avec Vue et le modularise au sein de fichiers JS qui seront servis **uniquement** par un serveur WEB. Ce qui permet de modulariser le code avant qu'il soit servis afin d'avoir des chargements plus rapides…

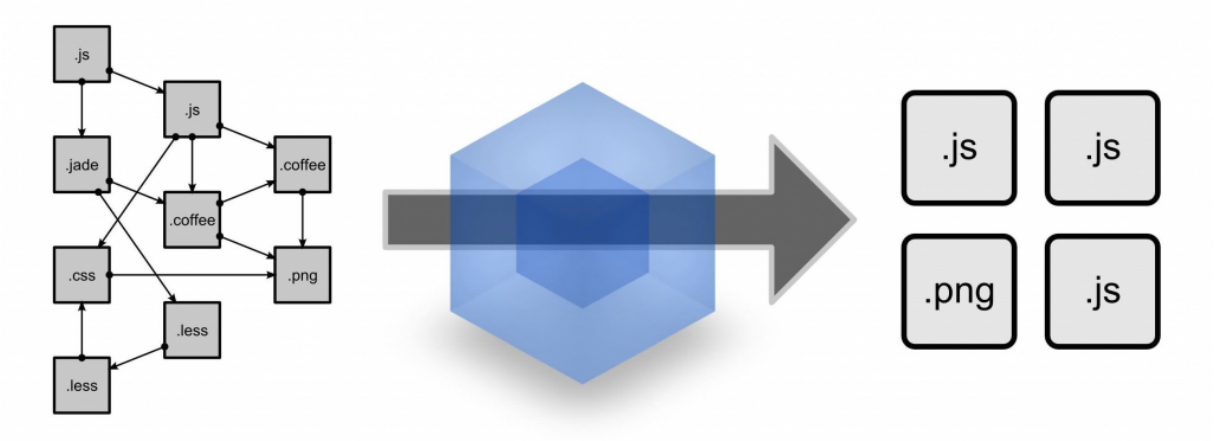


Figure 9 Schéma fonctionnement de Webpack

4.2 Organisation des tâches pour la réalisation

J'ai organisé en accord avec le client les tâches de la manière suivante :

- Mettre en place les Vues pour les archers authentifiés
- Feedback avec les testeurs et le client principal (le chef de projet a bien a été clair sur un point : faire de la partie *Archers* la partie principale et la plus fonctionnelle)
- (Mettre en place la partie des administrateurs) → Suite aux demandes du client cette partie est optionnelle

Cette organisation a été vue avec le chef de projet et discutée. Évidemment c'est la répartition idéale des tâches.

4.3 Menu et thème

4.3.1 Menu dynamique

Le menu est dynamique en fonction des accès que l'on a afin d'éviter de se retrouver sur des pages sans y avoir accès.

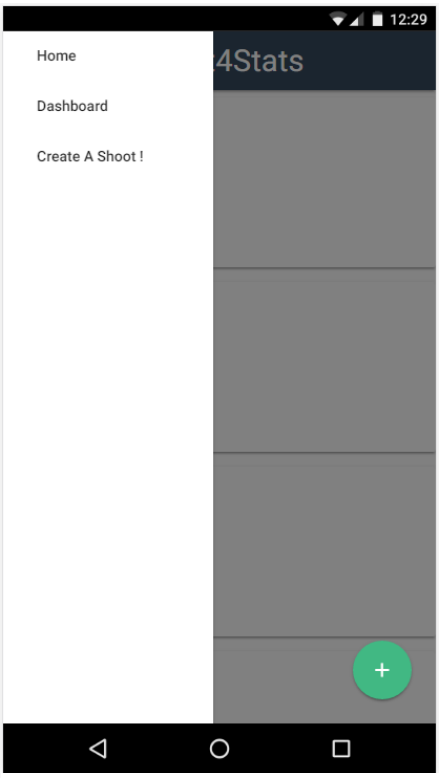


Figure 10 Menu en "Archers" v1

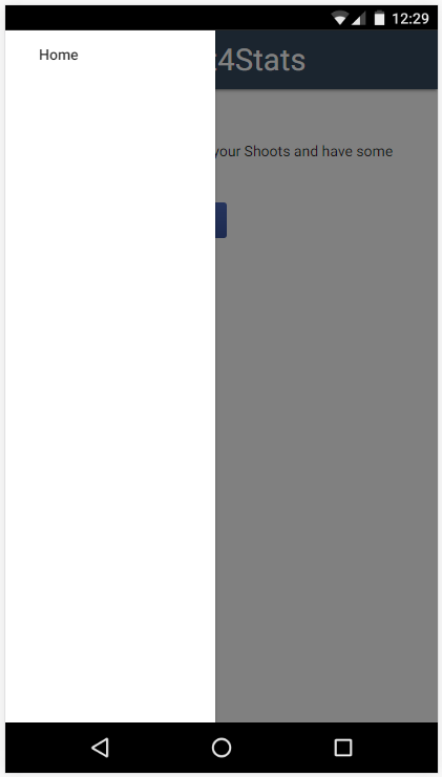


Figure 11 Menu en Anonyme

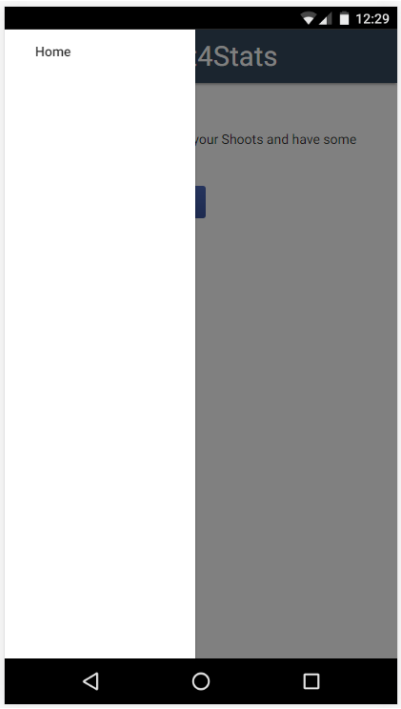


Figure 13 Menu en "Archers" v2

Ici le menu a été simplifié, le lien vers Home ramène sur la Dashboard de l'utilisateur s'il est connecté. Ainsi l'utilisateur n'est pas surchargé de liens.

Figure 12 Menu en Administrateur

4.3.2 Barre de Navigation

La barre de navigation prend 2 formes différentes :

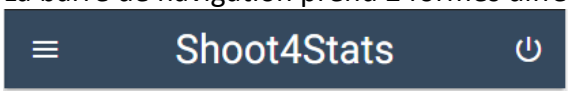


Figure 15 Barre de navigation "Authentifiés mobile"

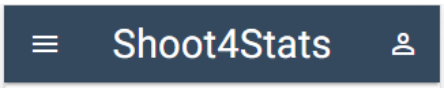


Figure 14 Barre de navigation lorsque l'on n'est pas authentifié

Nous pouvons voir ci-dessus *Figure 15* la barre de navigation qui s'affiche lorsque **l'on est authentifié**.
Et à droite avec la possibilité de « log out », à gauche le menu qui a été présenté plus tôt dans le rapport (cf. 4.3.1 ci-dessus).
Si **l'on n'est pas authentifié** l'icône n'apparaît simplement pas (*Figure 14*)

Figure 16 Barre de navigation en desktop

Le menu en version Desktop prends toute la largeur de l'écran.

4.3.3 Thème

Explication

Le choix des couleurs s'est fait en collaboration avec le chef de projet, nous avons donc décidé de partir sur une palette de couleur qui est celle du symbole de VueJS.
Ce qui donne un vert clair (#41B883) et un bleu marine foncé (#35495E)
Voici le logo de VueJS :

Figure 17 Logo de VueJS

Exemple

Voici les couleurs au sein d'un bouton p.ex. :

Figure 18 Exemple de bouton stylisé

V2

Ce que je vais appeler la V2 dans ce rapport, c'est en fait une révision complète du design de l'application pour que ce soit bien plus épuré et plus ergonomique → Le client a insisté là-dessus c'était donc un objectif crucial de ce travail.
Cette gestion du frontend m'a pris beaucoup de temps et j'ai reçu beaucoup d'aide du chef de projet sur **la gestion de l'espace**, mais aussi sur **la gestion des données** dans l'application qui a elle aussi été revue.

Boutons

Cela paraît anodin mais les boutons sont positionnés de la même manière tout au long du projet, en effet ils sont fixés en bas de l'écran :

- Ceci afin de permettre de bien les voir car ils prennent aussi tout la largeur
- Et de les atteindre très facilement du pouce sur la version mobile de l'application

4.3.4 Framework CSS

Finalement les principaux problèmes rencontrés viennent du Framework CSS Materialize (http://materializecss.com/) qui m'a embêté tout au long du projet.

Recherche

J'ai pris un moment afin de chercher le bon Framework pour me simplifier la vie côté CSS, j'ai penché pour Materialize car j'ai vu qu'il gérait les versions mobiles plutôt bien, qu'il avait des boutons bien stylisés et des formulaires plutôt sympas en apparence. De plus je m'étais mis d'accord avec le client pour utiliser un Framework qui respectait le Material Design.

Problèmes rencontrés

Le premier problème de ce Framework réside dans le fait qu'il utilise le JQuery en permanence pour initialiser ces composants ce qui ne me facilitait pas vraiment la tâche et ne plaisait pas non plus tellement au client. J'ai finalement fait fonctionner cela en regardant comment intégrer correctement le JQuery dans mes fichiers Vue.
De plus j'ai eu des petits soucis avec certains composants. Au final cela se révélait surtout par des oublis d'intégration dans Vue du code JQuery. Qui'a fait perdre un certain temps, c'est d'ailleurs pour cela que je n'utilise Materialize dans mon projet « que » pour la navbar, le bouton de création de Shoot, le squelette des cartes et le formulaire de création de Shoot.

4.3.5 Authentification

La majorité du travail d'authentification a été faite avant le projet en collaboration, j'ai tout de même dû revenir sur l'API à plusieurs reprises afin de fixer un bug sur l'authentification, une requête qui se fait à chaque changement de routes qui ne prenait pas les paramètres en chiffres mais en *string*, j'ai dû faire un *parse* (string > INT) correctement.

L'authentification au sein du Frontend grâce à une méthode dans VueRouter qui se nomme *beforeEach* qui sera lancée avant le rouage entre les différentes pages de l'appli.

4.4 Partie *Archers*

4.4.1 Maquettes

Les maquettes ont été faites avant de débuter les Vues et elles sont présentes en **Annexes**, elles m'ont bien aidées à la représentation des composants que j'allais devoir mettre en place. Ainsi qu'à la représentation dont les données seront gérées.
Malgré tout j'ai reçu de l'aide de mon chef de projet, en effet quand je lui ai montré les débuts « V1 » de l'application il n'était pas convaincu de la gestion des données que j'effectuais car comme vous le verrez dans les interactions avec l'API **ci-dessous** j'effectuais des appels pour chaque composant alors que j'avais mis en place le système VueX (*3.4.2 ci-dessus*) qui permet une gestion des données plus agréable pour ce genre de projet.

4.4.2 Vue « Home »

Description

Cette Vue sera le point d'entrée de l'application WEB, elle permettra à tout à chacun de s'authentifier (via Facebook, l'on récupère ainsi les emails, le nom et le prénom avec plus de facilité et plus d'ergonomie)

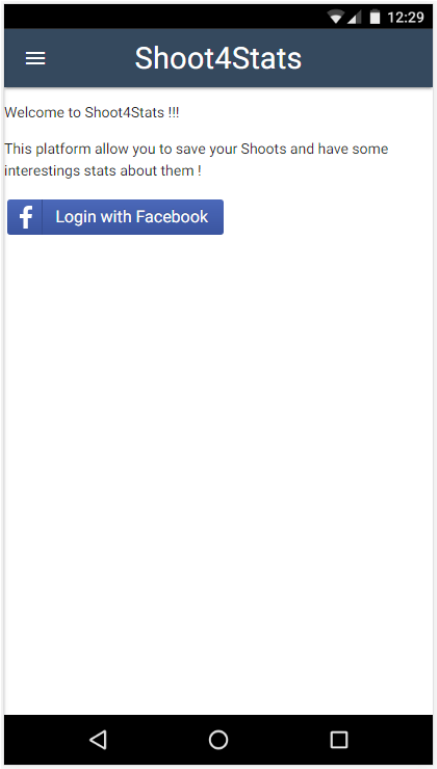


Figure 20 Vue Home v1

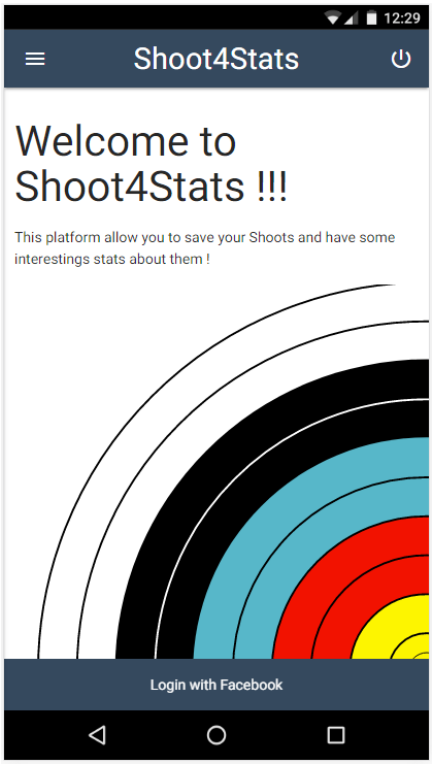


Figure 19 Vue Home v2

L'authentification est gérée en BackEnd et a été développée en collaboration avec le chef de projet, elle se base sur un middleware pour Node.JS qui se nomme **PassportJS.** Le système d'authentification est expliqué dans la Doc de l'API (cf. 3.3)

Logique

La vue « Home » n'a pas de logique, c'est une page HTML conventionnelle.

4.4.3 Vue « Dashboard » → *Home* lorsque l'on est authentifié

Description v1

La vue Dashboard a pu être discutée avec le client.
Le client m'a fait parvenir ses préférences (qui ne sont pas si différentes du CDC) du fait d'avoir un historique de Shoots et d'en avoir un bref aperçu rapidement et simplement. Puis avoir à l'aide d'un clic des détails précis concernant le Shoot sélectionné ou d'en reprendre un s'il n'est pas terminé.

Composant Dashboard

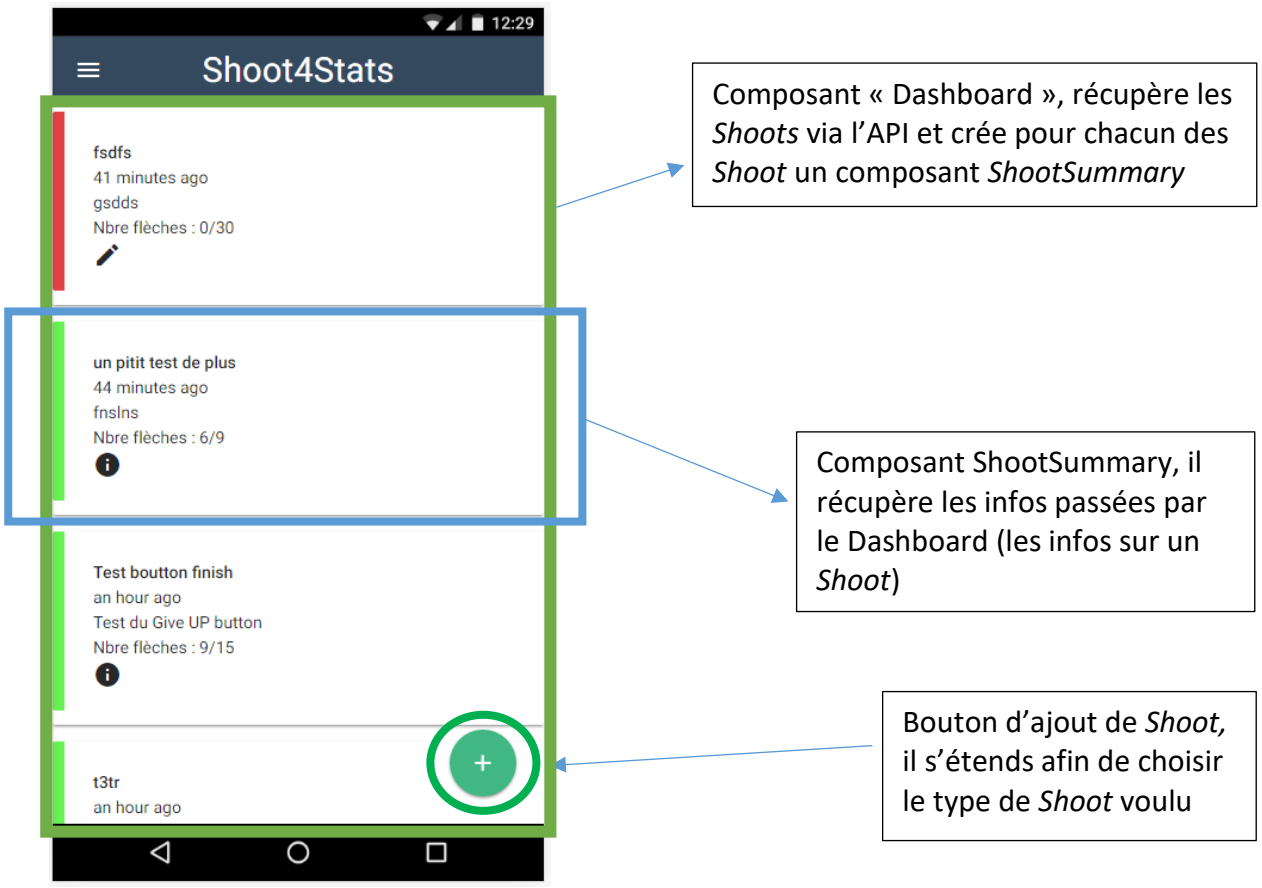
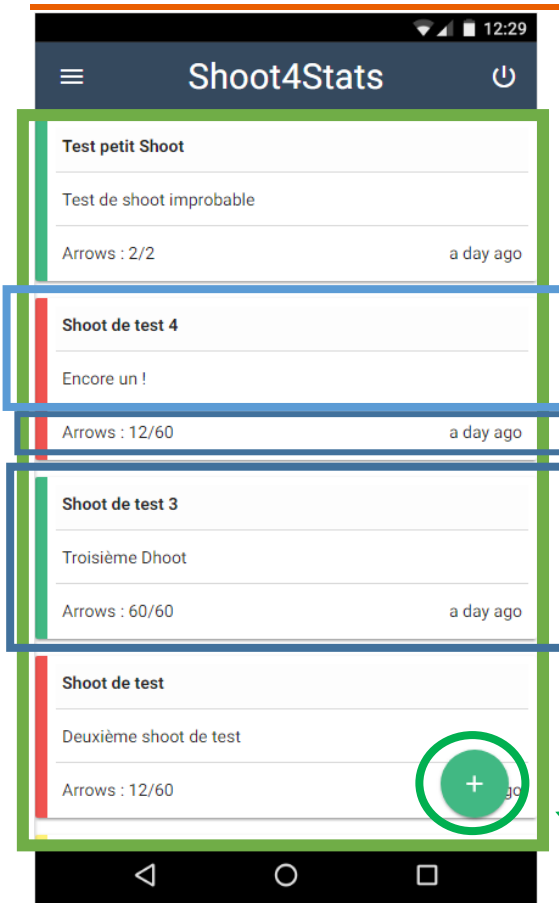


Figure 21 Vue Dashboard décomposée v1

Description v2

La « V2 » de la vue Dashboard est assez différente, nous avons décidé d'enlever les icônes afin de permettre à l'utilisateur de toucher n'importe où sur la carte pour voir les détails. En soi la structure du Dashboard a beaucoup changé, la structure des composants internes ont également changés comme on le verra au long du chapitre **en continuant avec la terminaison « v1 » et « v2 »**



Composant « *Dashboard* », récupère Shoots grâce à VueX et crée pour chacun un composant *ShootItem*

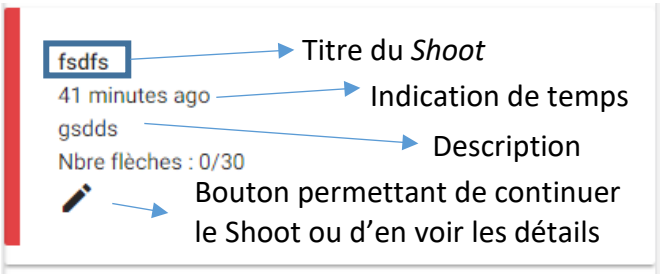
Composant *ShootItem*, il récupère les infos sur VueX

Composant *ShootSummary* qui affiche des infos très brèves sur le *Shoot*

Dans le cas d'un clic sur un Shoot terminé (vert ou jaune) le composant *ShootDetails* est appelé

Bouton d'ajout de *Shoot* il s'étends afin de choisir le type de *Shoot* voulu

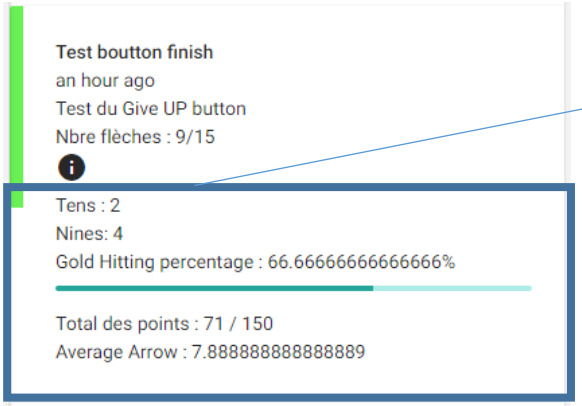
Composant ShootSummary v1



La bande rouge indique que le Shoot n'est pas terminé, à l'inverse du vert

Figure 22 Composant ShootSummary v1

Composant ShootDetails v1



Lors du clic sur cette icône le composant ShootDetails est affiché

Informations à propos du Shoot plus poussées (p.ex. les points moyens, le pourcentage de flèche dans le Jaune, …)

Figure 23 Composant ShootSummary et affichage ShootDetails

Composant ShootItem (v2)

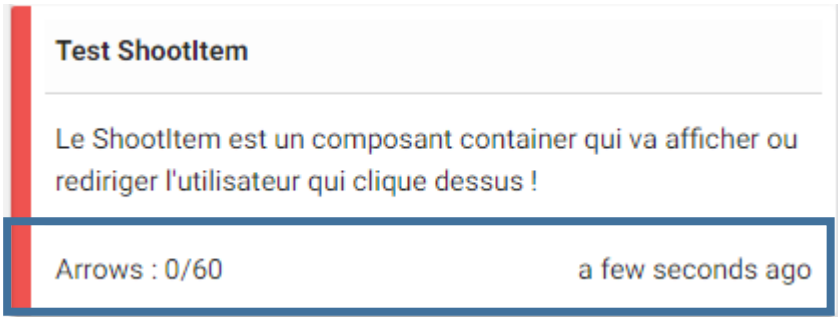
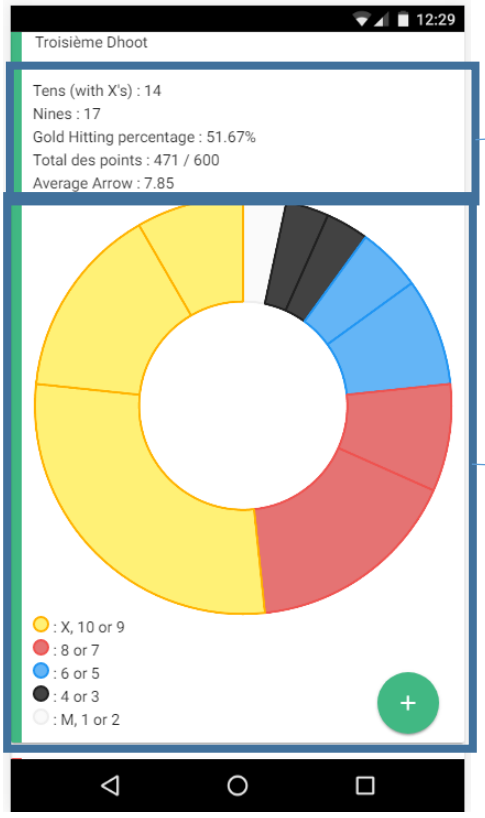


Figure 24 Composant container ShootItem

Le composant ShootItem contient le titre et la description mais aussi le composant ShootSummary dans son état non actif.
Mais si l'on clique dessus (le composant, soit la carte) on active la version détails du Shoot et l'on voit ainsi le composant ShootDetails prendre la place du composant ShootSummary encadré ci-dessus.

Composant ShootDetails v2

Le composant shootDetails intègre des chiffres à 2 décimales (pour plus de lisibilité évidemment) et d'un graphique affichant le nombre de flèches / points (5 flèches dans le 10, 6 flèches dans le 9, …).



Partie chiffres, ici je mets les chiffres que l'API calcul et me renvoi, cela donnes des indications précises sur le Shoot, ces informations sont des informations qui sortent souvent dans les compétitions internationales

Partie graphique grâce à Chart.JS, difficile à mettre en place mais je trouve que ça vaut le coup, ça donne une bonne illustration sur les points obtenus.

Logique – Interaction API Schémas v1

La Vue Dashboard va aller récupérer les shoots de l'utilisateur grâce à l'API (/api/shoots) puis elle va boucler sur la liste renvoyée en passant en propriétés d'un composant partagés[2] *shootSummary* un *Shoot.*

[2] Les composants partagés sont des composants qui seront utilisés à plusieurs reprises, ils seront donc mis au même endroit afin d'en faciliter l'accès.

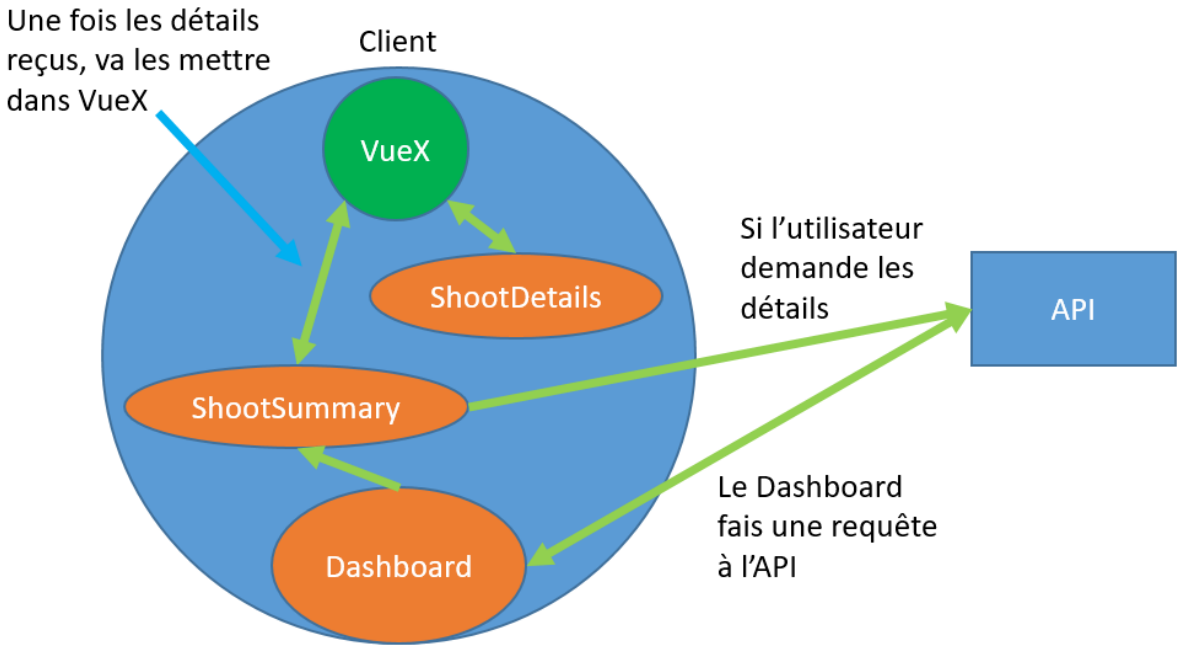


Figure 25 Interaction avec API v1

Le composant *shootSummary* appelle à son tour l'API si l'utilisateur désire voir les détails de son Shoot. Il va mettre ces détails dans un state de VueX *currentShoot* il va afficher un composant partagés[2] *shootDetails* qui prend les informations du state VueX et va afficher ses informations supplémentaires de manière simple.

Logique – Interaction API Schémas v2

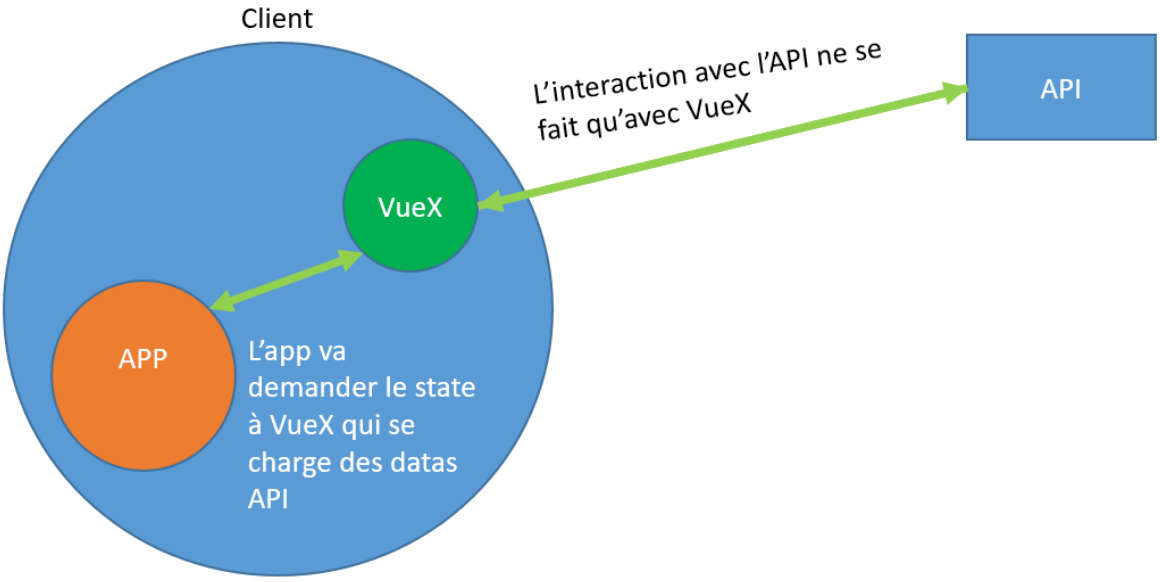


Figure 26 Interaction avec VueX

L'interaction avec l'API ne se fait que par VueX ainsi l'on sait que si l'on a besoin de quelques informations au sein du frontend on doit demander à VueX (de plus c'est réactif[3]).
L'on voit si l'on compare les schémas que la v2 est plus simple, plus pratique et plus logique.

[3] Réactifs c'est à dire que si une valeur change dans VueX dû à un appel à VueX d'un autre composant la valeur va changer en direct partout dans l'APP

Problèmes rencontrés

J'ai rencontré un problème lors de l'intégration de Chart.JS dans cette Vue.
En effet, lorsque j'appuyais sur le *Shoot*, une fois sur deux le graphique ne s'affichait pas.
Ceci est dû à la manière de VueJS de rendre ces composants, j'ai donc dû lui indiquer qu'il devait rendre le graphique lors de l'appel du composant après avoir reçu les informations nécessaires.

```
6  export default Doughnut.extend({
7    mixins: [mixins.reactiveProp],
8    props: ['chartData', 'options'],
9    mounted () {
10     this.$nextTick(() => {
11       this.renderChart(this.chartData, this.options)
12     })
13   }
14 })
15
```

Figure 27 Utilisation du nextTick()

J'ai utilisé Materialize au début pour faire mes cartes et j'utilisais des images fixes de couleur verte ou rouge pour afficher l'état du *Shoot.* Ce qui ne prenait pas toute la hauteur de la carte lorsque j'affichais les ShootDetails (comme vous pouvez le voir Figure 23) alors lors de la refonte j'ai décidé d'utiliser que le squelette des cartes Materialize, et j'ai changé la bordure gauche des cartes pour afficher l'état, tout bête mais il fallait y penser.

4.4.4 Vue « Create Shoot »

Description

Cette vue va permettre à un utilisateur de débuter un Shoot. Sa route (routage des pages en frontend → VueRouter) prend un paramètre, car le bouton « + » du Dashboard peut permettre à l'utilisateur de définir quel type de Shoot il veut débuter afin de d'accroître la facilité d'utilisation de l'application.

ScreenShots

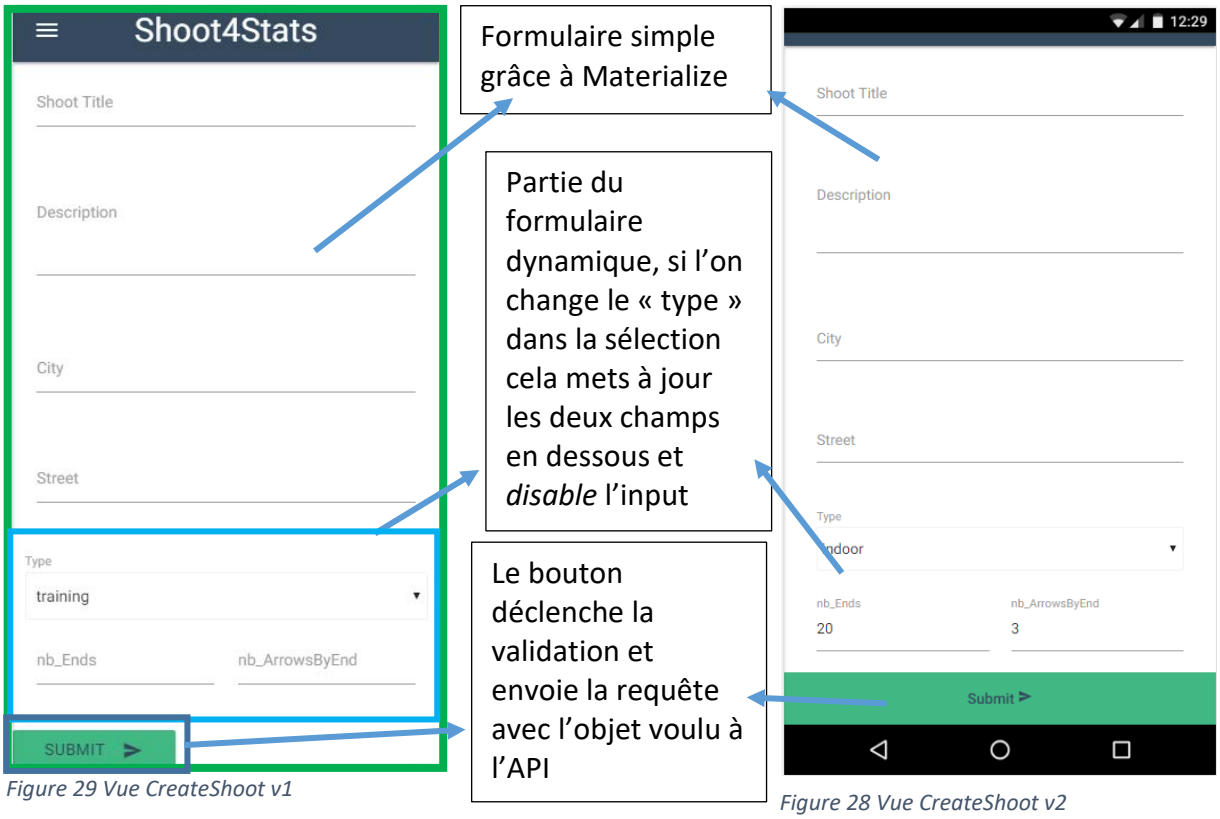


Figure 29 Vue CreateShoot v1

Figure 28 Vue CreateShoot v2

Logique

Cette Vue est un simple formulaire, je désactive l'envoi des données et « toast » (fenêtre modale) si la validation n'est pas ok, donc tant que tous les champs ne sont pas remplis ou que le format n'est pas respecté les données ne sont pas envoyées. Il ne fait par contre pas de POST comme un formulaire HTML, je récupère les données dans les balises « data » de mon composant et fait un POST uniquement lorsque j'ai vérifié les données.

Interaction API – Schémas

Ici pour la **V2** l'on passe par VueX et **V1** direct par l'API. Je mets le point sur les 2 versions afin de bien signifier qu'un travail sur les données a été effectué.

Problème rencontrés

Mon principal problème ici a été de mettre en place la partie dynamique du formulaire où j'ai passé un temps fou afin de mettre les valeurs correctes dans le v-model.
J'ai finalement réussi grâce au JQuery pour que les champs *s'updatent* automatiquement, tout ça parce que j'avais zappé un paragraphe dans la documentation de Materialize concernant l'initialisation des listes de sélections.
Pareil pour l'envoi des données, j'ai perdu un temps fou avant de me rendre compte que je n'envoyais juste pas à un objet correct à l'Api…

4.4.5 Vue « Edit Shoot »

Description v1

Cette Vue n'est pas accessible par le menu car elle a besoin d'avoir l'ID d'un Shoot pour fonctionner, de plus cette vue ne sert qu'à continuer un Shoot depuis le Dashboard ou commencer un Shoot lorsqu'on vient de le créer.
Cette Vue permet d'ajouter des *Arrows* et *Ends* au *Shoot* courant.

ScreenShots – Schémas v1

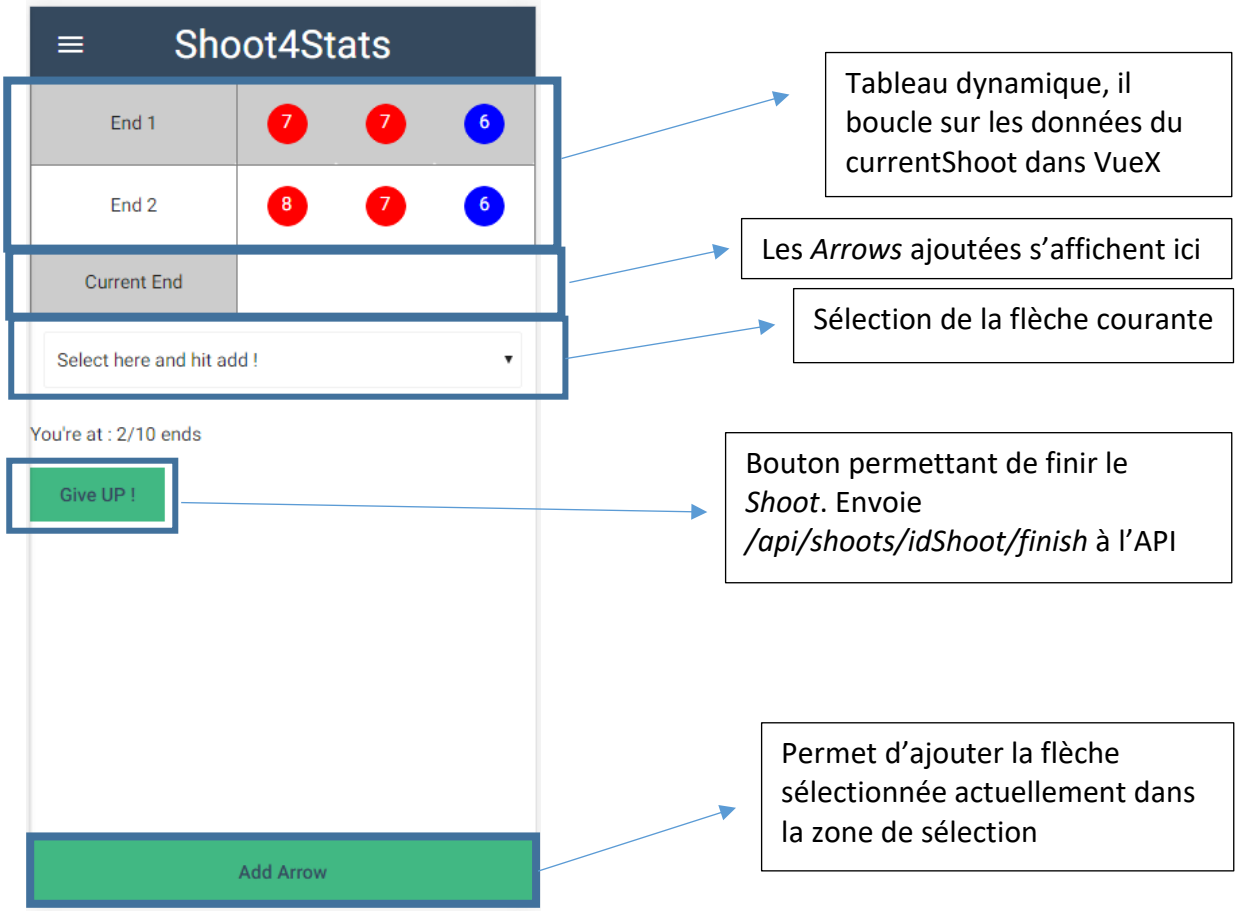


Figure 30 Vue editShoot v1

Composant arrowItem

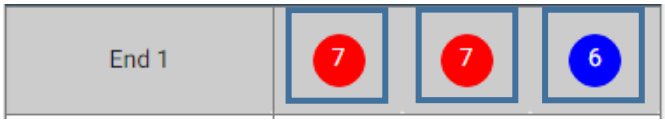


Figure 31 Composant arrowItem

Le composant arrowItem prend comme propriétés (paramètre) un nombre qui représente le point de la flèche (0 – 11).

Alors pour en revenir à notre composant, il prend la valeur du point et affiche une pastille de couleur afin que l'archer puisse facilement voir dans la globalité ses points. Le 0 est interprété par un « M » et le 11 par un « X ».

Problèmes rencontrés

En développant le composant je me suis rapidement aperçu que lors de la création de l'API je n'avais pas pris en compte les « M » et les « X » car le champ *point* de la DB n'accepte que des entiers.

J'ai alors modifié l'API pour que lors des calculs il transforme le « X » en 10 pour ne pas fausser les calculs mais qu'il envoie quand même « 11 » au Frontend afin que je puisse faire la différenciation entre le 10 et le X.

Cette partie de l'application est celle qui m'a prise le plus de temps, c'est aussi ici que l'utilisateur aussi va passer le plus de temps, il faut donc que l'interface soit explicite et qu'elle soit ergonomique.

Aide reçue

J'ai reçu de l'aide et discuter avec mon chef de projet pour la mise en place des pastilles pour plus d'ergonomie. Et en discutant ensemble on a pu choisir de jolies couleurs pastel qui n'agressaient plus l'œil.
Il m'a aussi demandé d'implémenter une fonction de « toaster » afin d'afficher les erreurs à l'utilisateur rapidement et simplement.
Ces 2 solutions ont étés mises en place par moi-même ensuite mais il m'a donné des pistes à la complétion de ces tâches. Vous pouvez le voir dans la **V2** ci-dessous.

Il m'a aussi aidé à m'imaginer la structure des données pour ces composants, c'est grâce à lui que je passe par VueX pour faire mes requêtes vers l'API qui permet alors d'avoir une meilleure modularisation du code par la suite.

Description V2

La vue a été repensée au long du projet, comme on le voit dans la v1 le bouton Give Up ressemble bien trop au bouton d'ajout et il est à trop courte portée de la liste de sélection ce qui induisait en erreur les utilisateurs. On a donc décidé de mettre un bouton *Give UP* plus discret et moins atteignable.

Et les pastilles… Les pastilles dans la v1 attaquent l'œil par leurs couleurs bien trop vives, je me suis donc inspiré des couleurs pastel de Material design pour mettre en forme un tableau plus épuré et plus visible…

Puis quant à faire les choses le mieux possible pour cette partie Archers j'ai créé un composant sélecteur fait maison qui reprend les pastilles faites grâce au composant *arrowItem*.
C'est dans ces cas-là que l'on voit la puissance de VueJS s'il est bien utilisé, vu que la modularité de mon composant *arrowItem* a été bien faite j'ai pu le réutiliser sans problèmes.
Le sélecteur est un composant fait maison mais on a tout de même la possibilité d'y mettre un v-model (récupération des données réactives) grâce à un événement lancé lors des changements de valeurs (ici le clic sur une case du tableau) et en retournant une propriété « data » au parent.

ScreenShots

Figure 32 Vue editShoot v2

Interaction avec l'API

L'interaction avec l'API est gérée une fois de plus par VueX et j'ai reçu l'aide du chef de projet afin de mettre en place la méthode pour ajouter les volées. Afin de mettre en place une *optimistic UI,* le state va changer et le tableau va afficher la volée qui vient d'être ajoutée sans même avoir fait la requête d'ajout à l'API. L'utilisateur va voir sa volée disparaître et une erreur *toastée* si la requête vers l'API n'a pas fonctionné.
Si la requête fonctionne on *toast* l'utilisateur un succès. Et après le succès on réinitialise le state avec la valeur reçue de la requête pour s'assure que c'est à jour et que le state soit propre.

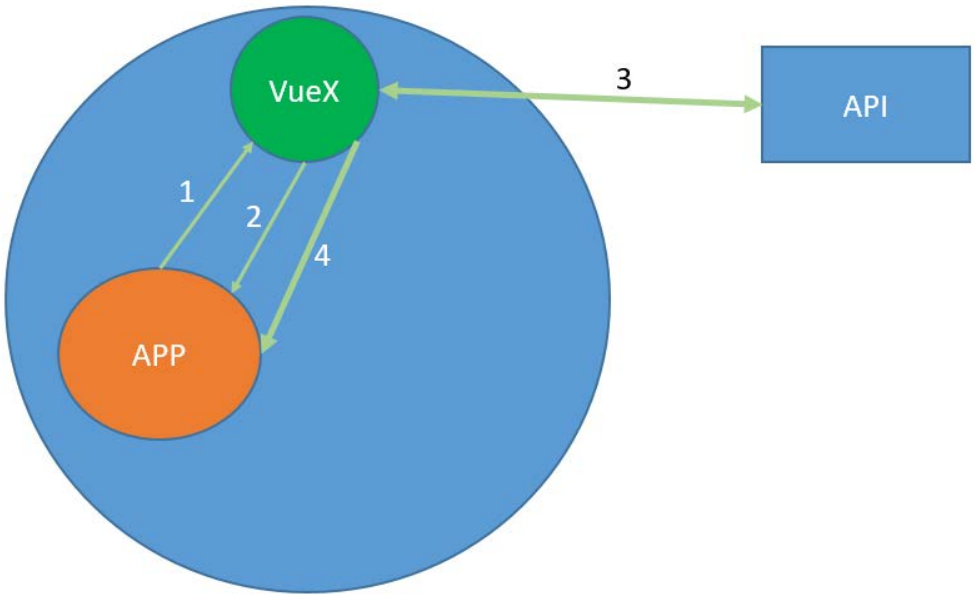


Figure 33 Exemple fonctionnement Optimistic UI

L'avantage de cette méthode c'est que l'utilisateur n'est pas interrompu dans sa complétion ou l'application n'est pas mise en pause durant l'ajout de volée car l'action s'effectue en arrière-plan.
Ça nous donne une application réactive et plutôt bien gérée.

Problèmes rencontrés

Lors d'une volée de 12 flèches (définit comme le maximum avec le client) le tableau prenait trop de place en version mobile :

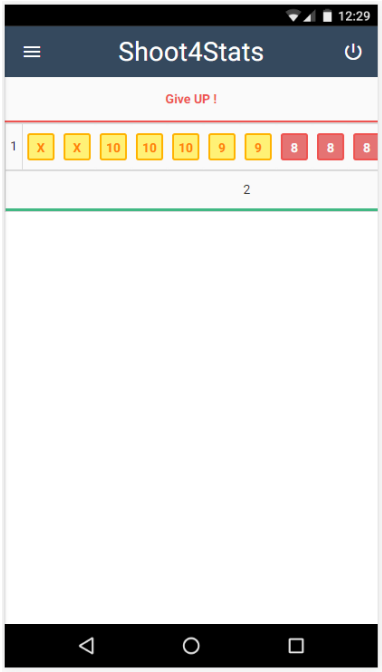


Figure 34 Exemple du bug trop d'Arrows

Résolution

Cette résolution m'a prise aussi énormément de temps à mettre en place mais au final c'est plus une fonctionnalité à laquelle je n'avais pas pensé au départ.
Elle permet de mettre 2 lignes de points dans le tableau si le nombre de flèches / volée est plus grand que 6. Le résultat pour 12 flèches par exemple :

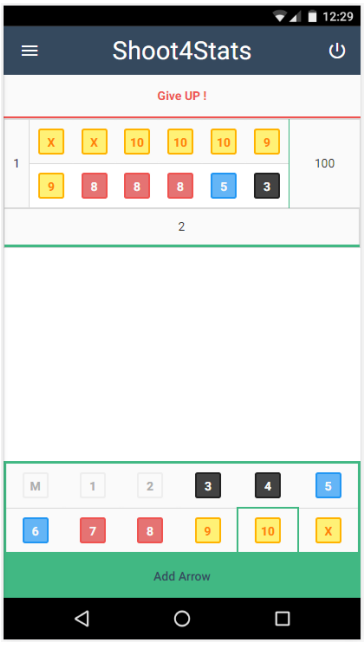
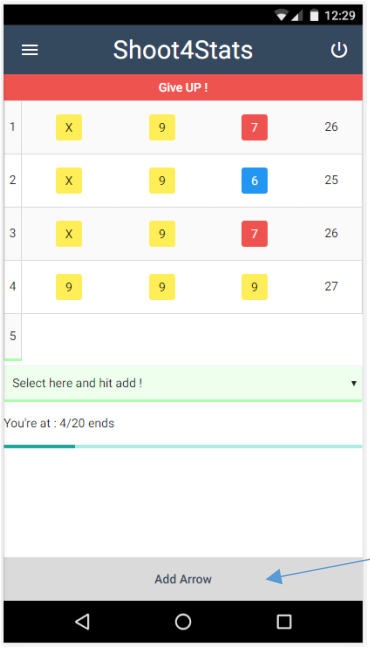


Figure 35 Exemple de lignes multiple

4.5 Description des tests effectués

4.5.1 Test sur editShoot

Le test a été effectué dans une version mise en production au cours du projet et hébergée chez moi afin de permettre aux testeurs externes de faire des tests.
Le test a été d'inclure des valeurs nulles et/ou inexactes. Ce test a été effectué par mon chef de projet et m'a donc averti de cette erreur, nous avons donc dû renforcé la faible validation qui était en place du côté de l'API puis éviter au moins de pouvoir mettre des valeurs nulles en désactivant le bouton lorsqu'aucun point n'était sélectionné :



L'on peut voir ici que le bouton est grisé et si l'on essaie de cliquer dessus rien ne se passera !

Figure 36 Exemple conditions boutons

4.5.2 Test sur CreateShoot

Problème

Là aussi c'est grâce à ma mise en production au cours du projet (de la v1) que j'ai pu avoir un retour de mon collègue Nicolas Crausaz, il m'a fait remarquer que lors de la création d'un Shoot il était possible de mettre un nombre d'Ends négatifs et de même pour les flèches dans les Ends.

Résolution

J'ai donc dû mettre une validation pour cela et toaster les erreurs correctement. Pour cela j'ai utilisé non seulement le champ HTML input type *number* mais aussi une validation sur la condition d'affichage du bouton, tant que les conditions ne sont pas respectées le bouton est désactivé.

4.5.3 Bouton Give UP

Problème

Le bouton Give Up s'affichait même si aucune volée n'avait été entrée ce qui n'a pas de sens et qui pose problème lors de l'affichage des ShootDetails, rien ne s'affichait !

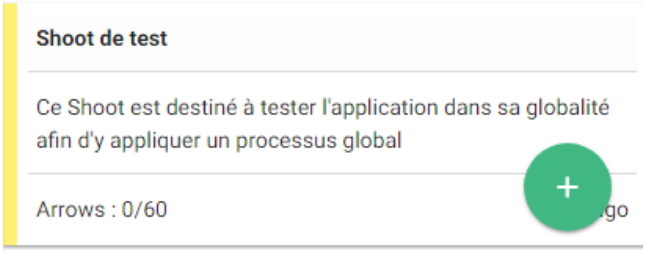


Figure 37 Exemple de Shoot sans Arrows

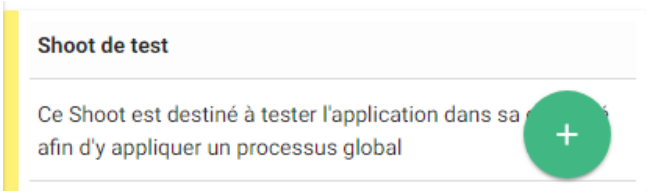


Figure 38 Exemple si l'on essaie d'afficher les ShootDetails

Résolution

J'ai donc modifié la condition d'affichage du bouton afin qu'il n'apparaisse que si le Shoot a été rempli d'au moins une volée

4.5.4 Tests sur formulaire *Create Shoot*

Tests des entrées

J'ai testé que mes conditions soient respectées en essayant tout simplement de remplir le formulaire de nombre, de rien ou de lettres dans les champs et vérifier que le bouton s'affiche au bons moments.

Résultats

Les résultats durant le tests ont révélés des erreurs (possible de mettre 0, maximum trop grand dans le nombre de flèches, nombres négatifs) ce qui m'a permis au fur et à mesure d'affiner ma validation pour ce formulaire.

4.6 Erreurs restantes

Le but du projet (et étant donné que je n'ai fait que la partie Archers) était qu'il reste le moins d'erreurs possible et dans l'idéal aucune (on en rêve tous évidemment). Cependant, il reste encore une possibilité d'amélioration :

- Lors de la création d'un *Shoot* quelqu'un qui s'y connaît peut forcer l'entrée de champs vides et comme la vérification du côté de l'API n'est pas faite le *Shoot* est tout de même crée sans *Titre* ou sans *Description* par exemple.
- Lorsque l'on se logue la première fois après un redémarrage du serveur WEB le serveur ne récupère pas la session de l'utilisateur et le client arrive sur une page blanche après s'être loguer, à voir si ce n'est pas à cause de la mise en production assez petite mise en place (voir chapitre 5 ci-dessous).
- Un utilisateur m'a rapporté une erreur lorsque l'on met des *Shoots* de 1 ou 2 flèches mais je n'ai pas pu reproduire les erreurs alors je ne sais pas si elle a réapparu depuis ou non…

5 Mise en service

Explication de la petite mise en production (fait maison) pour les tests

5.1 Rapport de mise en service

La mise en service afin d'effectuer les tests a bien fonctionné et j'ai ainsi pu administrer ça depuis chez moi.

5.1.1 Explication

J'ai fait la mise en place (comme expliquée en **Annexes**) en local sur mon poste privé chez moi. J'ai ensuite fait appel à un service de DNS Dynamique, récupère grâce à un client installé sur la machine l'IP publique du Routeur en permanence.

J'ai ensuite configuré mon routeur pour ouvrir le port 80 (HTTP) de celui-ci sur le port 3000 de ma machine au sein de mon réseau local. J'ai ainsi dû configurer mon application Facebook pour que le callback soit sur mbonjour.ddns.net

5.1.2 Utilité

Cette mise en service est évidemment temporaire mais elle m'a permis à faire tester des personnes externes au projet mon application, ce qui leur a plu, ils ont même, pour certains, commencer à l'utiliser malgré qu'elle soit encore en cours de développement.

Malgré ça et comme le client a insisté sur la partie Archers mes amis Archers sont plutôt satisfait et me donne des idées pour la suite et suivent bien le projet.

5.2 Liste des documents fournis

La liste des documents qui serait fournis au client serait notamment :

- Ce rapport
- La documentation de mise en production
- Le Manuel d'utilisation (en **Annexes**)
- Les sources du projet

6 Conclusions

Les conclusions, les améliorations possibles et le ressenti du projet

6.1 Objectifs

6.1.1 Atteints

L'objectif fixé par le client en cours de projet de faire la partie Archers le plus propre possible est atteint, j'ai d'ailleurs eu des bons retours et peu de bugs remarqués. Cela me rends confiant sur la partie Archers et me fait dire que la partie Archers et bien faite et complète.

Je me suis familiarisé à VueJS, VueX, VueRouter et pourrait continuer le projet sans problèmes. Cela me prouve un enrichissement personnel, un objectif que je m'étais fixé au début du projet. J'ai fait appel à des librairies que je n'avais jamais utilisées et je me suis malgré tout adapté.

Au niveau de la modularisation je pense avoir fourni un bon travail, j'ai fait au mieux de ce que j'imaginais en faisant des composants assez généraux auxquels l'on passe des propriétés simples et génériques afin de l'utiliser n'importe où. On verra si cela se confirme en continuant le projet plus en détails.

Utilisant un linter (qui me permet d'avoir un code propre selon les standards sans quoi il n'accepte pas de compiler) je pense que l'objectif d'utilisation de la norme « standardsJS » est atteint.

6.1.2 Non atteints

Évidemment je n'ai pas atteint les objectifs de la mise en place des Vues Anonymes et Administrateurs pour me consacrer à 100% dans la Partie dédiée aux Archers.

J'ai discuté beaucoup avec le chef de projet tout au long du projet, il m'a beaucoup aidé surtout. Et je pense donc que la méthode AGILE est à améliorer de mon côté car je ne suis peut-être pas allé vers lui assez (il est vrai que c'est dur en 3 semaines de se rendre compte comment bien appliquer la méthode AGILE).

Ma gestion du temps, il est vrai que l'on a peu de temps lors de son projet, je ne m'en rendais pas plus compte que ça et j'ai été un peu effrayé en fin de projet. Finalement tout s'est bien déroulé.

6.2 Difficultés

6.2.1 Connaissances techniques

J'ai eu des difficultés au niveau de la gestion des données au sein de l'application notamment par l'utilisation de VueX, une technologie que je connaissais à peine. En lisant et en me formant au cours du projet (ce qui m'a pris du temps aussi) j'ai pu aller de l'avant et utiliser de plus en plus cet outil qui s'est révélé très efficace pour la gestion des données. J'ai mis en place un outil VueX plus complet sous les conseils de mon chef de projet.

6.2.2 Créativité

Il est vrai que j'ai eu de la peine à bien m'imaginer ce que les pages allaient donner une fois que c'était fait comme ci ou comme ça. Je faisais quelques propositions au client afin de voir quel design je pouvais faire sur l'appli et j'ai reçu ainsi pas mal de ses propositions. Comme par exemples sur la forme des pastilles, les boutons assez *flats* qui prennent la largeur de l'écran. Ce sont des propositions simples qui ont plu au client mais j'ai eu de la peine à me les imaginer et le chef de projet m'a aidé à imaginer un *flat* design sympa.

6.3 Suite du projet

6.3.1 Mettre en place des rôles

Évidemment il faudrait que l'application puisse avoir un système de rôles, comme ça les coaches pourraient visionner les Shoots de leur Archers et même les exporter peut-être afin d'en avoir une sauvegarde ailleurs.
Ce système de rôle est déjà plus ou moins mis en place dans la base de données, il n'a pas été utilisé car l'authentification n'avait pas été mise en place correctement pour les gérer et que l'API n'avait pas non plus de requêtes afin d'administrer ces rôles.

6.3.2 Publication

La publication de la **V2** récemment sur ma mise en production m'a mise en confiance dans le sens où les quelques personnes qui l'utilisent sont assez satisfaites de ce qu'il y a actuellement et elles voient que c'est un projet qui bouge et qui continue d'avancer.

6.3.3 Sélecteur de flèches

À terme l'idée serait de mettre à la place de ce tableau sélecteur de flèches :

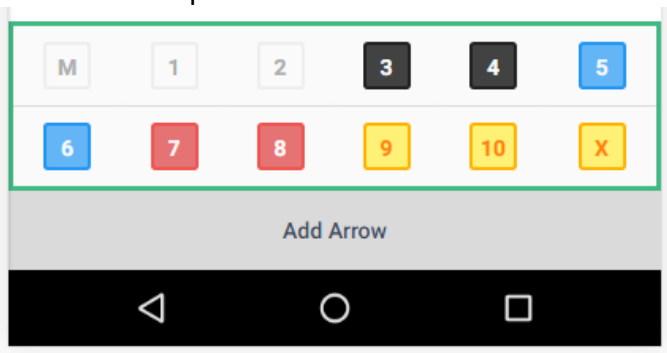


Figure 39 theSelector

Une cible sur la page et que l'on puisse ainsi montrer l'endroit d'impact des flèches, ce qui est vraiment utile pour un Archer. Il pourrait voir les tendances qu'il a et ainsi faire un travail plus précis sur sa technique ou son réglage viseur.

6.3.4 Social

Faire de cette application un genre de réseau social entre les Archers avec un partage des statistiques ou des scores, des comparaisons ou des Top Archers. Bref dans le social il y a encore une grande marche de manœuvre concernant l'application. Et grâce à ce TPI j'ai construit une base solide pour l'ajout des fonctionnalités futures.

6.3.5 Mode Offline

Le mode offline de cette application pourrait être mis en place afin que même si l'on n'a pas de réseau (compétitions dans un autre pays par exemple…) on puisse entrer nos *Shoots* et que l'application se chargerait de les charger une fois que le réseau serait rétabli ou que l'utilisateur le décide. Pour cela il faudrait changer quelque peu l'architecture actuelle de l'application, en effet, actuellement elle est énormément dépendante de l'API, ce qui est le but tout de même pour une Application WEB.

6.3.6 Modifications et Suppressions

Évidemment ceci reste dans les fonctionnalités de bases que je n'ai pu mettre en place dans mon TPI par manque de temps, en effet, ces requêtes (modifications de flèches, modification de *Shoot* ou même suppression d'un *Shoot*) n'ont pas étés codées dans l'API.
Cependant grâce à Vue les composants sont facilement modifiables et l'on peut rajouter la modification des flèches et cætera rapidement une fois que les requêtes auront étés crées dans l'API.

6.3.7 Améliorations Dashboard

Actuellement le dashboard récupère tous les *Shoots* fournis par l'API, dans l'avenir il serait nécessaire de mettre un système de pagination afin que quelques shoots soient affichés

petit à petit. Ceci permettra pour les utilisateurs ayant énormément de *Shoots* de ne pas attendre que tous leurs *Shoots* soient chargés et visionner ainsi les plus récents rapidement.

6.3.8 Pages d'erreurs

Suite à un oubli de ma part les pages d'erreurs retournées par l'API sont basiques, c'est du code JSON brut, pour le moment je *toast* l'erreur ce qui n'est pas totalement une mauvaise chose mais l'utilisateur n'y comprends pas grand-chose, il faudrait mettre en place des pages d'erreurs si l'utilisateur se trompe de chemin etc…

6.4 Ressenti

Mon ressenti sur ce projet est partagé, je suis fier de ce que j'ai accompli en 3 semaines et de ce que j'ai appris, je me suis surpassé ou en tout cas j'en ai eu l'impression car ce n'était pas évident pour moi.

En même temps je me dis que j'aurais pu faire plus si je recommençais. Il est vrai qu'au début j'avais des grosses lacunes sur les technologies utilisées.
Le rapport m'a pris énormément de temps et malgré les avertissements répétés je l'ai négligé au long du projet, je pense que sans le rapport j'aurais pu mettre en place bien plus de fonctionnalités.

Mais dans l'ensemble je suis satisfait de ce projet et me réjouis de continuer à développer et à améliorer ça hors TPI !

7 Annexes

Ici se trouve la description des Annexes et les sources utilisées lors du projet

7.1 Sources – Bibliographie

https://vuejs.org/v2/guide/
https://vuex.vuejs.org/en/
http://passportjs.org/docs
http://www.chartjs.org/docs/
http://materializecss.com/
http://stackoverflow.com/ (je n'ai malheureusement pas gardé tous les liens)
https://uxplanet.org/optimistic-1000-34d9eefe4c05
https://alligator.io/vuejs/global-event-bus/
https://webpack.js.org/concepts/

J'ai beaucoup utilisé les sources officielles de VueJS et des Framework / librairies que j'utilisais et peu d'autre liens.

7.2 Journal de travail

Présent en **Annexes (Journal de travail)**

7.3 Manuel d'Installation

Présent en **Annexes (Production)**

7.4 Manuel d'Utilisation

Présent en **Annexes (Utilisation)**

7.5 Archives du projet

Le courrier envoyé contient Le Rapport imprimé avec ses Annexes reliés par des anneaux métalliques ainsi qu'un CD ROM contenant toutes mes Sources :

- Documentation
  - Annexes
    - Dans ce dossier se trouve les docs et les avancements fait avant le TPI et les Annexes du rapport
  - Rendu
    - Le dossier rendu pour le dossier final du TPI et les différents PDF générés
- Sources
  - Cli
    - Les tests sur l'API sont présents dans ce dossier
  - Site
    - api
      - dal --> Data Access Layer (Dans ce dossier se trouve les modèles et les méthodes de la DB)
      - middlewares (Dans ce dossier se trouves l'initialisation du module d'authentification et les contrôles effectués)
      - controllers (Chemins différents pour gestion de L'API)
    - config --> config nécessaire pour l'API (connexion DB, ...)
    - client
      - Le dossier désigné pour le *build* de Vue.js (Ce sera donc majoritairement du code généré automatiquement
    - TPI_Frontend : Ici ce situe le code fait par mes soins qui est compilé par la suite dans le dossier api > client

Annexes