2017-2018

2ème année du cycle ingénieur

Spécialité Systèmes Automatisés et Génie Informatique

Babyfoot Connecté

Projet EI4

Ahmed Youssouf ZIYYAT
Steve DESPRES
Florent YVON

Sous la direction de
MM. Mehdi LHOMMEAU et Laurent BORDET

Membres du jury
Mehdi LHOMMEAU | Maitre de Conférences
Bertrand COTTENCEAU | Maitre de Conférences

Soutenu publiquement le :
16 avril 2018

IstiA
ÉCOLE D'INGÉNIEURS
UNIVERSITÉ D'ANGERS
École associée Polytech
université angers

REMERCIEMENTS

Nous tenions premièrement à remercier M. Mehdi LHOMMEAU pour nous avoir encadré et conseillé durant ce temps de projet. Ses conseils et son expertise nous ont permis d'arriver à un résultat convenable et dont nous sommes fiers.
Il convient aussi de remercier l'Université d'Angers à travers son vice-président délégué à la cohésion sociale : Laurent BORDET, sans qui ce projet n'aurait tout simplement pas pu avoir lieu puisque l'idée d'acheter et d'interconnecter les babyfoots est la sienne.
Nous voudrions aussi remercier M. Matthieu LEMAITRE, son aide pour dessiner, sous SolidWorks, les supports de nos capteurs nous a aussi permis de faire un grand pas dans les tests de nos solutions diverses.
Nous aimerions aussi remercier L'ISTIA – Ecole d'ingénieurs de l'Université d'Angers, qui via son FABLAB nous a mis a disposition un espace pour entreposer notre babyfoot de test mais aussi proposé des services d'impressions 3D parfaits.
Enfin, nous remercierons nos camarades d'EI4 SAGI pour nous avoir aider à réaliser des tests grandeur nature et toutes les personnes qui de près ou de loin nous ont permis d'avancer dans ce projet.

Table des matières

Introduction

Nous sommes Florent YVON, Ahmed Youssouf ZIYYAT et Steve DESPRES, trois étudiants de l'ISTIA, Ecole d'ingénieurs de l'Université d'Angers, en 2ème année de cycle ingénieur spécialisé en Systèmes Automatisés et Génie Informatique (SAGI).

Figure 1 : De gauche à droite, Steve, Ahmed Yousouf et Florent

La 2ème année de cycle ingénieur à l'ISTIA impose la réalisation d'un projet en groupe de 80h, notre projet s'est étalé de fin janvier à mi-avril 2018, à raison de 8h par semaine. Nous avons choisi de travailler sur un sujet proposé par l'Université d'Angers et plus précisément son vice-président délégué à la cohésion sociale : Laurent BORDET. Dans le cadre de l'achat de plusieurs babyfoots par l'Université, il souhaitait pouvoir connecter ces babyfoots disséminés partout sur Angers et ainsi permettre un contact différent tant entre personnels eux-mêmes qu'aussi entre personnels et étudiants. Il y avait là la volonté d'une dimension du « mieux-être au travail » et de peupler les espaces détentes. Enfin, M. BORDET voyait aussi l'opportunité de développer une culture numérique pour le personnel de l'Université d'Angers, les éveiller à des nouvelles technologies et pourquoi pas nouveaux hobbys.

Une fois que nous avons eu l'essentiel du cahier des charges en tête, nous avons cherché les solutions pour pouvoir détecter automatiquement les buts marqués mais aussi celles pour créer une application capable de gérer ce qui sera le début d'un réseau social : c'est-à-dire les profils des joueurs, l'affichage des scores, la mise en mémoire de ces derniers mais aussi la réservation d'un babyfoot.

Ce rapport détaillera les moyens que nous avons utilisés et les résultats que nous avons obtenus, les choix que nous avons dû faire et les possibilités d'amélioration de ce projet par une future équipe de projet. Il sera aussi accompagné d'une étude sur la blockchain car nous avons envisagé d'utiliser cette technologie pour stocker les scores de manière sûre.

Conception du projet

1. Origines

Ce projet a été proposé par M. BORDET pour le compte de l'Université d'Angers. En effet, plusieurs enquêtes ont montré que le babyfoot est un formidable moyen, dans les entreprises, de renforcer le contact entre les salariés mais aussi de favoriser l'échange en aplanissant les différences hiérarchiques. C'est dans cette optique que l'Université d'Angers a décidé de s'équiper de plusieurs babyfoots, à raison d'un par composante, et qu'elle a vu dans cette optique la possibilité de faire appel au savoir-faire de ces étudiants ingénieurs pour les connecter et réaliser un mini réseau social du babyfoot estampillé UA.

Chacun d'entre nous a choisi de participer à la réalisation de ce projet pour diverses raisons. Principalement c'est l'attrait des objets connectés, la possibilité d'avoir une application de la technologie blockchain et le fait que le projet implique à la fois une partie software et hardware donc qu'il y ait matière à faire avec ses mains. Le défi de partir de zéro est aussi un des aspects qui nous a attiré en plus du fait que l'on faisait un projet pour l'Université d'Angers dont nous pourrions plus tard être les bénéficiaires.

2. Cahier des charges

Les objectifs et consignes du cahier des charges étaient très clairs d'après la fiche projet que nous avons eu, disponible en Annexe 1. Une synthèse des livrables attendus pourrait être la suivante :

- Un dispositif de détection automatique des buts marqués
- Un système d'affichage en temps réel du score
- Un système de connexion par profil pour accéder à ses propres statistiques
- Stockage des statistiques et des données en général dans une base de données
- Mise en place d'un dispositif simple de déploiement et documenté

Pour beaucoup de ces attentes nous avons dû chercher les meilleures solutions avec le bénéfice de partir de rien et donc d'avoir toutes les possibilités, sachant que nous avions peu de connaissances sur le sujet donc aucun biais n'était possible.

3. Planification

Chaque projet de groupe nécessite une partie gestion de projet et une planification des tâches. Cela permet d'harmoniser les méthodes, de vérifier le bon déroulé du projet et d'avancer avec les mêmes outils.

3.1. Répartition des tâches

Le diagramme de GANTT suivant permet de se rendre compte quelles ont été les différentes phases du projet et les grandes étapes qui l'ont rythmé. Plusieurs tâches ont été réalisées en simultané et c'est là l'avantage d'avoir un groupe projet car chacun de nous a pu se concentrer sur un sujet différent. Nous avons toujours mis nos connaissances acquises en commun pour que cela profite à tous. Nous remarquerons que ce sont les parties serveur et application qui ont pris le plus de temps car c'était celles où nous avions le moins de connaissance

alors que pour la partie hardware avec la carte Arduino UNO et un Raspberry Pi nous avions quand même quelques bases bien utiles.

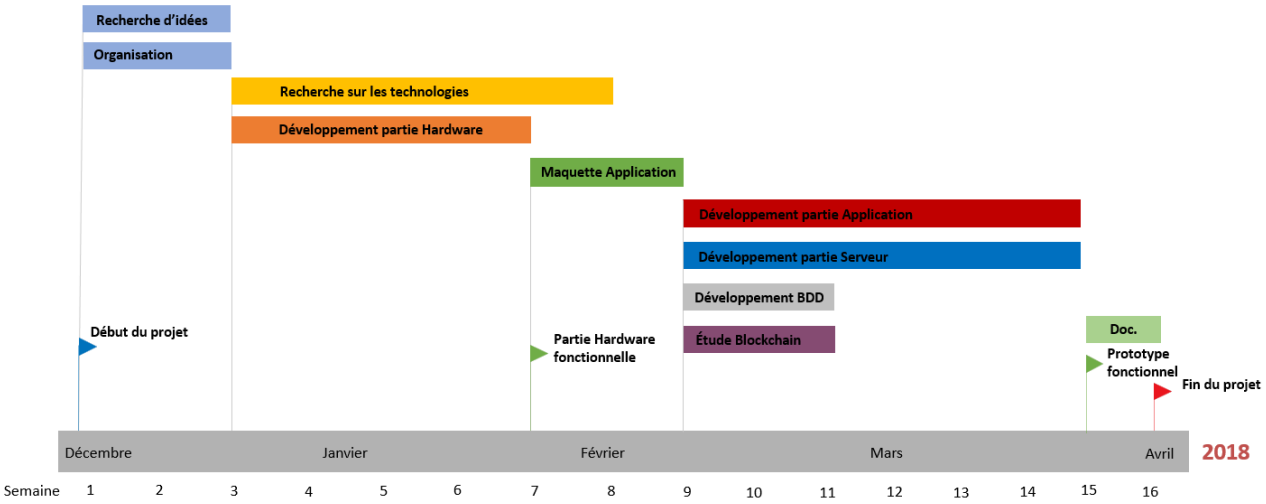


Figure 2 : Diagramme de GANTT du projet

Une autre vue du projet réside dans l'importance de chaque tâche sur le total de temps passé sur le projet, en sachant que nous avons tous travaillé en dehors des heures de projet. Ci-dessous un diagramme circulaire représentant le temps horaire passé sur chaque tâche :

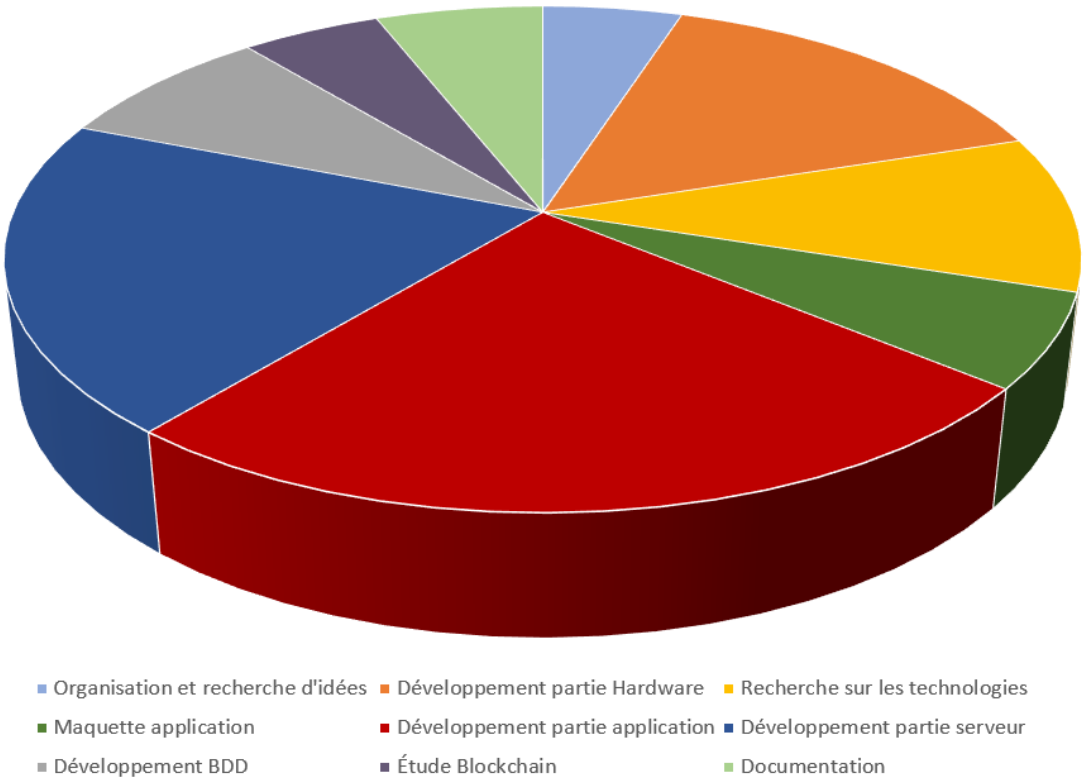


Figure 3 : Diagramme circulaire de la représentation de toutes les tâches du projet

On remarque ainsi que ce sont les parties de développement : software, hardware et serveur qui ont été les plus gourmandes en temps mais c'est le cœur du projet donc il y a une certaine logique.

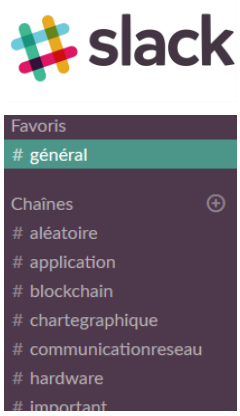
3.2. Méthodes de travail

Concernant les méthodes de travail, nous nous sommes retrouvés tous les mercredis au FABLAB de l'ISTIA pour travailler ensemble et faire le point régulièrement sur l'avancée du projet. Nous avons aussi chacun travaillé un certain temps en dehors des temps consacrés au projet afin de pouvoir aller le plus loin possible et avoir le meilleur rendu possible.

Pour une question d'organisation et de communication, nous avons utilisé plusieurs outils :

- **GitHub** : plateforme de gestion et d'hébergement de code. Nous l'avons utilisé pour tous les programmes que nous avons développés. De plus, cela nous permet de partager notre projet et de lui donner de la visibilité.

- **Slack** : plateforme de communication pour la gestion de projet. Nous avons mis en place un Slack avec différents Channel pour communiquer autour du projet et partager des documents. De plus ce Slack pourra servir aux futurs étudiants qui travailleront sur le projet. Une partie de l'arborescence de ce Slack est disponible à droite.



- **Machine virtuelle** : pour la connexion SSH avec la Raspberry, nous avons créé une machine virtuelle pour avoir tous la même configuration.

- **Adobe XD** : afin de prototyper notre application et réaliser plusieurs vues des pages importantes, nous avons utilisé Adobe XD. Elle nous a permis de montrer une première version des vues, de l'arborescence et de la navigation à M. LHOMMEAU et M. BORDET. Des vues de la maquette seront disponibles en Annexe 2.

Réalisation du projet

1. Choix des technologies

Réaliser et développer un projet implique de faire des choix principalement en termes de technologies. Dans notre cas, il fallait à la fois : choisir le matériel hardware pour détecter les buts et les faire remonter à l'application mais aussi choisir quelles technologies utiliser pour cette application et la base de données qui serait utilisée pour tout stocker.

1.1. Hardware

Concernant la partie Hardware de notre Babyfoot, voici les différents choix des technologies que nous avons fait :

- **Capteurs ultrason HC-SR04** :

Figure 4 : Capteur HC-SR04

Nous nous sommes renseignés sur plusieurs types de capteurs pouvant permettre de capter un but.

Les piézoélectriques (détection de choc) pourraient être placés au niveau des buts, cependant leurs installations seraient difficiles et les mouvements brusques des joueurs pourraient fausser les mesures.

Les infra-rouges pourraient répondre à nos besoins cependant en fonction de la matière des balles (réflexion de la lumière), les mesures pourraient être différentes.

Nous avons donc choisi d'utiliser des capteurs ultrasons, faciles à installer et à utiliser.

- **Arduino UNO** :

Pour traiter les données des capteurs et détecter lorsqu'il y a un but, nous utilisons une carte Arduino UNO qui est adaptée pour ce type de capteur et facile à utiliser elle aussi.

Figure 5 : Carte électronique Arduino UNO

- **Raspberry Pi 3** :

Le cœur de notre Babyfoot sera un Raspberry Pi 3 sous Raspbian. Son rôle est triple : récupérer les données de l'Arduino UNO, communiquer avec le serveur de notre application et afficher une interface utilisateur pour la création des parties.

Figure 6 : Rasberry Pi 3

- **Ecran tactile pour Raspberry**

Pour l'affichage d'une interface utilisateur, nous avons choisi l'écran tactile officiel de Raspberry.

Figure 7 : Ecran tactile pour Raspberry

1.2. Software

- Frontend avec **IONIC**

Pour le développement de notre application et sur les conseils de M. LHOMMEAU, nous avons utilisé le Framework IONIC 2 qui permet la création d'application Web mais aussi Android, IOS et Windows par l'exécution d'une ligne de commande. Ce Framework est écrit en TypeScript pour la logique algorithmique et utilise les langages tels que HTML et CSS pour l'affichage et AngularJS pour de la communication avec une base de données.

- Backend sous **NodeJS**

Côté Backend, il existe des APIs fournissant plusieurs fonctionnalités (système d'authentification, de notification, etc.) comme Firebase de Google, Ionic Pro, Parse, Passport et bien d'autres. Les avantages principaux sont le gain de temps, la simplicité et la sécurité. Cependant, ces outils sont limités et / ou payant, et en les utilisant nous sommes dépendants de leur système et de leur politique d'utilisation des données. Or, cette application devant être hébergée par des services de l'Université d'Angers, la Direction Du Numérique ne nous autorisait pas l'utilisation de tels services.
Nous avons donc décidé de développer notre propre backend, basé sur un serveur NodeJS.

- Base de données avec **MongoDB**

D'après nos recherches, les SGBD orienté documents sont les systèmes les plus adaptés pour fonctionner avec les technologies IONIC et NodeJS. Nous avons alors choisi MongoDB puisque c'était le système utilisé par les principaux tutoriels que nous avons suivi. Nous allons voir par la suite que son utilisation est bien différente des systèmes plus classiques comme MariaDB ou Oracle.

- Temps réel avec **Socket.io**

Pour communiquer en temps réel avec le serveur de notre application et la Raspberry du Babyfoot, nous utiliserons le module Socket.io qui est performant et adapté à ce type d'utilisation.

- Structure du Serveur avec **Express**

Notre serveur Node.JS sera structuré grâce à Express qui permet la mise en place d'une infrastructure flexible et fournit un ensemble de fonctionnalités pour les applications Web et mobiles.

Récapitulatif de notre architecture système :

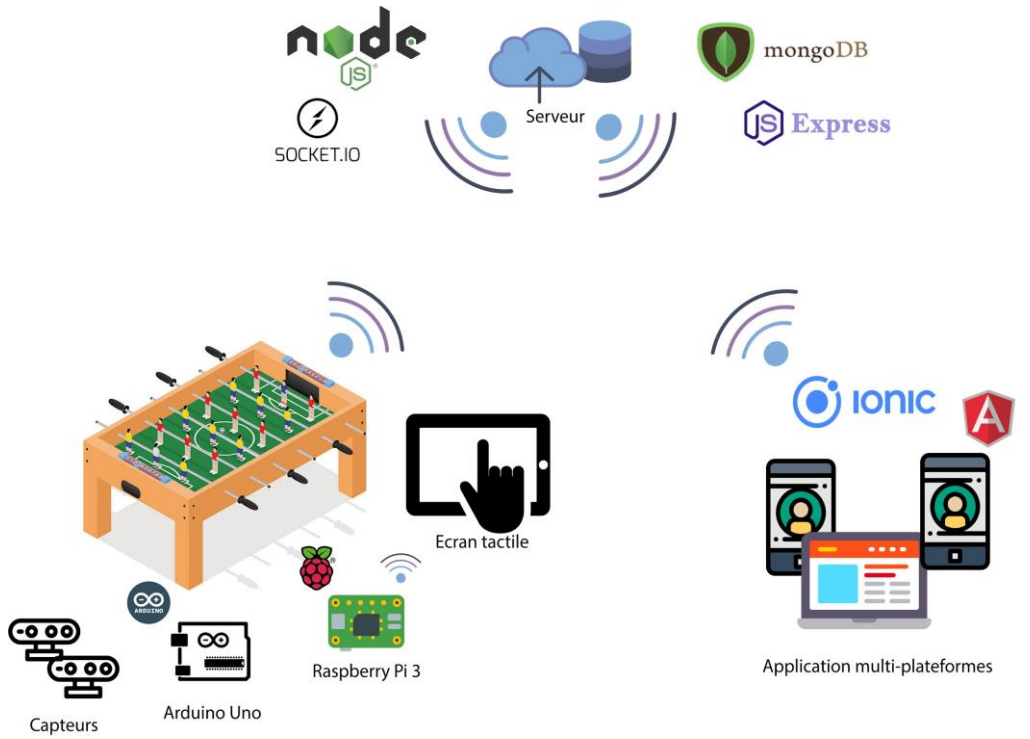


Figure 8 : Schéma de communication entre nos différents systèmes

2. Hardware

Une fois que nous avions arrêté nos choix sur des capteurs ultrason et un Raspberry Pi 3, il restait toute une partie configuration du matériel, intégration au babyfoot et communication entre tous ces éléments.

2.1. Arduino et Capteur

Afin de détecter les buts, nous utilisons le capteur ultrason qui sera fixé sur le passage de la balle, celui-ci détectera le passage de celle-ci dès que la distance calculée est différente de la distance référence. Avec ce capteur seul, la détection de gamelle ne sera pas possible.

Pour la détecter il faudra soit mettre en place un capteur infrarouge ou un capteur piézoélectrique afin de récupérer un signal lorsque la balle touche le fond du but. Si après ce signal, la balle n'est pas passé par le capteur ultrason, cela sera considéré comme une gamelle.

Dans notre application nous nous intéressons seulement à la détection de but. Nous utiliserons le capteur **HC-SR04**

Caractéristiques :

- Dimensions : 45 mm x 20 mm x 15 mm
- Plage de mesure : 2 cm à 400 cm
- Résolution de la mesure : 0.3 cm
- Angle de mesure efficace : 15°
- Largeur d'impulsion sur l'entrée de déclenchement : 10 µs

Figure 9 : Capteur HC-SR04

Broches de connexion :

- Vcc= Alimentation +5 V DC
- Trig= Entrée de déclenchement de la mesure (Trigger input)
- Echo= Sortie de mesure donnée en écho (Echo output)

Fonctionnement :

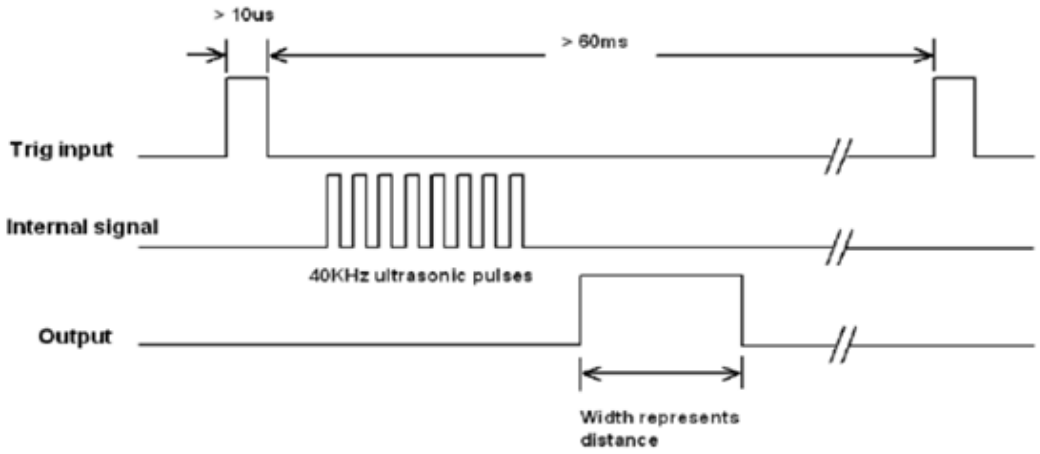


Figure 10 : Fonctionnement du capteur HC-SR04

Pour déclencher une mesure, il faut émettre une impulsion au Niveau Logique 1 (5V) d'au moins 10µs sur l'entrée "Trig". Le capteur émet alors une série de 8 impulsions ultrasoniques à environ 40 kHz, puis il attend le signal réfléchi. Lorsque celui -ci est détecté (~ détection d'objet), il envoie un signal Niveau Logique 1 sur la sortie "Echo", dont la durée est proportionnelle à la distance mesurée.

Calcul de distance :

d =Vs * t (distance = vitesse * temps) *Avec Vs(vitesse du son) = 340m/s ou cm/µs*

Programmation Arduino :

Grâce au code C exécuté à l'initialisation, nous prenons 10 valeurs de distance calculées par l'Arduino afin d'avoir une moyenne puisqu'il peut y avoir quelques petites différences entre chaque mesure. La moyenne est alors la référence sur laquelle se base le programme pour détecter la balle qui passe devant le capteur.
Afin d'éviter les fausses détections, nous autorisons 15 mm de variation par rapport à la moyenne.

Câblage :

```
// ---- capteur 1 -----
        const byte TRIGGER_PIN = 2; // Broche TRIGGER
        const byte ECHO_PIN = 3;        // Broche ECHO
// ---- capteur 2 -----
        const byte TRIGGER2_PIN = 6; // Broche TRIGGER
        const byte ECHO2_PIN = 7;       // Broche ECHO
```

```
/* 1. Lance une mesure de distance en envoyant une impulsion HIGH de 10µs sur la broche TRIGGER */
digitalWrite(TRIGGER_PIN, HIGH);
delayMicroseconds(10);
digitalWrite(TRIGGER_PIN, LOW);

/* 2. Mesure le temps entre l'envoi de l'impulsion ultrasonique et son écho (si il existe) */
long measure = pulseIn(ECHO_PIN, HIGH, MEASURE_TIMEOUT);

/* 3. Calcul la distance à partir du temps mesuré */
float distance_mm = measure / 2.0 * SOUND_SPEED;
if(distance_mm >=(moy1 + distanceDif) || distance_mm < (moy1-distanceDif))
{
  Serial.println("b");
  delay(2000); //delay 2s
}
```

Exemple de mesure (*GoalCatcher.ino*) :

Ce code réalise des mesures et envoie un « b » quand il y a but. Le « b » signifie qu'il y a but pour l'équipe « Bleue » (et un « r » pour rouge). Le delay de 2 secondes sert à éviter une détection de deux buts au même instant. Cela sera utilisé par le serveur Ionic afin de changer l'affichage pour la bonne équipe.

2.2. Modélisation 3D du support capteur

Comme indiqué dans les caractéristiques du capteur HC-SR04, la plage de mesure est à partir de 2 cm. En dessous de cette distance, les valeurs ne sont pas forcément correctes et sont donc à éviter. Pour cela, nous avons pensé à forcer la balle à passer devant le capteur avec une distance comprise dans la plage de mesures avec le support suivant :

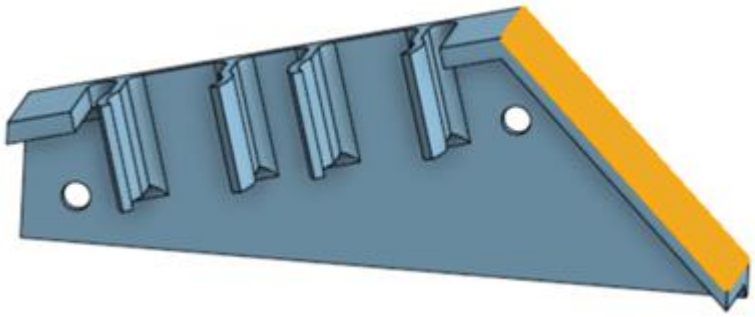


Figure 11 : Modélisation 3D du support du capteur HC-SR04

Ce modèle est réalisé pour la partie droite du Babyfoot. Celui de gauche est un miroir de celui-ci. La partie orange est l'angle qui forcera la balle à passer dans la plage de mesures du capteur, qui sera placé là où passe toutes les balles avant d'aller dans le réceptacle. L'angle a été calculé de telle manière que le capteur soit en parallèle avec l'obstacle d'en face qui est le bois du Babyfoot mais aussi pour avoir un angle dans lequel les mesures sont plutôt efficaces.

Le hasard fait bien les choses, la plage de mesures efficaces du capteur comprend notre angle de passage (15°). Ce support a été réalisé de manière à être facile à mettre en place. Les deux trous sont prévus pour visser le support tandis que la surface plane peut être utilisée pour y mettre de la colle forte.

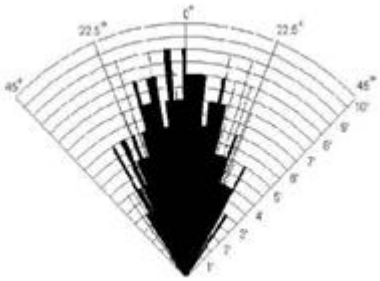


Figure 12 : Efficacité du capteur HC-SR04 en fonction de l'angle

2.3. Raspberry Pi 3

Configuration :

La première chose à faire sur le Raspberry a été d'installer un système d'exploitation. Nous avons donc installé Raspbian Stretch sur une carte SD bootable ; cela se fait assez facilement en suivant la documentation officielle.

Sur nos postes de travail, nous avons créé une machine virtuelle sous Ubuntu avec VirtualBox pour avoir le même environnement lors du développement sur Raspberry (mot de passe : azerty). De plus, elle pourra facilement être reprise par les futurs étudiants qui travailleront sur ce projet. Nous avons affecté une IP statique à cette machine virtuelle : **192.168.1.x** (x = 1,2 ou 3 suivant l'utilisateur).

De la même manière, nous avons affecté une adresse IP statique sur le Raspberry **192.168.1.10.** Une fois le Raspberry et les postes de travail sur le même réseau, grâce à un **switch**, nous avons pu établir une connexion SSH (user : pi, mot de passe : raspberry). Enfin nous avons installé **rsub** pour pouvoir ouvrir les fichiers de le Raspberry en SSH avec *SublimeText* qui est installé sur la machine virtuelle.

```
ssh -R 52698:localhost:52698 pi@192.168.1.10
```

Par la suite, nous avons installé l'écran tactile officiel Raspberry. L'interface graphique se lance parfaitement dessus.

Communication avec l'Arduino UNO :

Pour faire communiquer l'Arduino et le Raspberry, plusieurs solutions sont possibles :

- Via les Bus I2C
- Via Modbus over TCP/IP
- Via le Port Série (câble USB)

Pour une question de simplicité et d'efficacité, nous avons choisis la communication via le port Série. Comme expliqué précédemment, l'Arduino lit les données des capteurs et écrit "r" ou "b" sur le port série lorsqu'un but est marqué par l'équipe rouge ou bleue.

Le Raspberry doit alors récupérer ces données, "r" ou "b", puis les transmettre au Serveur. Pour faire cela, nous avons donc besoin d'un serveur nodeJS local, du module **serialport** installé pour lire les données du port série et de **socket.io** pour les transmettre au serveur. Nous envoyons aussi le nom du babyfoot au serveur ; cela lui permet d'identifier le babyfoot connecté (le serveur enregistre la liste des babyfoots connectés dans un objet global ; explication dans la partie "Système de création de partie").

```
"use strict";
//Lecture du port serie (usb) de l'arduino
const SerialPort = require('serialport');

//Connexion du client au serveur
var io = require('socket.io-client');
//Connection de Socket.io au serveur
var socket = io.connect('http://192.168.1.1:8080');
//Nom du babyfoot
var nameBF = "BF_ISTIA";

//Connection au Port Série pour lire les données arduino
const Readline = SerialPort.parsers.Readline;
const port = new SerialPort('/dev/ttyACM0', {
    baudRate: 115200
});
//On envoi la confirmation de connaxion au serveur
socket.emit('connection-bf', {"nameBF" : "BF_ISTIA" });
const parser = port.pipe(new Readline({ delimiter: '\n' }));

//Quand on reçoit une donnée sur le port Série
parser.on('data', function (data) {
    //On envoie les scores au serveur
    socket.emit('maj-score',{ "goal": data[0], "nameBF" : nameBF} );
});

console.log("Serveur running");
```

Figure 13 : Serveur local sous NodeJS

3. Software

La partie Hardware étant définie, configurée et installée. Nous pouvons nous intéresser à la partie software et toutes les technologies utilisées. Des exemples documentés et commentés seront disponibles.

3.1. Application cross-platform avec Ionic

Le Framework a été choisi tout de suite dès le début du projet pour ses avantages :

- Responsive pour un affichage dans un navigateur Web aussi bien que sur un écran de mobile ou tablette
- Facilité de développer l'appli Web que l'on déploie ensuite sous Android, iOS ou Windows
- Facilité de rédaction en HTML et CSS mais aussi grâce aux Ionic Components signalés par une balise ***<ion-...>***
- Facilité d'ajout de contenus en lignes de commande : en se plaçant dans le dossier Ionic du projet, l'exécution de la commande : `ionic generate page NomDePage` ajoutait directement un dossier NomDePage au dossier Pages de Ionic et le peuple d'un fichier HTML, CSS et TypeScript du même nom et tous reliés. L'arborescence de notre projet n'en était que plus claire. Il en était de même pour un provider, fichier en TypeScript, que l'on pouvait partager et intégrer à plusieurs fichiers Typescript de pages à la fois.

Nous allons donc vous présenter un exemple de code utilisant tous les fichiers d'un dossier d'une page. Nous prendrons l'exemple de l'affichage de la liste des babyfoots.

```
<ion-header>

    <ion-navbar>
        <ion-title>Les babyfoots de l'UA</ion-title>
    </ion-navbar>

</ion-header>


<ion-content class="card-background-page">

    <ion-card>
        <img src="../assets/imgs/b1.jpg" />
        <div class="card-title">Présidence</div>
        <div class="card-subtitle">41 Parties Jouées</div>
        <ion-fab right top>
            <button ion-fab (click)="map('1')">
                <ion-icon name="pin"></ion-icon>
            </button>
        </ion-fab>

    </ion-card>
```

Figure 14 : Page Liste Babyfoots Application et son code correspondant en HTML

Comme vous pouvez le voir, la combinaison de balises classiques HTML comme « div » avec une « ion-card » donne un résultat très correct en très peu de lignes.

Pour obtenir un tel résultat, il faut ajouter quelques lignes au fichier CSS, comme suit :

```css
page-listbabyfoots {
    .card-background-page {
        ion-card {
            position: relative;
            text-align: center;
        }
        .card-title {
            position: absolute;
            top: 36%;
            font-size: 2.0em;
            width: 100%;
            font-weight: bold;
            color: #fff;
        }
        .card-subtitle {
            font-size: 1.0em;
            position: absolute;
            top: 52%;
            width: 100%;
            color: #fff;
        }
    }
}
```

Figure 15 : Fichier CSS associé à l'affichage de la liste des babyfoots

A ces deux fichiers, il est possible d'ajouter un fichier TypeScript (.ts) qui est un sur-ensemble de JavaScript en plus sûr et avec une logique orientée objet empruntée au C# puisque son inventeur, chez Microsoft, a énormément contribué au développement de C#. Ce fichier TypeScript permet l'écriture de toutes les fonctions à exécuter en appelant ou cliquant sur la page.

Dans la Figure 16, la ligne `<button ion-fab (click)="map('1')">` exécute la fonction map avec l'argument 1 qui est codée dans le fichier TypeScript associé. Cette fonction est la suivante :

```typescript
@IonicPage()
@Component({
  selector: "page-listbabyfoots",
  templateUrl: "listbabyfoots.html"
})
export class ListbabyfootsPage {
  destination: string;
  start: string;

  constructor(public navCtrl: NavController, public alertCtrl: AlertController, private launchNavigator: LaunchNavigator) { this.start = "";}

  ionViewDidLoad() {
    console.log("ionViewDidLoad ListbabyfootsPage");
  }

  map(adr) {
    switch (adr) {
      case "1":
        this.showPopup('Babyfoot de la Présidence', "Présidence de l'Université d'Angers<br/>40 rue de Rennes<br/>49035 Angers<br/>02 41 96 23 23","Présidence de l'université d'
        break;
      case "2":
        this.showPopup(`Babyfoot de l'ISTIA`, "Ecole d'Ingénieurs ISTIA<br/>62 avenue Notre Dame du Lac<br/>49000 Angers<br/>02 44 68 75 00","ISTIA, 62 avenue Notre Dame du Lac,
        break;
      case "3":
        this.showPopup('Babyfoot Fac de Sciences', "Faculté de Sciences<br/>2 Boulevard Lavoisier<br/>49000 Angers<br/>02 41 73 53 53","Fac de Sciences, 2 Boulevard Lavoisier, 4
        break;
      case "4":
          this.showPopup('Babyfoot SUAPS', "SUAPS<br/>6 Boulevard Beaussier<br/>49000 Angers<br/>02 41 22 69 49","S.U.A.P.S, 49000 Angers");
        break;
    }
  }
```

Figure 17 : Fichier Typescript (.ts) pour l'affichage de la liste de babyfoots

On peut voir que les premières lignes servent à faire le lien entre le fichier HTML et le fichier TypeScript, les lignes sous l'export permettent la déclaration de variables globales, est déclarée ensuite la fonction autogénérée ionViewDidLoad(), qui est appelée à chaque affichage de la page ce n'est qu'ensuite qu'on retrouve notre fonction map(adr), cette même fonction fait appel à une fonction écrite plus bas, cela se reconnait par l'emploi du préfixe « this. ». Cette fonction permet de créer un alertController qui est en fait une fenêtre flottante avec un message et des boutons. La fonction navigate permet l'ouverture d'un navigateur type Google Maps ou Apple Maps et la recherche du lieu préciser en argument.

```
showPopup(title, text, destination) {
  let alert = this.alertCtrl.create({
    title: title,
    message: text,
    buttons: [
      {
        text: 'Retour',
        role: "cancel",
      },
      {
        text: "Voir sur Maps",
        handler: () => {
          console.log("Maps clicked");
          this.destination=destination;
          this.navigate(this.destination);
        }
      }
    ]
  });
  alert.present();
}
```

Figure 18 : Fonction d'affichage d'un AlertController et son résultat dans l'application

3.2. Serveur sous NodeJS et ExpressJS

Node.JS est une plateforme logicielle libre et événementielle en JavaScript et offre un environnement côté serveur qui est rapide, léger et efficace, notamment pour traiter des requêtes en temps réel. Couplé avec le Framework Express, il est possible de créer des applications complètes et structurés.

Voici la structure de notre API :

Le serveur se lance avec la ligne de commande : `node app.js`

app.js : fichier "parent" ou "principale". C'est dans ce fichier qu'on fait appel aux autres fonctions / pages.

```
var app = require('express')();
var server = require('http').createServer(app);
var io = require('socket.io').listen(server);

// Config
require("./config")(app);

// Models
require("./models")(app);

// Routes
require("./routes")(app);

// Communication Socket.io
require("./sockets")(io);

// listen (start app with node server.js) ======
//Localhost
//server.listen(8080);
//Serveur local
server.listen(8080, "192.168.1.1");
console.log("App listening on port 8080");
```

Figure 19 : Configuration du Serveur NodeJS avec ExpressJS

config.js : fichier de configuration de notre serveur

models.js : contient les Schémas et la structure de la Base de données

routes.js : contient les routes de l'API ; précise quelle fonction est appelé lors d'une requête GET ou POST.

```
var auth = require("./controllers/auth");
var data = require("./controllers/data");
var game = require("./controllers/game");

// Routes
module.exports = function(app) {
    app.post('/signin', auth.signin);
    app.post('/signup', auth.signup);
    //app.post('/logout', auth.logout);

    app.post('/getDataPlayer', data.getDataPlayer);
    app.post('/getStatsPlayer', data.getStatsPlayer);
    app.post('/bookReservation', data.bookReservation);
    app.post('/getMyReservations', data.getMyReservations)

    app.post('/joingame', game.joinGame);
    app.post('/createSpeedGame', game.createSpeedGame);
    app.post('/directGame', game.directGame);
}
```

Figure 20 : Code du fichier Routes.js : Routes de l'API

sockets.js : contient toutes les interactions et fonctions des communications avec Socket.io

controllers : contient les contrôleurs de notre API (fonctions)

node_modules : contient les modules de Node.JS installés

package.json et **package-lock.json** : contiennent les informations sur la configuration de notre environnement Node.JS (liste des packages, versions, etc …).

3.3. Base de données avec MongoDB

MongoDB est un système de gestion de base de données basé sur NoSQL, non relationnel, orienté documents.

Les bases NoSQL (Not Only SQL) sont des bases de données issues du monde web et répondant aux problématiques de hautes disponibilités, grandes performances en lecture/écriture ainsi que le traitement de grands volumes de données. Parmi les entreprises ayant d'énormes bases de données, et qui utilisent des bases NoSQL au sein de leur architecture, on trouve Facebook, Twitter, Google, LinkedIn et Amazon.

Différences entre SQL et NoSQL :

- Première différence de taille, SQL organise le stockage de données sur le principe de tables reliées entre elles. La structure et les types des données sont rigides, c'est-à-dire fixés à l'avance avant d'implémenter une logique métier. NoSQL stocke et manipule des documents qui correspondent à des collections d'objets. On peut stocker toutes les données que l'on souhaite dans n'importe quel document.
- En relationnel, avant d'ajouter des données, il est obligatoire de définir une table et de décrire celle-ci avec les noms et le types de données qui y seront. Cette table aura donc une clé primaire (soit simple soit composée), peut être des indexeurs et une clé étrangère si celle-ci a un lien avec d'autres tables de la base. Les tables SQL imposent donc un modèle de données strictes. NoSQL est plus flexible puisqu'on peut stocker des données n'importe où et à tout moment. Il n'est pas nécessaire de spécifier une conception de document ou même une collection à l'avance. Donc une base de

données NoSQL peut être plus adaptée aux projets où les exigences initiales en matière de données sont difficiles à déterminer, malgré que le fait de pouvoir stocker les données n'importe où peut entraîner des problèmes de cohérence.

- Il n'existe pas de jointure en NoSQL.
- NoSQL, contrairement au SQL, est dénormalisé, ce qui veut dire qu'au lieu d'avoir des liens entre les tables, Quelques données seront répétées dans les collections.
- NoSQL stocke et manipule des documents qui correspondent à des collections d'objets.

Revenons à MongoDB, le document est l'unité de base : ce que peut être inséré, supprimé, modifié, recherché... Ces documents sont contenus dans des collections réunies en base de données sous un style JSON. Un document peut être vu comme un ensemble de couples de clef/valeur tel que :

```
{
        'nomEtud' : 'Dupond',
        'NumEtud' : '201601'
}
```

Une collection est un ensemble de documents, qui est équivalent à une table en relationnel.

Pour notre projet, on a créé des collections pour les joueurs, les statistiques (buts, victoires, défaites, etc..) et d'autres collections qui sont présentées ci-dessous en forme relationnel pour que cela soit clair.

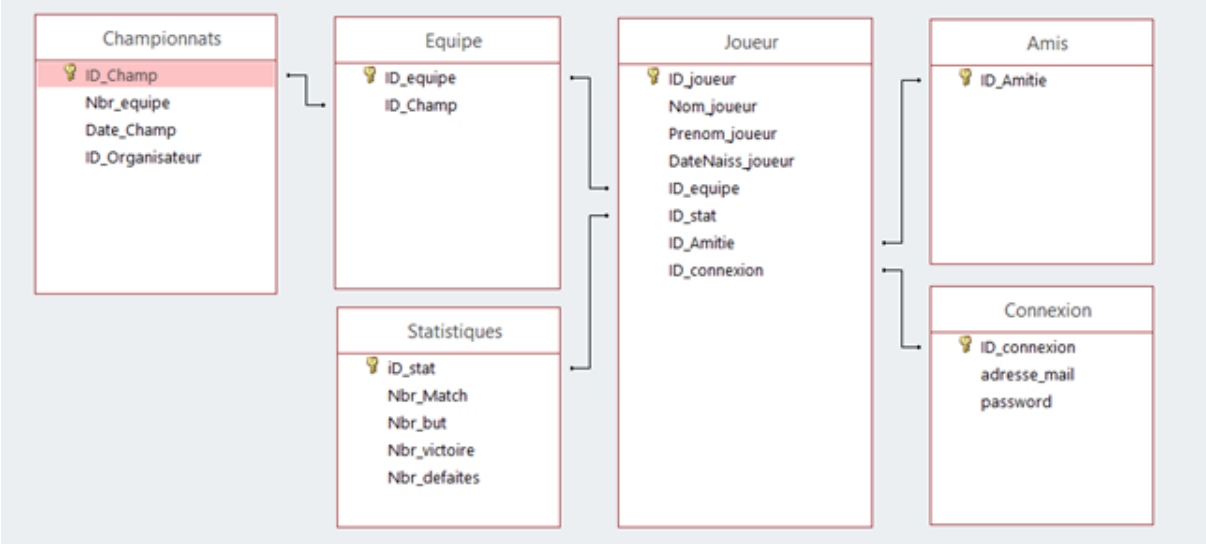


Figure 21 : Aperçu de la modélisation Access de la base de données

3.4. Système d'authentification

Nous avons développé un système d'authentification pour que les joueurs puissent créer un compte et se connecter.

On distingue deux types de comptes, repérable par l'attribut "typeCompte" dans la base de données : le type "joueur" et le type "babyfoot". Un compte de type joueur à accès aux pages lui étant destinées, et le type babyfoot seulement à la page de création de partie. Chaque Raspberry devra être connecté avec un compte de type "babyfoot".

Voici la page "login" permettant l'authentification :

Figure 23 : Page authentification Application

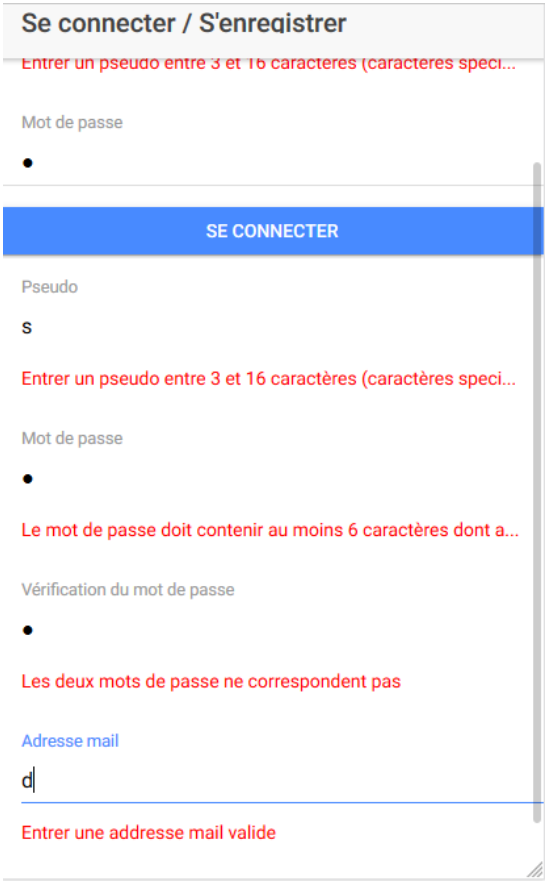


Figure 22 : Page Authentification application avec test des expressions régulières

Création de compte :

Voici les informations que nous demandons à l'utilisateur :

- **Pseudo** : entre 3 et 16 caractères (minuscules, majuscules, chiffres)
- **Mot de passe** : au moins 6 caractères (au moins une minuscule, une majuscule, un chiffre)
- **Vérification du mot de passe**
- **Adresse mail** : format d'une adresse mail

L'utilisateur doit utiliser un pseudo et une adresse mail qui ne sont pas déjà utilisés. Nous contrôlons que les entrées utilisateurs soient au bon format, à l'aide d'expressions régulières, côté client et côté serveur.
La validation des entrées utilisateurs se fait dans le fichier "/pages/login/login.ts". Si les entrées sont valides, la fonction **signup** du **Provider Authentification** ("/providers/authentification/authentification.ts") est appelée. Sur IONIC, un Provider permet de réaliser des requêtes vers un serveur.
La fonction **signup** du provider envoie alors les données de l'utilisateur au serveur à l'aide d'une requête POST :

```
this.http.post('http://192.168.1.1:8080/signup', data, options)
```

Côté serveur, via les routes, la fonction signup du controller auth.js est appelée et reçoit en paramètre les données de l'utilisateur.

Elle contrôle les entrées utilisateurs avec des expressions régulières, vérifie que le pseudo et le mail ne sont pas déjà utilisés puis effectue un salage puis un hachage du mot de passe entrées à l'aide de cette fonction :

```
var crypto = require('crypto');

//Fonction pour création du salage
exports.createSalt = function(){
    //Génère une chaine de caracteres aléatoire (longueur de 16)
    return crypto.randomBytes(Math.ceil(16/2))
    .toString('hex')
    .slice(0,16);
};

//Fonction du hachage + salage du mot de passe
exports.hashPassword = function(password, salt){
    //Hachage du mot de pass + salage avec algorithme sha512
    var hash = crypto.createHmac('sha512', salt);
    //Mise à jour du mot de passe
    hash.update(password);
    //Valeur en hexadecimal
    var pswd = hash.digest('hex');
    return {
        pswd,
        salt
    };
};
```

Figure 24 : Code de la procédure de salage et de hachage des mots de passe

```
//Creation du nouveau joueur dans la base de données
Joueur.create({ pseudo : dataPlayer.pseudo_signup,
                mail : dataPlayer.mail_signup,
                password : secureData.pswd,
                salt : secureData.salt,
                niv : 1,
                typeCompte : 'joueur'},
function(err, user) { console.log(err, user)
    if(err){
        res.send(err);
    }
    console.log("Nouveau compte créé");
    //On envoie true si le compte est bien créé
    res.json('OK');
```

Figure 25 : Code de la procédure d'enregistrement d'un nouveau joueur

Ensuite, les données utilisateur, l'empreinte du mot de passe et le salage sont enregistrés dans la base de données.

Une confirmation est envoyée au client, qui enregistre dans un fichier local (*window.localStorage*) le Pseudo du joueur. Faute d'utiliser des jetons de connexion, cela permet de vérifier que l'utilisateur est bien connecté lorsque l'on change de page.

Connexion d'utilisateur :

De la même façon que la création de compte, les données utilisateur entrées sont vérifiées puis envoyées au serveur.

Puis la fonction **signin** du serveur vérifie que le pseudo existe dans la base de données et que le mot de passe entré, une fois passé par la fonction de hachage avec le salage stocké dans la base de données, correspond à l'empreinte du mot de passe de la base de données.

Si le pseudo et le mot de passe sont vérifiés, une confirmation est envoyée. Sinon, un message d'erreur est affiché.

3.5. Système de création de partie

Voici le schéma général du système de création de partie

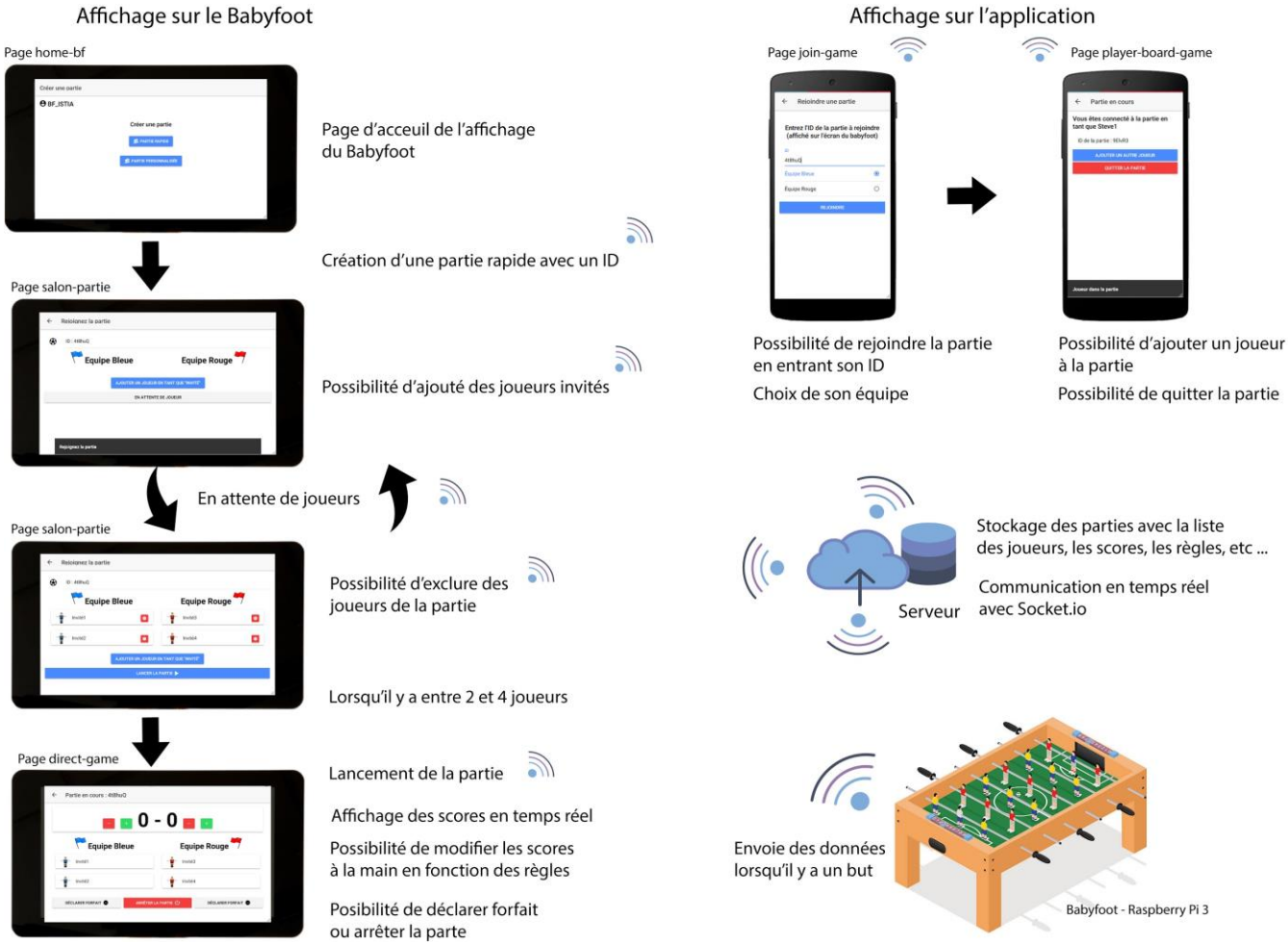


Figure 26 : Schéma général du déroulement de création de partie

Nous donc avons créé 5 pages pour réaliser ce système :

<u>Page Home-bf :</u>

C'est la page d'accueil de l'affichage du Babyfoot. Lorsque l'on se connecte sur l'application avec un compte de type "babyfoot", c'est cette page qui est affichée.

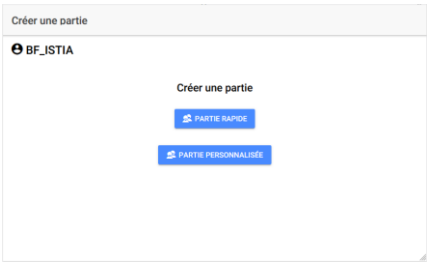


Figure 27 : Vue de la page d'accueil d'un babyfoot

Elle affiche 2 boutons :

- **Partie rapide** : créer une partie rapide, entre 2 et 4 joueurs, en 10 points.
- **Partie personnalisée** : fonctionnalité non développée. Elle devait permettre de définir les règles de la partie (nombre de joueurs, score à atteindre, temps de la partie, etc.)

Lorsque l'on clique sur Partie Rapide, une requête est envoyée au serveur qui crée ensuite une partie dans la base de données avec ces éléments (controllers/game.js) :

- **ID** : id unique de 6 caractères (minuscule, majuscule et chiffre)
- **etat** : état de la partie, ici "en Attente" de joueurs. Son état peut être "enAttente", "enCours" ou "terminee"

- **nomBabyfoot** : correspond au pseudo du compte babyfoot (BF_ISTIA par exemple)
- **nbJoueurs** : object contenant le nombre de joueurs dans chaque équipe et le nombre de joueurs maximum autorisé par partie
- **config** : objet contenant les règles de la partie (limite de score, limite de temps, etc …)
- **joueurs** : objet contenant les informations sur les joueurs de la partie (pseudo, équipe, etc…)
- **score** : contient le score de chaque équipe

Page Salon-Partie :

Une fois une partie rapide créée dans la base de données, le babyfoot affiche le salon de la partie. Cette page avec l'ID de la partie et les joueurs qui la rejoigne. Il est aussi possible d'ajouter des joueurs "invité", sans compte, et d'exclure des joueurs de la partie.

Figure 28 : Vue de l'affichage d'un match avant d'ajouter des joueurs

A partir de ce moment, ce sont des communications en temps réel qui s'effectuent avec le serveur, grâce au module Socket.io. Son fonctionnement est assez simple.

Un socket à deux rôles ; émettre un message et écouter un message. Lorsqu'un client est connecté avec l'application (compte "babyfoot" et "joueur"), il est automatiquement connecté au serveur avec sa socket. Grâce à son socket, il peut envoyer à tout moment un message au serveur Par exemple :

```
//Envoi d'un message nommé 'launch-game' avec l'id de la partie
this.socket.emit('launch-game',  { "idgame" : this.idgame });
```

Le Serveur, de son côté, écoute tous les messages ayant comme nom 'launch-game' et récupère les données envoyées (ici l'id de la partie) :

```
//Ecoute la réception de message nommé 'launch-game'
socket.on('remove-player', function(data){
      //Effectue une action}
```

Il est aussi possible de définir des Rooms. Chaque client dans une même Room peut envoyer un message à tous les autres, et le serveur peut envoyer un message à une Room définie.

Dans notre cas, une Room correspond à une Partie identifié par un ID. Par exemple pour une Partie (ID = df87eR), nous créons une Room (ID=df87eR) dans laquelle sont connectés le serveur, les joueurs de la partie et le babyfoot concerné.

```
//Envoie d'un message nommé "maj-players" dans la Room df87eR
io.sockets.to("df87eR").emit('maj-players', { "dataGame" } });
```

Voici donc comment sont organisés les communications temps réel sur le système de création de partie. Décrire tout le code serait beaucoup trop long, nous vous invitons à regarder notre code commenté du projet pour plus d'informations (Server/socket.js et sur les pages concernées de l'application).

Côté serveur, nous stockons dans des objets globaux toutes les parties en cours (imaginons plusieurs parties lancées sur plusieurs babyfoot), les scores actuels et les babyfoots actuellement connectés. Pour avoir un temps de réponse correct, il n'est pas possible de mettre à jour la base de données à chaque événement, c'est pour cela que nous utilisons des objets globaux.

C'est ces objets, contenant toutes les informations, qui sont mis à jour en temps réel et envoyés à la page salon-partie pour qu'elle affiche les joueurs connectés à la partie, le score, etc …

Objets globaux :

```
var allCurrentGames = {};
var allBabyFoot = {};
var scoreByGame = {};
```

Structure des objets globaux :

```
allCurrentGames : {
    //Exemple pour une partie ayant pour id : df87eR
    "df87eR" = {
        "idgame" : "df87eR",
        "etat" : "enAttente",
        //Nombre joueurs par équipe
        "nbPlayersB" : 1,
        "nbPlayersR" : 1,
        //Nombre joueurs autorisés par la partie
        "nbPlayersMin" : 2,
        "nbPlayersMax" : 4,
        //Objets contenant les info des joueurs de la partie
        "players" : {
                //equipe b
                "b" :{
                    "Steve" : {
                        "pseudo" : "Steve",
                        "idgame" : "df87eR",
                        "equipe" : "b",
                        }
                },
                //equipe rouge
                "r" : {
                    "Florent" : {
                        "pseudo" : "Florent",
                        "idgame" : "df87eR",
                        "equipe" : "r",
                            }
                }
        },
        "config" : {
            "limiteScore" : 10,
        },
    }
};
```

```
allBabyFoot = {
    //Babyfoot identifié par son nom
    "BF_ISTIA" : {
            "nameBF" : "BF_ISTIA",
            //ID de la partie en cours sur ce babyfoot
            "currentGame" : "df87eR",
    }
}
```

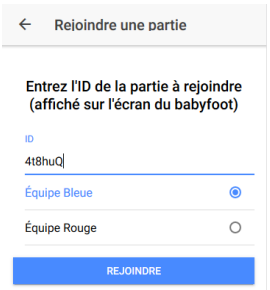
```
scoreByGame = {
    //score de la partie df87eR
    "df87eR" : {
        //Score par equipe
        "b" : 0,
        "r" : 0,
    }
}
```

Figure 29 : Codes des structures des objets d'une partie de babyfoot

Une fois que le nombre de joueurs connectés est suffisant, il est possible de lancer la partie. Lorsque la partie est lancée, le serveur vérifie que le babyfoot correspondant à la partie soit bien connecté, c'est-à-dire qu'il soit bien dans l'objet global "allBabyfoot". Puis la base de données est mise à jour avec l'état "enCours", la liste des joueurs et la page Direct-game est affichée.

Page Join-Game :

Côté compte "joueur", il est possible de rejoindre une partie grâce à la page Join-Game. Il suffit d'entrer l'ID de la partie qui a été généré sur l'affichage du babyfoot, choisir son équipe et cliquer sur "Rejoindre la Parte". Un message va être envoyé au serveur via une socket. Celui-ci va vérifier si la partie existe bien et qu'elle n'est pas déjà complète. En cas de succès, le joueur va être ajouté à l'objet global de la partie en cours et son pseudo sera affiché sur la page Salon-Partie du babyfoot. La page Player-Board-Game est ensuite affichée.



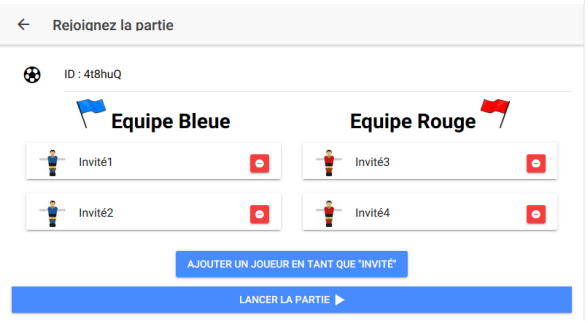
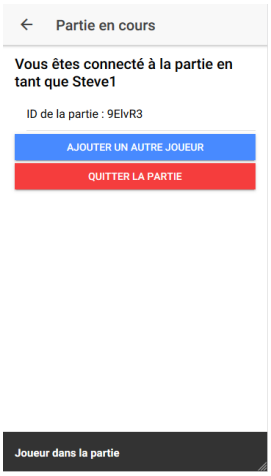


Figure 30 : Vue du match prêt à démarrer

Page Player-Board-Game :

Cette page permet au joueur d'ajouter un ami à la partie en entrant son pseudo. Cela peut être pratique si son ami n'a pas d'accès à l'application. Le joueur peut aussi décider de quitter la partie.



Page Direct-Game :

Cette page affiche le score en temps réel ainsi que les joueurs de la partie. Il est possible de modifier le score actuel à la main, pour gérer les règles (gamelles, but défenseurs, etc.) ou en cas de problème. Il est aussi possible d'arrêter la partie avant la limite de score atteinte, ou encore de déclarer forfait.

Figure 31 : Vue d'un match une fois lancé

Le babyfoot étant connecté (Raspberry connecté au serveur) et la partie en état "enCours", celui-ci envoie en temps réel "r" ou "b" au serveur lorsqu'il y a un but. Cela déclenche un événement durant lequel le serveur met à jour l'objet global des scores et l'envoie à la page Direct-Game, tout cela via des sockets.

Lorsque la partie est arrêtée (arrêt ou forfait) ou terminée (limite de score atteinte), la base de données est mise à jour avec le score actuel et l'état "terminee". Puis l'écran du babyfoot retourne sur la page d'accueil Home-Bf.

3.6. Système de réservation d'un babyfoot et de consultation de ses propres réservations

Dans notre application, nous avons voulu permettre la possibilité de réserver un des babyfoots de l'Université d'Angers, cela permettra à chacun de pouvoir s'organiser et aussi d'être sûr de pouvoir jouer au moment voulu, cette dernière garantie serait assurée par une désactivation du babyfoot ou de l'écran servant à utiliser le système connecté du babyfoot. Il est intégré aussi à l'appli, un moyen de consulter et supprimer ses réservations encore valides.

Pour réserver un babyfoot, il n'y a qu'une seule page qui entre en jeu et elle est accessible depuis le menu principal via le bouton : « Réserver un babyfoot ». Une fois que l'on a cliqué sur ce bouton, on arrive sur cette page, à droite :

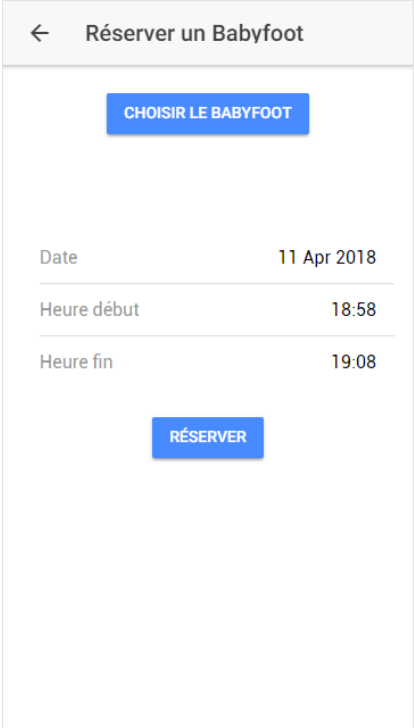
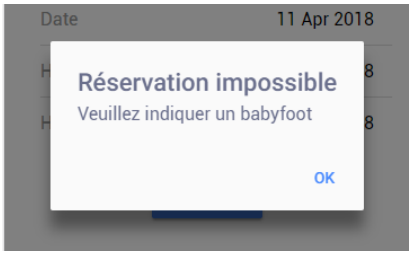


Figure 32 : Accueil pour la réservation d'un babyfoot

Il faut alors choisir un babyfoot, en cliquant sur « Choisir le babyfoot ». Le choix de cacher la liste et de ne sélectionner aucun babyfoot de base a été un parti pris pour éviter des réservations maladroites ou par défaut. Si quelqu'un ne sélectionne pas de babyfoot, il ne pourra pas réserver de créneau et verra apparaitre un pop-up lui signalant que sa réservation est impossible. De plus, sachant que la liste est cachée, une fois qu'un babyfoot a été sélectionné, ce choix s'affiche en dessous du bouton pour que la personne ait toujours l'information à disposition et ne se trompe pas.



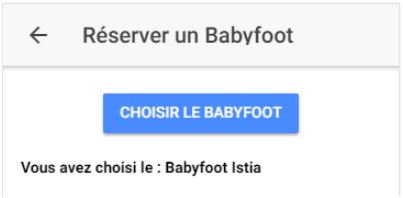


Figure 33 : Résultat du choix d'un babyfoot pour une réservation

Pour cacher et afficher cet élément, il a été nécessaire faire un lien entre le fichier HTML et TypeScript :

- Dans le fichier HTML : `<p id='Resultat'>Vous avez choisi le : {{ testRadioResult }}</p>`
- Dans le fichier TypeScript : `document.getElementById("Resultat").style.visibility = "visible";`

L'élément contenant la phrase s'est vu attribué un ID, qui permettra de le retrouver dans tout le fichier HTML et de modifier sa visibilité dans le fichier TypeScript, une fois la validation faite. De plus, la visibilité est forcée a « Hidden » dans le fichier CSS pour que la phrase ne s'affiche pas au premier chargement alors qu'aucun babyfoot n'a été sélectionné.

Ensuite, la partie choix du créneau a ses champs générés automatiquement par des balises Ionic. Le plus important dans cette partie réside dans l'attribution par défaut des valeurs à ces champs, à chaque fois que l'utilisateur ouvre la page, les champs sont actualisés avec la date, l'heure actuelle pour le début et l'heure de fin est définie à l'heure actuelle plus dix minutes. Le code de cette attribution de variable est assez long et incompréhensible, il sera disponible en Annexe 3. Nous nous attarderons seulement sur une partie de celui-ci.

Le plus important dans la réservation d'un babyfoot est d'enregistrée la réservation avec les conditions que le créneau n'est pas trop long n'y trop loin dans le temps ou avant la date actuelle.
Pour cela, comme pour le login, les entrées utilisateurs sont basiquement testé numériquement. Voici par exemple quelques tests et messages d'erreurs potentiels, on remarque qu'il y a deux paramètres réglables : maxTime, la longueur d'un créneau en minutes et maxDelay, le délai maximal avant la réservation en jours. Ici les deux sont à 30.

```
var bookData = req.body;
var success = true;
//On récupère la date de début et de fin du créneau voulu
var DateDeb = bookData.DateDeb;
var DateFin = bookData.DateFin;
//On enregistre la date à l'instant t en millisecondes
var DNow = Date.parse(now);

//Test des entrées utilisateurs
if (DateDeb > DateFin) {
    success = false;
    result = "Veuillez entrer une date de fin supérieure à l'heure de début";
}
if ((DateFin - DateDeb) > (maxTime * 60000)) {
    success = false;
    result = "Veuillez entrer un créneau inférieur à " + maxTime + " minutes";
}
if (DateDeb < DNow) {
    success = false;
    result = "Veuillez entrer une date et une heure postérieure à maintenant";
}
if (DateDeb - DNow > (maxDelay * 60000 * 24 * 60)) {
    success = false;
    result = "Il est impossible de réserver un babyfoot plus de " + maxDelay + " jours à l'avance, entrez une date plus proche";
}
```

Figure 34 : Test numériques des entrées utilisateurs pour la réservation d'un babyfoot

Une fois que l'on s'est assuré que le créneau était conforme, il faut vérifier qu'il est libre avant de l'enregistrer. Cette opération se fait en testant, sur le babyfoot sélectionné, que pour tous les créneaux déjà enregistrés leur date de début ou de fin ne soient pas contenues dans l'intervalle entrée par l'utilisateur, la requête est :

```
booking.findOne({ ID_baby: bookData.ID_baby, $or: [{ DateDeb: { $gte:
bookData.DateDeb, $lte: bookData.DateFin } }, { DateFin: { $gte: bookData.DateDeb,
$lte: bookData.DateFin } }] }).exec(function(err, Reserv)
```

Si le résultat `Reserv` est `null` donc qu'il n'y a aucune réservation qui est à cheval, on peut passer à l'enregistrement qui se fait comme suit :

```
if (Reserv === null) {
                booking.create({
                        ID_baby: bookData.ID_baby,
                        ID_Joueur: bookData.ID_Joueur,
                        DateDeb: bookData.DateDeb,
                        DateFin: bookData.DateFin,
                    },
```

Dans le cas contraire, l'application informe le joueur que le créneau n'est pas disponible et lui indique de quelle heure à quelle heure est réservé le babyfoot sur le créneau qu'il voulait afin qu'il puisse s'adapter.

La dernière partie importante concernant les réservations réside dans la consultation de toutes les réservations que l'on a faites dans la limite où leur date de fin n'est pas inférieure au moment présent, sinon les réservations sont supprimées pour ne pas encombrer inutilement la base de données. Par manque de temps et de connaissances, la vérification et la suppression sont effectuées à chaque fois qu'un joueur consulte ses réservations plutôt que d'être exécutées régulièrement via du JavaScript stocké. Encore une fois, la récupération et la suppression de réservations se font via des requêtes qui sont plus ou moins similaires à celles que vous avez déjà vu, il ne réside pas un grand intérêt à les détailler. La partie intéressante dans l'affichage des réservations est l'adaptation au contenu. En effet, lorsque qu'un utilisateur demande à voir ses réservations, on ne peut prédire à l'avance combien il va en récupérer, nous ne pouvions donc prédire le nombre de réservations à afficher. Or, il faut bien s'adapter, nous ne pouvions pas prévoir 20 réservations et prendre le risque d'en avoir 19 vides voire même 20 ! Tout comme nous ne pouvions pas prendre le parti d'en afficher que 5 sur 10. C'est à ce moment que Ionic intervient et nous rend la vie bien plus facile, oubliez script, PHP et autres ajouts au fichier HTML. La ligne magique est :

```
<ion-card *ngFor="let stack of reservations">
```

Avec cette ligne et surtout ce qui vient après l'astérisque, le fichier HTML comprend qu'il faut faire un forEach sur le tableau reservations qui est référencé, instancié et peuplé dans le fichier TypeScript. Il ne reste qu'à se soucier de bien remplir le tableau reservations dans le fichier correspondant, Ionic et ses modules s'occupe du reste pour l'affichage. Sachant que ce tableau reservations contient des élements JSON, il ne nous reste plus qu'à appeler les champs de la variable stack dont nous avons besoin pour finaliser notre affichage, exemple :

```
<b> Réservation du  {{ stack.DateDeb }}</b>
```

4. Etude sur la technologie Blockchain pour l'enregistrement des scores

Dans le cadre de notre projet de babyfoot connecté, nous avons pensé qu'il était pertinent d'utiliser une technologie comme la Blockchain pour l'enregistrement des scores. Cela permettrait de garantir que les scores ne seront pas truqués ni effacés, mais aussi de nous familiariser avec cette technologie tout en montrant son utilité aux futurs joueurs.

La Blockchain est une technologie de stockage et de transmission d'informations, transparente, sécurisée et fonctionnant sans organe central de contrôle. En d'autres mots, il s'agit d'une base de données, contenant l'historique complet des échanges effectués depuis sa création, sécurisée grâce à la cryptographie et distribuée sans intermédiaire à tous les utilisateurs, ce qui permet à chacun de vérifier l'intégrité et la validité de la chaîne.

Nous avons réalisé une **étude** présentant dans un premier temps la technologie Blockchain, et dans un second temps comment nous pouvons intégrer cette technologie sur le projet du babyfoot connecté. Cette étude est fournie en Annexe 4, nous vous invitons à aller la lire pour plus d'informations.

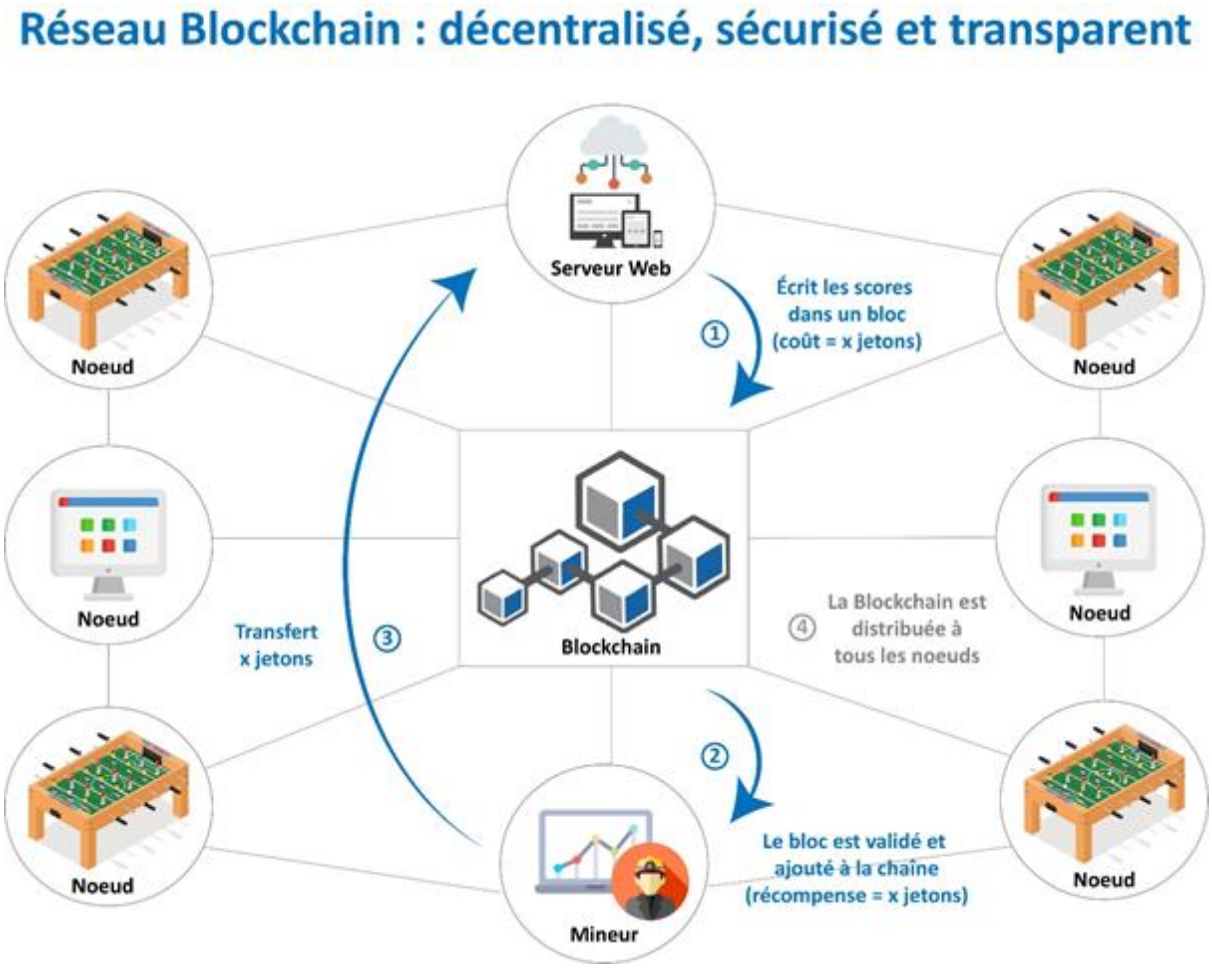


Figure 35 : Schéma de l'utilisation potentielle de la blockchain sur notre projet

Bilan du projet

1. Travail réalisé

Description du prototype final :

Côté Hardware

Nous avons donc un Babyfoot avec 2 capteurs afin de détecter les buts des deux équipes ainsi qu'un écran pour afficher l'application et permettre la création de partie.

Côté Software

Nous avons un Serveur (Backend) en local, une Base de données ainsi qu'une application multiplateforme.

Utilisation côté Babyfoot

Sur l'écran du babyfoot, l'application est lancée et connectée avec un compte de type "babyfoot". Cela permet ces fonctionnalités :

- Création de **partie rapide**, en 10 points avec entre 2 et 4 joueurs
- Création d'un **salon de jeu** que les joueurs peuvent rejoindre à l'aide de leur application. Il est aussi possible d'ajouter des joueurs "invité" et d'exclure des joueurs du salon
- **Lancement de partie** avec l'affichage des scores en temps réel et gestion des scores "à la main" pour modifier les scores en fonctions des règles (gamelles, buts défenseurs, etc.)
- **Enregistrement des parties** dans la base de données

Utilisation côté Joueurs

Le côté Joueur de l'application permet plusieurs fonctionnalités :

- **Création de compte** de type "joueur" dans la base de données
- **Connexion** sur l'application avec son compte créé
- **Déconnexion** de l'application
- Page **Accueil** : permet d'accéder aux différentes fonctionnalités
- Page **Réservation** : permet de réserver un babyfoot pendant une période définie
- Page **Liste Babyfoots** : affiche la liste des babyfoots de l'Université d'Angers
- Page **Profil** : Affiche le profil du joueurs (nom, amis, statistiques, etc…)
- Page **Historique** : historique des parties jouées
- Page **Mes Réservations** : affiche la liste de ses réservations de babyfoot
- Page **Rejoindre une partie** : permet de rejoindre une partie créée à partir du Babyfoot
- Page **Tableau de jeu** : permet de quitter la partie actuelle et d'ajouter un ami au salon de jeu

Lancer le prototype :

1. *Récupérer tous les documents du projet*

Vous pouvez clonez le projet de GitHub : https://github.com/florentyvon/projetbabyfoot. Voici, à droite, ce que vous devez avoir :

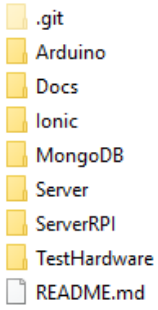


Figure 36 : Arborescence du projet

2. *Raspberry*

Vous devez avoir la Rapsberry pi 3 sous Raspbian avec l'écran tactile. Le Raspberry a déjà les modules Socket.io, Rsub, SerialPort, Npm et Node.

Le Code (*ServerRPI/index.js*) sur la RPI doit être dans */home/pi/ProjetBabyFoot/public_html/index.js.*

Il faut alimenter la Rasbperry avec son chargeur officiel pour ne pas manquer de puissance.

3. *Arduino et capteurs*

L'Arduino doit être branché en USB au Raspberry (Attention que ce soit le bon port USB de la RPI, il est défini dans le code ServerRPI). Les capteurs doivent être reliés à l'Arduino et le code (Arduino/goalCatcher.ino) téléversé.

4. *Environnement de travail*

Vous devez installer MongoDB sur votre ordinateur (voir tutoriel sur internet). Puis lancer le service mondod.exe et MongoDB Compass.

Vous devez installer Ionic, Cordova et Node sur votre ordinateur (voir tutoriel sur internet).

5. *Réseau*

Votre ordinateur et le Raspberry doivent être sur le même réseau local. Utiliser un câble Ethernet et un switch pour les relier. L'adresse IP de la RPI est 192.168.1.10. L'Adresse IP de votre ordinateur doit être 192.168.1.x (x=1,2 ou 3).

Vous pouvez alors vous connectez en SSH à le Raspberry (pi : raspberry) :

- Soit avec Putty
- Soit avec la Machine virtuelle PROG_RPI (mdp : azerty ou Azerty)

6. *Lancement du prototype*

Lancez Ionic sur le serveur local (rep. */Ionic*) : `ionic serve -192.168.1.1`. Ensuite, sur votre navigateur à l'adresse *192.168.1.1:8100*, vous avez accès à l'application. Lancez le serveur Node.JS (rep. /Server) : `node app.js`

Créer un compte Babyfoot dans la BDD, créer un compte joueur avec comme pseudo "BF_ISTIA" puis avec MongoDB Compass, changer l'attribut *typeCompte="joueur"* par ="*babyfoot*".

Connecter vous avec ce compte avec l'écran de la RPI.

Avec votre connection SSH, lancer le serveur de le Raspberry pour lancer la détection des buts : `node /home/pi/ProjetBabyfoot/public_html/index.js` Voilà, vous pouvez jouer.

Lancer toujours dans cet ordre : Ionic (/Ionic), puis le Serveur (/Server), puis le Serveur de le Raspberry.

2. Difficultés rencontrées

Hardware :

Nous avons eu quelques difficultés à réaliser la modélisation 3D des supports de capteurs. Nous n'avons pas beaucoup d'expérience sur les logiciels comme SolidWorks ou OnShape.

Sinon la configuration de le Raspberry nous a pris un peu de temps, notamment la partie communication avec l'Arduino et le serveur.

Software :

Côté Software, les principales difficultés rencontrées ont été sur la prise en main des technologies que nous ne connaissions pas.

Ça a été le cas avec Ionic (TypeScript, AngularJS), nous avons mis du temps à comprendre l'architecture du Framework ainsi que les communications entre l'application Ionic et le serveur.

De même avec le serveur (Node.JS) et communications temps réelles (Socket.io), n'ayant jamais utilisé ces langages nous avons un peu de difficultés à les utiliser.

La base de données MongoDB a aussi posé quelques difficultés au début du développement, sa structure orientée documents et son utilisation sont bien différentes des bases de données classiques que nous connaissions (MySQL principalement).

Cependant après plusieurs heures de recherches, de documentation et de tests, nous avons réussi à utiliser ces technologies pour en tirer profit et développer les fonctionnalités souhaitées.

3. Pistes d'amélioration

Au début du projet, les idées ont fusé et les objectifs étaient plutôt grands. Au moment de conclure ce rapport, voici une liste non exhaustive des améliorations possibles du projet du babyfoot connecté et des solutions pour les mettre en œuvre auxquelles nous avons pensé.

Coté hardware :

- **Détection de gamelle** : beaucoup de joueurs aimeraient que les babyfoots soient équipés de systèmes détectant les gamelles. Pour les non-initiés, la gamelle est une balle qui rentre dans le but, tape le fond et ressort sur le terrain. Dans la pratique générale, cette action vaut la perte d'un point pour le joueur qui l'a subi. Une des solutions serait de mettre un capteur de pression sur la plaque du fond du but et si ce capteur détecte une pression tandis que le capteur de but normal ne détecte pas de balle dans un laps de temps donné, le but soit déclaré gamelle et entraîne la perte d'un point.
- **Support écran sur babyfoot** : Aujourd'hui, nous utilisons un support officiel d'écran de Raspberry qui se pose sur une surface plane plutôt conséquente. Penser à une solution qui permettrait de fixer l'écran

directement sur le babyfoot pour qu'il soit à portée de vue et d'interaction des joueurs serait une solution des plus utiles.

- **Support pour téléphone portable** : Dans une idée de solution toujours plus économique, on pourrait penser à une solution où le téléphone d'un des joueurs servirait d'écran pour tous. Il suffirait juste de fournir un support de portable universel. Cette solution est indissociable de la suivante.
- **Communication Wi-Fi / Bluetooth entre Arduino - Raspberry – Server** : dans un monde toujours plus sans fil mais aussi une logique de simplicité, faire communiquer les différents éléments en Wi-Fi / Bluetooth peut être une solution à envisager. Aujourd'hui, sur notre solution, nous avons dû entailler le babyfoot pour éviter d'écraser le fil et pouvoir fermer tout de même le babyfoot. Avec une solution sans fil, ce serait une tâche et un souci en moins.
- **Motorisation de la libération des balles** : Dans notre projet, on peut réserver un babyfoot pour être sûr de pouvoir jouer dessus aux heures voulues mais le babyfoot reste en libre accès et quelqu'un peut se trouver en train d'y jouer librement sur votre créneau, vous devrez donc lui dire gentiment que ce babyfoot est réservé et qu'il doit s'en aller. Ce n'est pas forcément aisé et ça peut être source de conflit si la personne n'est pas au courant que le babyfoot peut faire l'objet de réservations. Une solution serait de motoriser la libération des balles et de la connecter à la présence du joueur qui a réservé. Le joueur se connecte et lance sa partie, on active le moteur qui libère les balles et la personne est sûre de pouvoir jouer et de ne pas se faire prendre sa place. De plus, la libération des balles pourrait se bloquer à la fin du créneau pour que le joueur ne puisse plus jouer et laisse éventuellement la place aux suivants.
- **Flux en ligne et vidéo des matchs** : Une des idées, il faut le dire utopiste vu notre temps imparti, était de pouvoir installer une ou plusieurs caméras qui filmeraient un match en direct et qui retransmettrait le flux vidéo en ligne pour qu'on puisse suivre un match depuis n'importe où et pourquoi pas revoir un match ou les meilleures actions d'un match : à méditer.

Coté software :

- **Possibilités de connexions diverses** : Dans le livrable de notre projet, il est possible de se connecter à son profil via des identifiants dans notre base de données. Il serait bon dans une optique de diversification et de simplification de la connexion de pouvoir se connecter avec un compte déjà créé comme celui de l'ENT qui est stocké dans l'annuaire LDAP. Nous n'avons pas réussi à mettre ça en place pendant notre projet mais ce n'est pas faute d'avoir essayé. Le manque de temps et de compétences nous en ont sûrement empêché. On pourrait penser à une connexion par un compte Facebook, Twitter, Google+, LinkedIn etc. pour pouvoir tirer les avantages des réseaux sociaux et pouvoir créer des liens entre joueurs après le match, proposer des recommandations d'amis, de liens et proposer la publication de contenu automatique sur le résultat du match.
- **Lancement automatique des serveurs** : Le lancement des serveurs et de l'application sont manuels aujourd'hui, nous n'avons pas réussi à les automatiser tout ou partie mais c'est un axe à développer si de futurs étudiants veulent reprendre le projet.
- **Affichage de calendrier** : Chaque utilisateur a aujourd'hui accès à toutes ses réservations via son profil mais il nous a été proposé de mettre à disposition un calendrier des réservations par babyfoot pour que chacun puisse savoir quelles sont les disponibilités de chaque babyfoot.

- **Gestion Statistiques – Classement - Historique** : Une page « Statistiques » est disponible à l'heure actuelle, elle récupère bien les données statistiques dans la base de données mais malheureusement les statistiques ne sont jamais peuplées ni modifiées après chaque match donc il faudrait penser à une stratégie pour calculer les statistiques et peupler la base de données des statistiques. Cela permettrait de pouvoir gérer les classements de même et cela pousserait à enregistrer les scores des matchs et peupler une base de données des historiques.
- **Interface administrateur** : Deux types de comptes sont aujourd'hui paramétrés, un type joueur et un type babyfoot. Toutes les données sont insérées ou modifier en dur dans les fichiers. Prévoir un type administrateur et une interface dédiée serait un bon moyen de laisser l'application se développer et s'enrichir d'elle-même. Un administrateur pourrait par exemple, supprimer un joueur, ajouter un babyfoot, lancer une compétition etc.
- **Sécurité des comptes** : Dans une future version et des futurs avancements il faudra penser à assurer la sécurité des mots de passes et comptes des utilisateurs pour que le jeu reste sûr. Il est possible que certaines personnes utilisent un mot de passe courant et se le fassent subtiliser**.**

Conclusion

Ce projet de Babyfoot Connecté â été en plusieurs points bénéfique à tous. Premièrement en termes de cohésion d'équipe, nous avons dû apprendre à nous écouter, à prendre en compte l'avis des autres mais aussi aider les autres et leur restituer ce que l'on a appris par nos recherches. C'est un bon moyen de se préparer au stage de fin de 4ème année qui arrive.

Pour mener à bien ce projet et rendre un travail qui nous semble correct, il a fallu que l'on se documente, que l'on cherche des solutions mais aussi que l'on réutilise des connaissances déjà acquises telles que l'usage d'un système Linux, la programmation d'une carte Arduino etc. Il a soit fallu mobiliser des connaissances ou en acquérir de nouvelles, citons par exemple les découvertes de logiciels de programmation (TypeScript), de Framework (Ionic) et un type de base données (NoSQL) très différents de ceux vus en cours, ces dernières seront bien utiles pour nous dans le futur, cela nous apporte beaucoup !

De plus, chacun de nous a choisi ce projet pour des raisons diverses et variées et il est possible d'affirmer aujourd'hui, après les 4 mois de travail, que ce dernier a permis à chacun de confirmer ses choix. Aucun de nous n'a été déçu d'avoir choisi ce sujet et nous avons tous donner notre maximum pour la réussite de ce projet quitte à travailler en dehors des heures réservées aux projets. Chacun de nous est fier du résultat obtenu et serait prêt à continuer à développer le projet et apporter toujours plus de fonctionnalités en EI5 s'il en avait le temps.

Le fait que le projet se déroulait sur un babyfoot qui est un objet de détente et qui peut faire passer notre projet pour non sérieux nous a aussi permis d'aborder le travail sous un autre angle. Il n'était pas rare que des camarades ou professeurs viennent jouer au babyfoot tout en nous servant de bêta-testeurs. Les relations avec ces personnes ont été différentes.

Enfin le fait que notre projet soit commandité par l'Université d'Angers, qu'il suscite la curiosité et l'enthousiasme de beaucoup là-bas, mais surtout que le résultat final plaise tant à M. LHOMMEAU notre tuteur qu'à M. BORDET le commanditaire nous permet de nous rassurer quant à nos méthodes de travail et de gestion de projet pour les stages et embauches à venir.

Bibliographie

Documentation :

- Ionic : https://ionicframework.com/docs/
- MongoDB : https://docs.mongodb.com/
- TypeScript : http://www.typescriptlang.org/docs/home.html
- Angular : https://angular.io/docs
- Socket.io : https://socket.io/docs
- Nodejs : https://nodejs.org/en/docs/
- Express : http://expressjs.com/fr/guide/routing.html
- Onshape : https://www.onshape.com/ (modélisation 3D en ligne)
- Lien Github : https://github.com/florentyvon/projetbabyfoot

Tutoriels – Sites Web :

- MORONY, Josh, Dernière modification le 25 janvier 2018, Building a Review App with Ionic 2, MongoDB & Node. Disponible sur : https://www.joshmorony.com/building-a-review-app-with-ionic-2-mongodb-node/
- MORONY, Josh, Dernière modification le 25 janvier 2018, Building a Hotel Booking App with Ionic 2, MongoDB & Node. Disponible sur : https://www.joshmorony.com/building-a-hotel-booking-app-with-ionic-2-mongodb-node/
- MongoDB, Consulté en février 2018, Install MongoDB Community Edition on Windows. Disponible sur : https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/
- CASTELLOTTI, Jean-Jacques, Publié le 4 novembre 2016, Raspberry Pi – Raspbian pixel et Chromium en mode kiosk – ouverture automatique d'un navigateur en mode plein écran au démarrage de Raspbian. Disponible sur : https://www.jjtronics.com/wordpress/2016/11/04/raspberry-pi-raspbian-pixel-et-chromium-en-mode-kiosk-ouverture-automatique-dun-navigateur-en-mode-plein-ecran-au-demarrage-de-raspbian/

Ouvrages :

- SARRION, Éric, 2014, Programmation avec Node.js, Express.js et MongoDB, Eyrolles, 586 pages, Collection Noire. Lien vers le site éditeur
- RAVULAVARU, Arvind, 2017, Learning Ionic, Packt, 365 pages. Lien vers le site éditeur

Table des illustrations

Annexe 1 : Fiche spécifications du projet

Le baby foot connecté

Encadrants : Laurent Bordet et Mehdi Lhommeau
Nombre d'étudiants : 3-4

Présentation du projet

Le projet consiste à "instrumenter" un baby foot afin de le rendre "connecté". On souhaite enrichir une table de baby foot de fonctionnalités comme la détection automatique des buts (et pourquoi pas des gamelles), l'affichage automatique des scores, ... L'ensemble des ces informations sera transmis à un serveur et alimentera, automatiquement, une base de données. On imagine un réseau social du baby foot interne à l'université permettant aux joueurs (équipes) d'accéder à leurs statistiques, leurs classements et leurs trophées gagnés, et surtout permettant aux équipes/joueurs (étudiants/personnels) de rester en contact après la partie. Le dispositif numérique qui équipera le baby foot devra être le moins encombrant possible et très simple à mettre en œuvre. L'application cliente devra être pensée pour une utilisation la plus large possible (débutants, experts, ...).

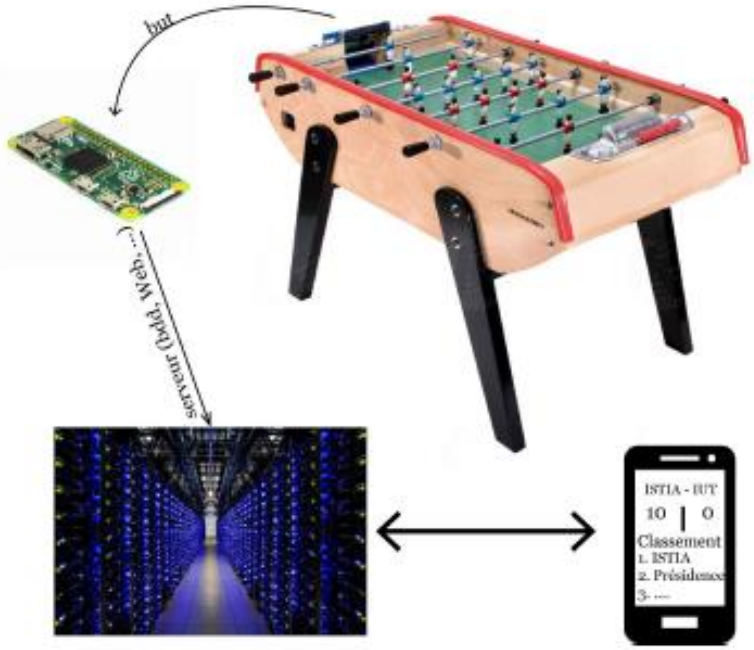


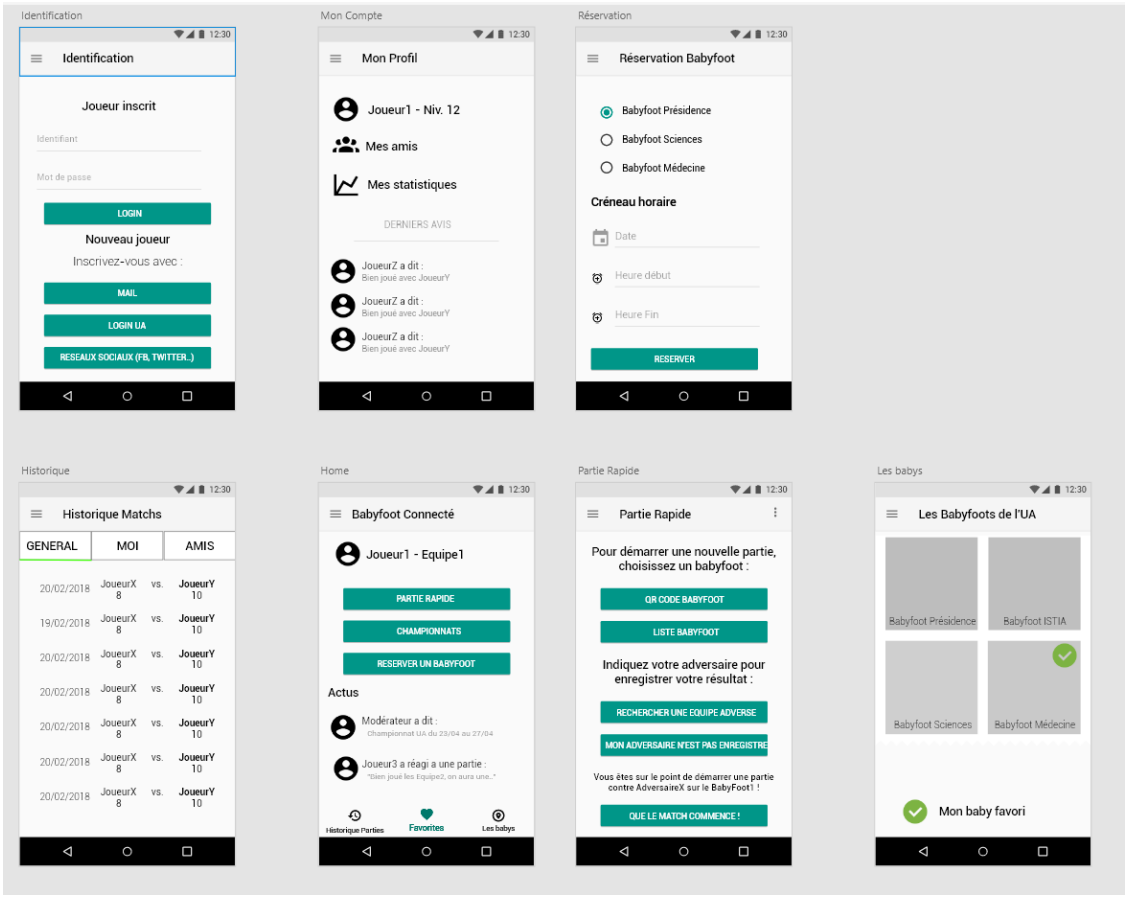
FIGURE 1 – Architecture du projet

La plate-forme BlueMix proposée par IBM fera partie des solutions technologiques à envisager pour la mise en place de ce projet au regard du retour d'expérience d'un projet similaire développé en 2015 par IBM[1]. Le projet devra être documenté pour pouvoir être reproduit au sein de L'Université qui s'équipera dans le courant 2018 d'une série de baby-foot déployés sur les différents campus. A ce jour, un baby foot est disponible à l'UFR Santé et un second devrait être disponible à la fin de l'année à la présidence de l'Université. Ce projet pourra être valorisé lors de la 2nde édition du BabyFoot Challenge, compétition inter-entreprise qui se déroulera au printemps au Quai.
En fonction de l'avancement du projet, il pourrait être intéressant de regarder la technologie Blockchain pour l'enregistrement des scores, des joueurs, ...

Mots clés : Rasbperry PI ZERO, NodeJS, Javascript, ...

1. Il faudra se renseigner auprès de IBM pour avoir un accès

Annexe 2 : Maquette application



Annexe 3 : Code créneaux horaires réservation d'un babyfoot

```
/*Construction Date - Il faut ajouter un 0 si le chiffre est in inférieur à 10*/
if (this.now.getMonth() + 1 < 10) {
  this.event.month = `${this.now.getFullYear()}-0${this.now.getMonth() + 1}`;
} else {
  this.event.month = `${this.now.getFullYear()}-${this.now.getMonth() + 1}`;
}
if (this.now.getDate() < 10) {
  this.event.month += `-0${this.now.getDate()}`;
} else {
  this.event.month += `-${this.now.getDate()}`;
}

/*Construction créneau - Il faut ajouter un 0 si le chiffre est in inférieur à 10**/
if (this.now.getHours() < 10) {
  this.event.timeStarts = `0${this.now.getHours()}`;
} else {
  this.event.timeStarts = `${this.now.getHours()}`;
}
if (this.now.getMinutes() < 10) {
  this.event.timeStarts += `:0${this.now.getMinutes()}`;
} else {
  this.event.timeStarts += `:${this.now.getMinutes()}`;
}
/*On teste si l'heure de fin ne dépassera pas sur l'heure d'après*/
if (this.now.getMinutes() + 10 > 59) {
  /*Si oui et que l'heure est inférieure à 9 on met un 0 comme plus haut*/
  if (this.now.getHours() < 9) {
    this.event.timeEnds = `0${this.now.getHours()+1}`;
  } else {
    this.event.timeEnds = `${this.now.getHours()+1}`;
  }
  /*Il faut remettre à 00 si il est 23h*/
  if (this.now.getHours() == 23) {
    this.event.timeEnds = `00`;
  }
  /*On ajoute 10 minutes modulo 60*/
  this.event.timeEnds += `:0${(this.now.getMinutes() + 10) % 60}`
} else {
  /*Si l'heure de fin ne dépasse pas mais qu'elle est inférieure à 10*/
  if (this.now.getHours() < 9) {
    this.event.timeEnds = `0${this.now.getHours()}`;
  } else {
    this.event.timeEnds = `${this.now.getHours()}`;
  }
  this.event.timeEnds += `:${this.now.getMinutes() + 10}`
}
```

Annexe 4 : Etude sur la blockchain

Introduction

Dans le cadre de notre projet de babyfoot connecté, nous avons pensé qu'il était pertinent d'utiliser une technologie comme la Blockchain pour l'enregistrement des scores. Dans un premier temps, il est important de comprendre ce qu'est la technologie Blockchain, comment cela fonctionne et en quoi c'est utile. Dans un deuxième temps, nous présenterons notre étude de la Blockchain sur le projet du babyfoot.

1. La technologie Blockchain

1.1. Définition

La technologie Blockchain est née en 2008 avec la monnaie virtuelle Bitcoin, créé par un inconnu nommé Satoshi Nakamoto, qui permet l'échange d'actifs monétaires entre utilisateurs sans tiers de contrôle (banques) et avec des frais minimes.

Une Blockchain est une technologie de stockage et de transmission d'informations, transparente, sécurisée et fonctionnant sans organe central de contrôle. En d'autres mots, il s'agit d'une base de données, contenant l'historique complet des échanges effectués depuis sa création, sécurisée grâce à la cryptographie et distribuée sans intermédiaire à tous les utilisateurs, ce qui permet à chacun de vérifier l'intégrité et la validité de la chaîne. Les cas d'usages sont nombreux comme le transfert d'actifs (monétaires ou non), l'horodatage, la traçabilité, l'identification, etc…

Il faut s'imaginer « un très grand cahier, que tout le monde peut lire librement et gratuitement, sur lequel tout le monde peut écrire, mais qui est impossible à effacer et indestructible. »

Il existe des blockchains publiques, accessibles à tous, mais aussi des blockchains privées dont l'accès est limité à un certain nombre d'acteurs, ce qui peut poser des questions sur le caractère décentralisé.

1.2. Le potentiel

Le potentiel de la technologie Blockchain est immense et ses applications vont, dans les prochaines années, changer la manière de fonctionner dans de nombreux domaines, notamment grâce à ses particularités : **Décentralisée, sécurisée et transparente**. Elle pourrait remplacer la plupart des « tiers de confiance » (banque, notaire, Uber, etc…) par des systèmes distribués. Dans un idéal, la gouvernance serait alors totalement distribuée et non pas entre les mains d'une petite poignée de personnes.

Elle permet de garantir la confiance entre deux individus et de permettre des échanges sans l'aide d'un tiers. De plus, Les données inscrites sur une blockchain sont protégées contre la modification, la falsification et la suppression.

On peut alors distinguer plusieurs types d'utilisation :

- Des applications pour les transferts d'actifs (monnaie, acte de propriété, actions d'entreprise) ; cryptomonnaies.

- Des applications en tant que registre ; stocker toutes les données nécessaires sur une blockchain pour garantir leur intégrité et donc améliorer la traçabilité de produits ou d'actifs.

- Les smart-contracts : programmes autonomes stockés sur une blockchain, qui s'exécute automatiquement sans intervention humaine.

- Les ICO (Initial Coin Offering) : méthode de levée de fond fonctionnant via l'émission d'actifs numérique (jeton, token, cryptomonnaie).

1.3. Fonctionnement technique

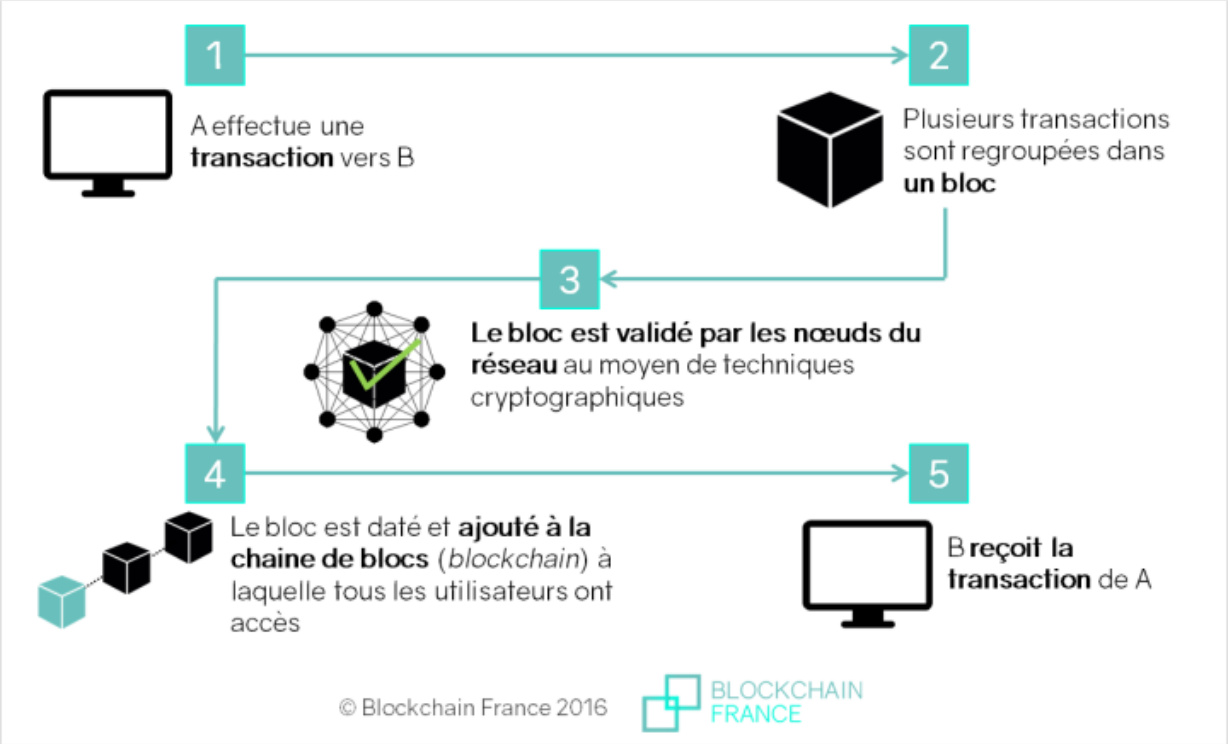
Une blockchain fonctionne sur un **réseau**, formé par des nœuds (aussi appelé « **mineurs** ») sur lesquels elle est déployée. Chaque nœud contient la blockchain complète, c'est-à-dire l'historique de toutes les transactions depuis sa création.

Lorsqu'un utilisateur A effectue une transaction à un utilisateur B sur un réseau Blockchain, cette transaction est regroupée avec d'autres dans un **bloc.** Le bloc est ensuite validé par les **mineurs** grâce à des algorithmes complexes (de type Proof-of-Work, Proof-of-stake, etc…) ; ils vérifient l'intégrité des blocs de la chaîne depuis leur création. On appelle cette action le **minage**.

Une fois le bloc validé par un ou plusieurs mineurs, il est horodaté et ajouté à la chaine de blocs dont tout le monde à accès. En contrepartie, le(s) mineur(s) qui ont validé le bloc, et donc dépensé de la puissance de calcul, sont récompensés par des jetons (tokens ou cryptomonnaies).

Ces jetons représentent une véritable valeur selon les blockchains ; on parle de cryptomonnaies. Ils peuvent être échangés, vendus ou utilisés pour réaliser des opérations sur la Blockchain et leurs valeurs sont sujets à une forte volatilité, due à l'autorégulation entre l'offre et la demande. (Par exemple, au 13/02/2018, 1 Bitcoin = 7000euros, 1 Ether = 684euros, 1 Litecoin = 128euros). Pour finir l'utilisateur B reçoit la transaction de l'utilisateur A.

Ce processus prend un certain temps en fonction de l'algorithme utilisé.



1.4. Les acteurs

Depuis la création du Bitcoin, de nombreuses blockchains publiques sont nées, avec chacune des spécificités, raisons d'être et philosophies différentes. La plupart sont Open Source et chacune ont une communauté (mineurs), qui vote pour les choix technologiques à mettre ou non en place. En cas de désaccord au sein de la communauté, des forks peuvent se produire donnant naissance à une nouvelle Blockchain.

Voici les projets les plus populaires basés sur Blockchain :

- Bitcoin : essentiellement réservé aux transferts monétaires
- Ethereum : création de smart-contracts et outils de développement d'applications décentralisés (dApps)
- IOTA : blockchain à destination des objets connectés (IoT)
- Litecoin : destiné aux paiements entre particuliers
- Ripple : systèmes de paiement destiné aux entreprises
- StorJ : Cloud décentralisé
- IPFS : protocole décentralisé pour concurrencer http
  Et beaucoup d'autres

2. Etude pour le projet du babyfoot connecté

2.1. Quelle utilité pour ce projet ?

Pour notre projet de babyfoot connecté, l'objectif est d'enregistrer l'historique de toutes les parties de babyfoot qui seront jouées sur une blockchain. Cela permettrait de garantir que les scores ne seront pas truqués ni effacés, mais aussi de nous familiariser avec cette technologie tout en montrant son utilité aux futurs joueurs.

2.2. Quelle technologie choisir ?

Pour réaliser cela, plusieurs choix sont possibles en fonction du type de blockchain ;

Blockchain publique :

- Sur un réseau public
- Protocole géré par la communauté
- Totalement décentralisée, aucun contrôle des utilisateurs
- Accessible à tous
- Valeur des jetons (tokens, cryptomonnaies) volatile
- Chaque transaction effectuée sur la Blockchain a un coût (payé en jetons)

Blockchain privée :

- Sur un réseau privé
- Protocole géré par un ou des acteurs définis (gouvernance centralisée)
- Beaucoup utilisé par des entreprises qui veulent garder un contrôle (Banques, assurances, etc…)
- Frais lors des transactions minimes

Dans notre cas, enregistrer les scores sur une Blockchain publique n'a pas forcément d'intérêt et engendrait des coûts (cryptomonnaie à acheter).

Nous nous sommes donc renseignés sur comment créer une Blockchain privée et deux technologies nous ont intéressées ; les plus populaires dans le monde professionnel et les plus développées.

IBM Blockchain :

Basée sur le projet open-source HyperLedger de la Fondation Linux et financée par plusieurs grandes entreprises, la plateforme d'IBM permet la conception de réseaux et d'applications blockchain. estinés principalement aux entreprises et la tarification est assez élevée.

Ethereum :

La technologie Ethereum permet elle aussi de créer des blockchains privées ainsi que des applications décentralisées avec des smart contracts. Elle dispose d'un réseau public (concurrent de celui du Bitcoin), dont les jetons (Ether) pour réaliser des transactions ont atteint récemment une valeur de 1000Euros.

Les avantages de cette technologie, c'est qu'il est possible de créer son propre réseau blockchain privé gratuitement, et de développer ses applications directement sur celui-ci ; le protocole utilisé pour un réseau privé est le même que celui du réseau public.

Cette technologie est donc bien adaptée pour réaliser un réseau blockchain privé et d'enregistrer les scores des parties dessus. La documentation se trouve assez facilement et c'est plus formateur de créer son réseau de A à Z pour mieux comprendre son fonctionnement.

2.3. Mise en œuvre

Architecture globale du réseau

Si tous se passe bien, notre projet de babyfoot connecté sera à l'avenir déployée sur 4 Babyfoots, tournant sous Raspberry PI, avec une application Web unique. Celle-ci sera alors hébergé sur un serveur web.

Pour exister, un réseau blockchain à besoin d'au moins 2 nœuds connectés en permanence (**full node**), dont au moins un ordinateur qui a une puissance de calcul suffisante pour pouvoir miner les blocs et valider les transactions (**mineur**).

Le **serveur web**, sera aussi connecté au réseau blockchain, et se contentera d'écrire les scores des parties de babyfoot sur la blocs chaines. **Les babyfoots sous Raspberry Pi** seront des nœuds du réseau.

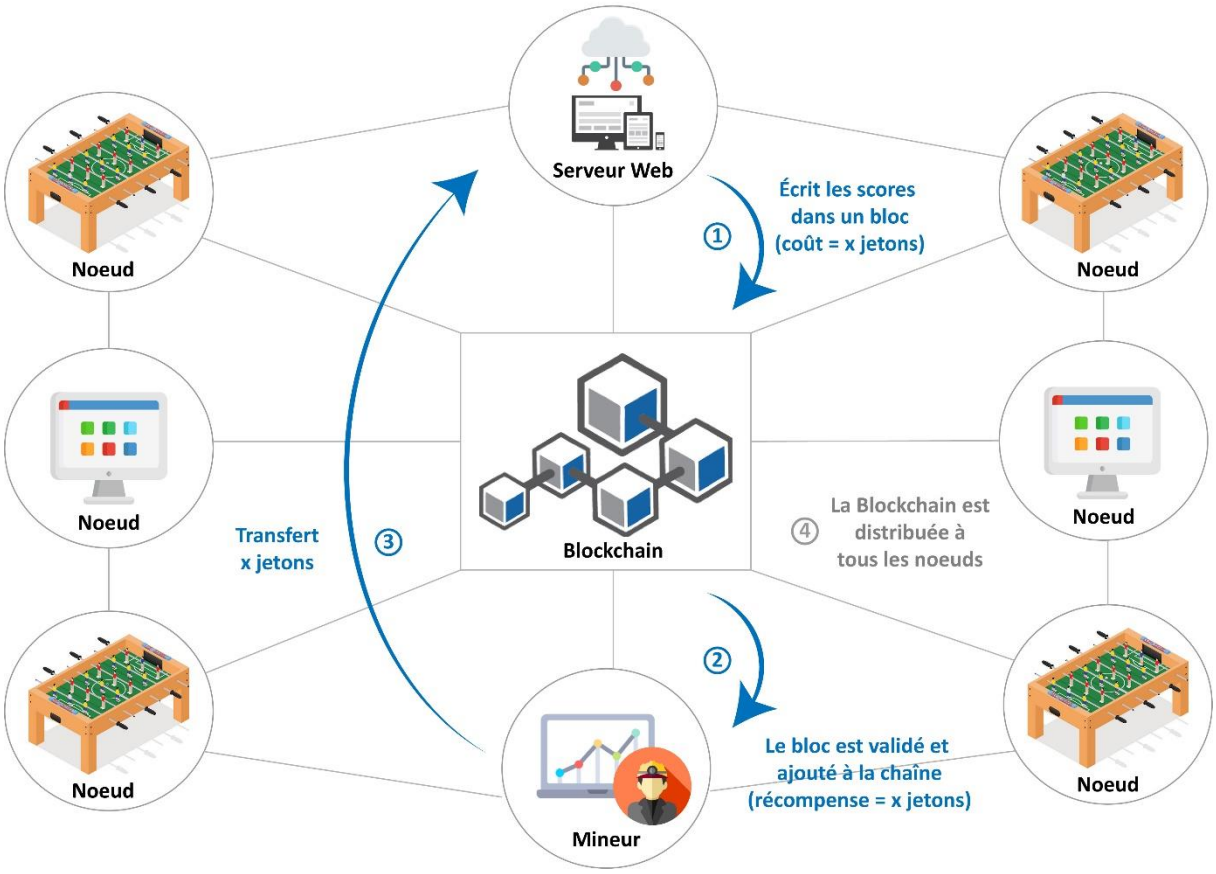
La blockchain sera déployée sur les full-nodes, le serveur-web et les babyfoots. Le **mineur** validera les blocs avec une certaine fréquence, il recevra en récompense des **jetons**. Il transfèrera ses jetons au **serveur web**, qui lui les utilisera pour faire ses **transactions** (écriture des scores sur la blockchain) et récompenser le mineur qui les valide.

Donc voici de ce dont on a besoin pour réaliser cela :

- 2 à n full-nodes* : ordinateurs / serveurs toujours connectés au réseau
- 1 à n mineurs : ordinateur avec une puissance de calcul suffisante pour valider les blocs de transactions
- 1 serveur web : qui héberge l'application et réalise les transactions (écriture des scores sur la blockchain)
- 1 à n babyfoot : Raspberry PI qui se connecte au réseau, ce sont des nœuds du réseau.

*Plus il y aura de nœuds, plus le réseau sera décentralisé et plus la blockchain sera indestructible et sécurisée.

Chaque nœud du réseau stockera et aura accès à la même version de la Blockchain, qui contiendra l'historique de toutes les parties depuis sa création ; cet historique sera alors non modifiable et authentique.



Développement

Au cours de notre projet, nous nous sommes surtout concentrés sur la conception du babyfoot connecté, c'est-à-dire au développement de l'application sous Ionic, du babyfoot et des communications entre eux. Le temps passe vite et nous n'avons pas eu le temps de mettre en place l'enregistrement des scores sur Blockchain. Cependant, d'après nos recherches, voici les principales étapes qu'il aurait fallu réaliser. Pour plus de détail, il faut se référer au tutoriels complets (cf bibliographie).

1. <u>Mettre en place une blockchain privée</u>

Le protocole Ethereum fonctionne avec une interface appelée Geth (Go implementation of Ethereum protocol) écrit en Go. Il faut donc l'installer sur ordinateur (full-node) qui sera toujours connecté.

Grâce à Geth, on peut créer 2 nœuds de réseau qui vont interagir entre eux. Chaque nœud contiendra une base de données et un portefeuille virtuel pour y stocker les jetons.

Ensuite il faut déclarer le premier bloc de notre chaine qui permettra de l'initialiser. Il s'appelle de bloc de Genesis et contient les informations qui définissent la configuration de notre réseau. Voici un exemple de bloc de Genesis.

// genesis.json

```
{
 "alloc": {
   "0xca843569e3427144cead5e4d5999a3d0ccf92b8e": {
    "balance": "1000000000000000000000000000"
   },
   "0x0fbdc686b912d7722dc86510934589e0aaf3b55a": {
    "balance": "1000000000000000000000000000"
   }
 },
 "config": {
  "chainID": 68,
  "homesteadBlock": 0,
  "eip155Block": 0,
  "eip158Block": 0
 },
 "nonce": "0x0000000000000000",
 "difficulty": "0x0400",
 "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
 "coinbase": "0x0000000000000000000000000000000000000000",
 "timestamp": "0x00",
 "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
 "extraData": "0x43a3dfdb4j343b428c638c19837004b5ed33adb3db69cbdb7a38e1e50b1b82fa",
 "gasLimit": "0xffffffff"
}
```


- « alloc » : permet de lié la chaine à un ou plusieurs nœuds
- « config » : configuration de la chaine, avec un id de sécurité
- « nonce » : permet de vérifier cryptographiquement que le bloc a été miné
- « difficulty » : difficulté de validation des transactions (minage). Plus la valeur est basse, plus il est facile de miner et plus les transactions seront rapides (mais diminue la sécurité)
- « mixhash » : hash utile à l'algorithme Proof-of-Work pour valider les blocs
- "parentHash" : contient l'id du bloc parent (nonce + mixhash). 0 pour le bloc de Genesis
- « gasLimit » : définie une limite de gas. Gas : unité de coût d'une transaction ⇔ puissance de calcul qu'elle utilise lors de la validation de blocs.

Ensuite il faut initialiser la chaine à l'aide de Geth : *geth init genesis.json*

Il faut répéter cette opération dans les deux nœuds créés, ils seront chacun initialisé avec le même bloc de Genesis.

2. Créer des utilisateurs pour chaque « nœud »

Chaque nœud doit être lié avec un compte « mineur », qui recevra alors des jetons à chaque bloc validé. Il faut renseigner un mot de passe et une adresse sera généré

geth --datadir ~/Repertory/miner1 account new

⇨ Address: {**3e3753727dd6d965c0c696ea5619b8050ca89a49**}

On peut ensuite miner avec ces comptes. Voici à quoi ressemble une ligne de commande pour lancer le minage :

computer$ ***geth --identity "miner1" --networkid 42 --datadir "~/Repertory/miner1" --nodiscover --mine --rpc --rpcport "8042" --port "30303" --unlock 0 --password ~/Repertory/miner1/password.sec --ipcpath "~/Library/Ethereum/geth.ipc"***

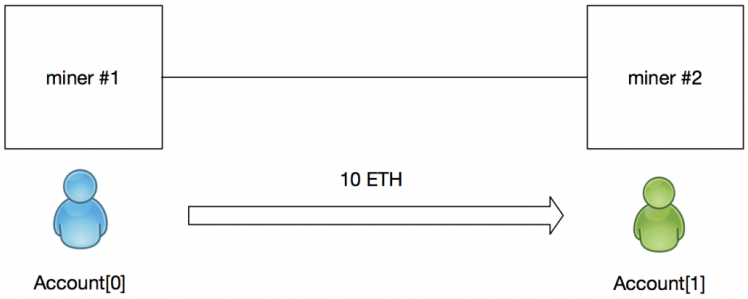
Ou utiliser la console Javascript de Geth :

computer$ **geth attach**

…

> **miner.start()**
...
> **miner.stop()**
...

Une fois un bloc miné par un utilisateur (nœud), celui-ci reçoit des jetons « ether » dans son portefeuille virtuel, qu'il peut transférer à d'autres utilisateurs ou portefeuille.



3. Synchroniser les mineurs

Il faut ensuite synchroniser les mineurs sur le même réseau, toujours avec Geth (geth attach ipc:/data) , en renseignant dans un fichier les informations (ip, adresse) des mineurs. Puis vérifier la bonne synchronisation en envoyant des jetons entre mineurs (***eth.sendTransaction()*** **).**

4. Installer Geth sur Raspberry et la configurer comme nœud

Les Raspberry des babyfoots peuvent être configurer comme nœud connecté à notre réseau précédemment mis en place. Il faut tout d'abord d'installer Geth sur Raspbian, en choisissant une version compatible avec le processeur).

Comme précédemment, on utilise le même fichier Genesis.json pour initialiser le nœud. Puis on créer un compte lié à ce nœud, que l'on synchronise avec les mineurs de notre réseau. Après cela, notre Raspberry est connecté à notre réseau avec le protocole Ethereum.

5. Développer notre application décentralisée (dApp)

Une fois notre réseau mis en place, il reste à développer une application permettant d'enregistrer les scores des parties de babyfoot sur notre blockchain.

Pour interagir avec une blockchain, le protocole utilise des smart contracts (contrats intelligents). Ce sont des programmes informatiques, qui comme des contrats traditionnels, vérifie, garantissent et exécutent automatiquement des conditions définis (cf. bibliographie).

Dans notre cas, ces contrats vont devoir assurer le stockage des scores.

Le développement est assez technique, et se fait généralement avec des Framework pour Ethereum comme Truffle ou Solidity (le plus utilisé). Une fois les fonctions développées, il faut ensuite déployer le contrat et le tester sur la blockchain.

Un smart-contract contient des fonctions, ici il recevra en entrée les données de parties de babyfoot, envoyées par le serveur web (équipes, joueurs, scores, etc..), contenues dans un bloc. Lorsque le bloc sera validé par les mineurs, les utilisateurs du réseau pourront avoir accès aux données du blocs (historique des parties de babyfoot) et le contrat déclenchera l'envoie de jeton automatique au serveur web.

Il est aussi possible de réaliser une interface utilisateur pour permettre la consultation des données simplifiées pour les utilisateurs, car il ne faut pas oublier qu'une blockchain doit rester transparente.

3. Conclusion

Grâce à sa décentralisation et sa sécurité, la technologie Blockchain est prometteuse dans de nombreux cas d'utilisation, notamment pour garantir l'intégrité de données ou encore la traçabilité.
Dans le cas de notre projet de babyfoot connecté, il serait intéressant de mettre en place une Blockchain pour l'enregistrement des scores pour faire un « état de l'art » de la technologie et de comprendre son utilité et fonctionnement.
Cependant malgré nos recherches, le développement d'applications utilisant les smart-contracts pour interagir avec une Blockchain reste assez flou et complexe.
Cette étude nous a surtout permis d'aborder la technologie d'un point de vue théorique, durant les temps de projet nous avons surtout travaillé sur l'application web / mobile et la partie hardware du babyfoot connecté.

4. Sources

- **Blockchain France**, *Qu'est-ce que la Blockchain ?*
  https://blockchainfrance.net/

- **Blockchain Partner & LearnAssembly**, *MOOC Blockchain*
  http://www.moocblockchain.com/17/mooc-blockchain-particuliers

- **IBM**, *Blockchain Essentials*
  https://developer.ibm.com/courses/all/blockchain-essentials/

- **Wikipedia**, *Blockchain*
  https://fr.wikipedia.org/wiki/Blockchain

- **ChainSkills**, *Create a private Ethereum blockchain with IoT devices*
  http://chainskills.com/2017/02/24/create-a-private-ethereum-blockchain-with-iot-devices-16/

RÉSUMÉ

Ce rapport s'inscrit dans le cadre du projet de deuxième année cycle ingénieur option systèmes automatisés et génie informatique à l'ISTIA, école d'ingénieur de l'université d'Angers. Ce projet, d'une durée de 80h, a été encadré par M. LHOMMEAU Mehdi et M. BORDET Laurent.

Celui-ci consiste à rendre un **babyfoot connecté** grâce à des **capteurs** fixés sur la table de jeu et qui seront reliés à une **application web/mobile** et ce, dans le but de créer un réseau social autour de ce loisir pour faciliter l'échange entre le personnel de l'Université ainsi qu'entre étudiants et professeurs mais aussi pour favoriser le bien-être au travail.

Le cahier de charges étant assez complexe, le projet est réalisé par une équipe de 3 étudiants. Il a été rendu possible en utilisant essentiellement des **technologies légères** comme **Ionic** et **NodeJS** ainsi qu'un **Raspberry Pi** et une carte **Arduino UNO**. Le but est de créer un kit de déploiement rapide et peu couteux pour connecter tous les babyfoots de l'Université d'Angers.

Ce rapport détaille les méthodes et les technologies utilisés pour mener à bien ce projet. Il fournit des exemples de code et de vues sur l'application pour une meilleure compréhension de cette dernière. Il permet à la fin aussi de voir toutes les possibilités d'amélioration pour rendre le projet encore plus complet attractif. Il fourni aussi un **tutoriel** pour pouvoir installer ce système sur votre propre babyfoot.

mots-clés : Babyfoot connecté, capteurs, application web/mobile, technologies légères, Ionic, NodeJS, Raspberry Pi, Arduino UNO, , tutoriel, blockchain

ABSTRACT

This report is part of the second-year engineering project option automated systems and computer engineering at ISTIA, engineering school of the University of Angers. This 80-hour project was supervised by Mr. LHOMMEAU Mehdi and Mr. BORDET Laurent.

This consists in making a **foosball table connected** thanks to **sensors** fixed on the game table and which will be connected to a **web/mobile application** with the aim of creating a social network around this leisure time to facilitate the exchange between the personnel of the University as well as between students and professors but also to support the well-being at work.

The specifications being rather complex, the project is realized by a team of 3 students. It was made possible using mainly **quick technologies** such as **Ionic** and **NodeJS** as well as a **Raspberry Pi** and an **Arduino UNO** card. The goal is to create a fast and inexpensive deployment kit to connect all the Angers University foosball tables.

This report details the methods and technologies used to carry out this project. It provides sample code and application views for a better understanding of the application. It also allows at the end to see all the possibilities of improvement to make the project even more attractive. It also provides a **tutorial** to be able to install this system on your own foosball table.

keywords : foosball table connected, sensors, web/mobile application, quick technologies, Ionic, NodeJS, Raspberry Pi, Arduino UNO, tutorial, blockchain

Présidence de l'université
40 rue de rennes – BP 73532
49035 Angers cedex
Tél. 02 41 96 23 23 | Fax 02 41 96 23 00

ENGAGEMENT DE NON PLAGIAT

Nous, soussignés Steve DESPRES, Ahmed Youssouf ZIYYAT et Florent YVON
déclarons être pleinement conscients que le plagiat de documents ou d'une
partie d'un document publiée sur toutes formes de support, y compris l'internet,
constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.
En conséquence, nous nous engageons à citer toutes les sources que nous avons utilisées
pour écrire ce rapport ou mémoire.

signé par les étudiants le **13** / **04** / **2018**

Cet engagement de non plagiat doit être signé et joint à tous les rapports, dossiers, mémoires.

Présidence de l'université
40 rue de rennes – BP 73532
49035 Angers cedex
Tél. 02 41 96 23 23 | Fax 02 41 96 23 00