Année 2014-2015

Rapport de projet – EI4 AGI

Système de vote interactif

Projet réalisé par :

BELLAJ MEHDI
LABCHARA OMAR

Projet Encadré par :

Mehdi Lhommeau

ISTIA – Université d'Angers

REMERCIEMENTS

Tout d'abord, nous tenons à remercier tout le corps enseignant de l'ISTIA pour les connaissances qu'ils nous ont apporté, et particulièrement Mr. Mehdi LHOMMEAU notre tuteur, responsable projet, dont l'aide nous a été précieuse tout au long de ce projet pour ses conseils éclairés, sa patience, sa disponibilité et pour la confiance qu'il nous a accordé dès l'ébauche du projet et tout au long de ces cinq mois.

Nous remercions également nos familles et nos amis pour leurs soutiens quotidiens, leur présence lors des coups durs mais aussi lors des moments de joie.

Table des matières

INTRODUCTION

Étudiants à l'ISTIA, l'école d'ingénieur de l'université d'Angers, nous avons dû dans le cadre de notre cursus scolaire, réaliser un projet afin de mettre en pratique l'ensemble des techniques apprises au cours de l'année.

Le projet consiste à réaliser un système de vote interactif permettant de réaliser un sondage des participants d'une assemblée en leur posant des questions, et en recueillant leurs réponses, grâce à une application Web basée sur une carte Raspberry Pi. On peut imaginer utiliser ce système de vote dans un amphi lors d'un cours magistral pour recueillir des réponses à des questions de cours. Le système développé doit être très simple à déployer.

On peut utiliser les téléphones des étudiants comme système de vote dans la limite ou on a accès au site Web mis à la disposition. Le serveur permettant de recueillir les votes sera basé sur la Raspberry Pi.

Ce projet a été proposé et supervisé par note tuteur Mr. Mehdi LHOMMEAU, enseignant-chercheur de l'ISTIA.

I. Présentation du projet :

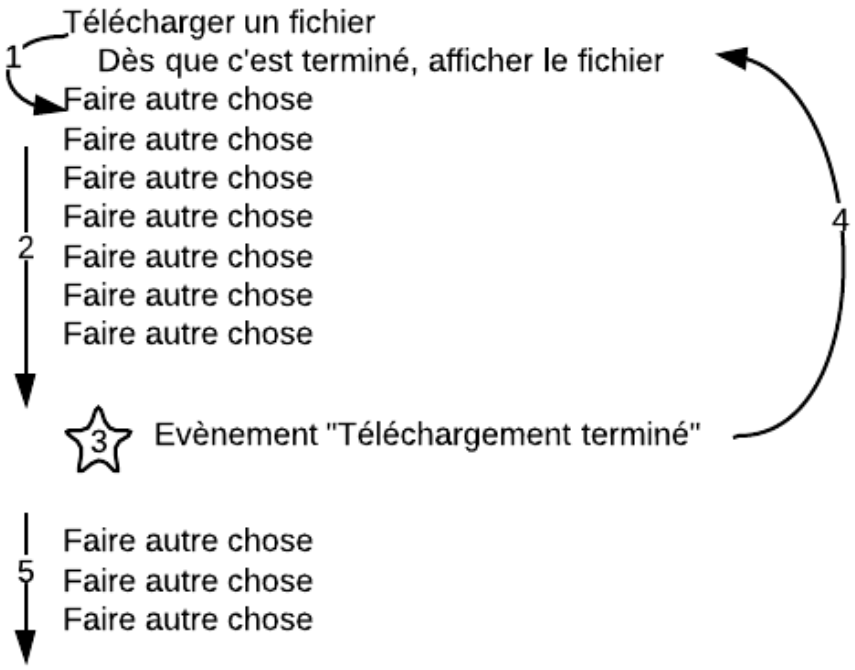
1. Objectifs

Le projet est prévu pour être réalisé à l'aide de Node.js et JADE, un Framework en JavaScript, aux spécificités particulièrement adaptées à ce type de projet et aussi à l'aide de langage de développement Web.

Je me suis d'abord familiarisé avec JavaScript et Node.js. Je me suis basé sur les tutoriaux [1] pour le JavaScript et [2] [3] [4] pour Node.js. J'ai aussi eu fréquemment recours à la documentation Node.js [5] [6].

À l'issue de ce tutoriel, j'ai appris à maitriser les différentes commandes console nécessaires pour faire fonctionner les scripts Node.js. Les premiers scripts effectués, permettaient la communication entre un serveur basique et un client. Au fur et à mesure, je m'intéressais à quelques Template comme Express et Jade qui pouvaient m'aider à mieux mener le projet à terme. Tout en apprenant à récupérer les informations dans la barre d'adresse, à écouter les évènements et à créer des fonctions de callback (fonctions de rappel qui s'exécutent quand un évènement se réalise) je me documentais sur la gestion des routes grâce à Express. Ces fonctions sont à la base même de Node.js. En effet, ces outils permettent d'adopter un modèle non bloquant, c'est-à-dire que le programme n'est pas obligé d'attendre le retour d'un processus particulièrement long. Il peut continuer à s'exécuter et revenir déclencher un évènement une fois le processus long terminé.

Par exemple :



Une fois un premier contact effectué avec ce nouvel environnement, on a décidé d'utiliser la méthode indiquée dans ce tutorial [3] qui nous semble encadrer d'une façon très élégante pour développer notre projet grâce aux outils énoncés auparavant.

Notre projet doit implémenter une communication avec une base de données qui nous permettra de sauvegarder les sondages recueillis et aussi les informations nécessaires pour mettre en place le formulaire.

Il nous a fallu commencer par rechercher quel SGBD (Système de gestion de Base de Données) utiliser avec Node.JS, et MongoDB s'est avéré être un choix pertinent car il permettait une utilisation aisée à l'aide d'un module spécifique prévu à cet effet et dédié à Node.JS

Pour le bon fonctionnement du projet on a décidé de départager les taches pour qu'un de nous se charge de la partie «administrateur» qui est consacrée à l'enseignant pour mettre en œuvre le formulaire et l'autre s'occupera de la partie « client » qui consiste à recueillir les sondages.

2. Présentation des technologies utilisées :

a. Node.JS

Node.js est une plateforme de logicielle libre et événementielle en JavaScript orienté vers les applications réseau qui doivent pouvoir monter en charge.

Elle utilise la machine virtuelle V8 et implémente sous licence MIT les spécifications CommonJS.

Node.js contient une bibliothèque de serveur HTTP intégrée, ce qui rend possible de faire tourner un serveur web sans avoir besoin d'un logiciel externe comme Apache , et permettant de mieux contrôler la façon dont le serveur web fonctionne.

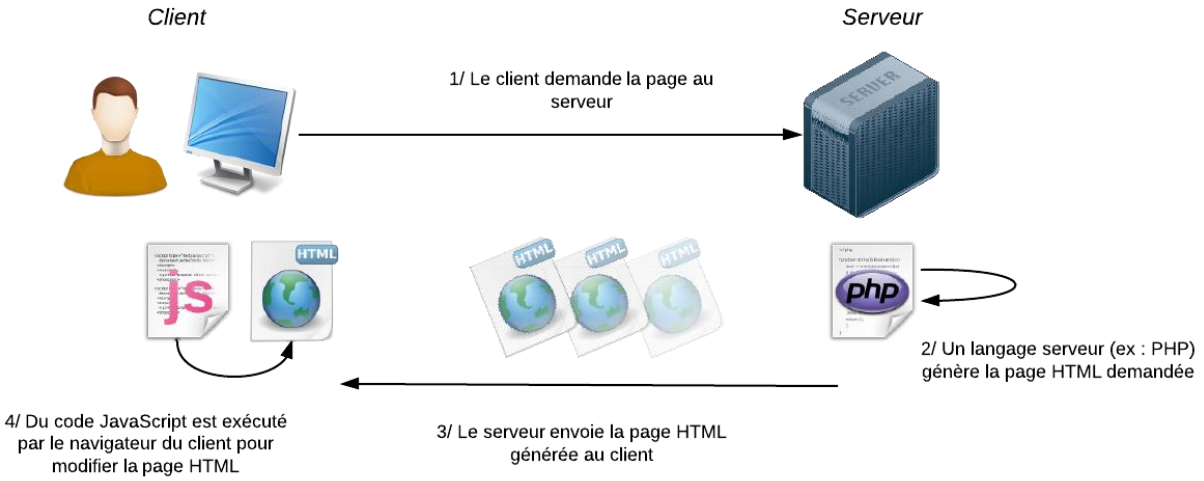


FIGURE 1 : ILLUSTRATION DU FONCTIONNEMENT CLIENT-SERVEUR AVEC NODE.JS (SOURCE : OPENCLASSROOMS.COM)

b. Express

Le Framework Express est un ensemble de modules Node.Js permettant de créer facilement des applications web à partir de Node.Js
Il est basé sur le modèle MVC «Model View Controller» qui permet de donner une architecture cohérente à une application web.

Parmi les modules les plus intéressants d'Express on trouve la gestion des routes qui est une partie importante dans une application car elle gère la façon d'accéder aux ressources, contrairement à PHP qui associe une URL à l'emplacement d'un fichier alors qu'Express dissocie les deux (on peut donc avoir une URL qui est différente du fichier associé).

c. Jade

Jade est un moteur de Template et aussi un langage qui compile au format HTML.

Jade peut s'avérer complexe à cause de ses règles, sa syntaxe, les indentations à respecter, etc. Mais, passer ces petits détails, il nous permet de gagner énormément de temps! De plus, il nous offre, la possibilité d'utiliser des variables Javascripts.

d. Base de donnée

MongoDB est une base de données robuste qui se base sur le système de NoSQL, est orientée documents. L'objectif est donc de pouvoir gérer de grandes quantités de données, ces informations sont modélisées sur un document au format JSON qui est sous la forme suivante.


```
{
   "id": 1,
   "name": "A green door",
   "price": 12.50,
   "tags": ["home", "green"]
}
```

3. Organisation du projet :

Le projet a débuté le 10 décembre 2014 et a duré 4 mois. La figure suivante présente le diagramme de GANTT qui représente le planning qui a été suivi pour mener à bien le projet.

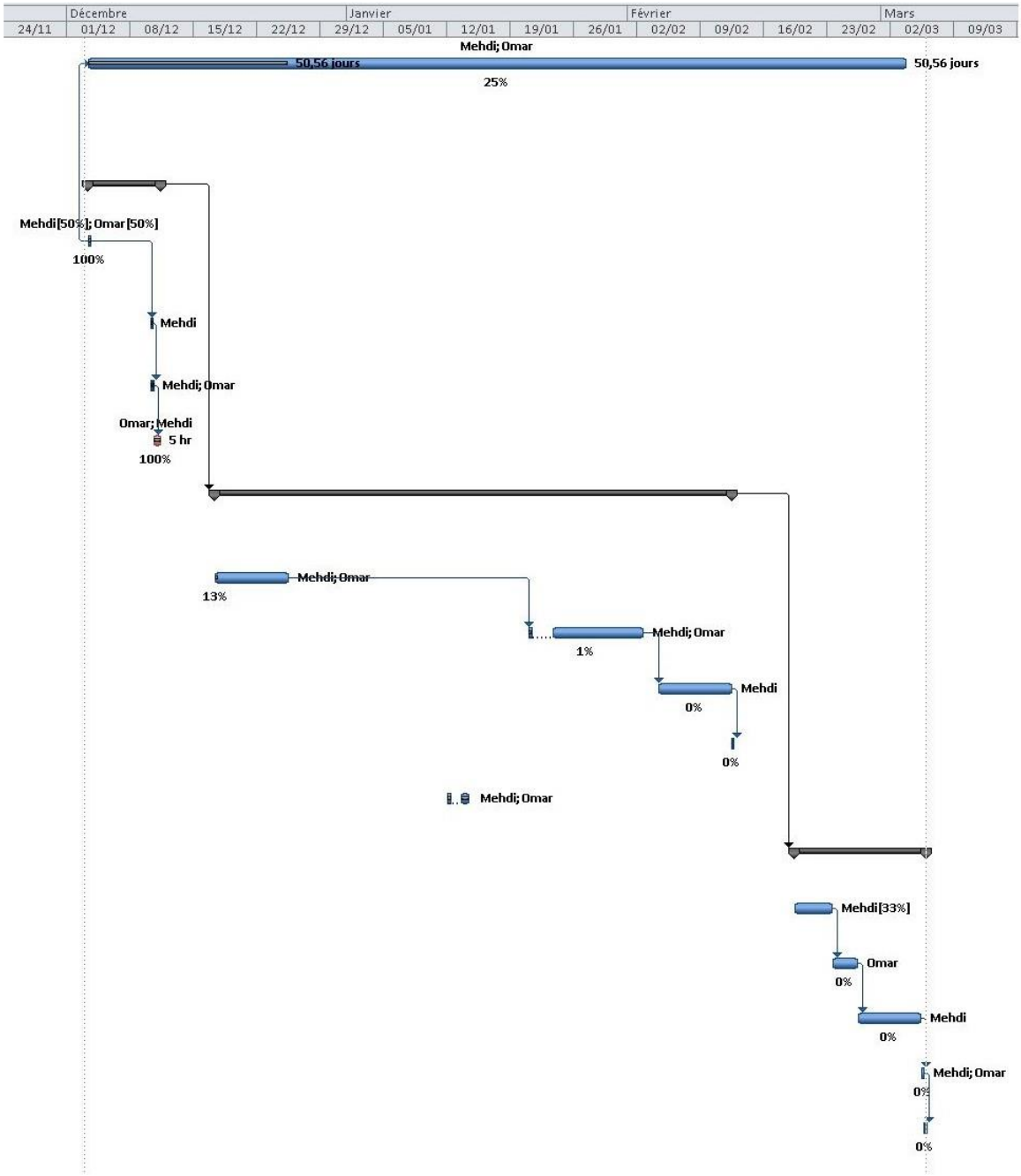


FIGURE 2 : DIAGRAMME DE GRANT

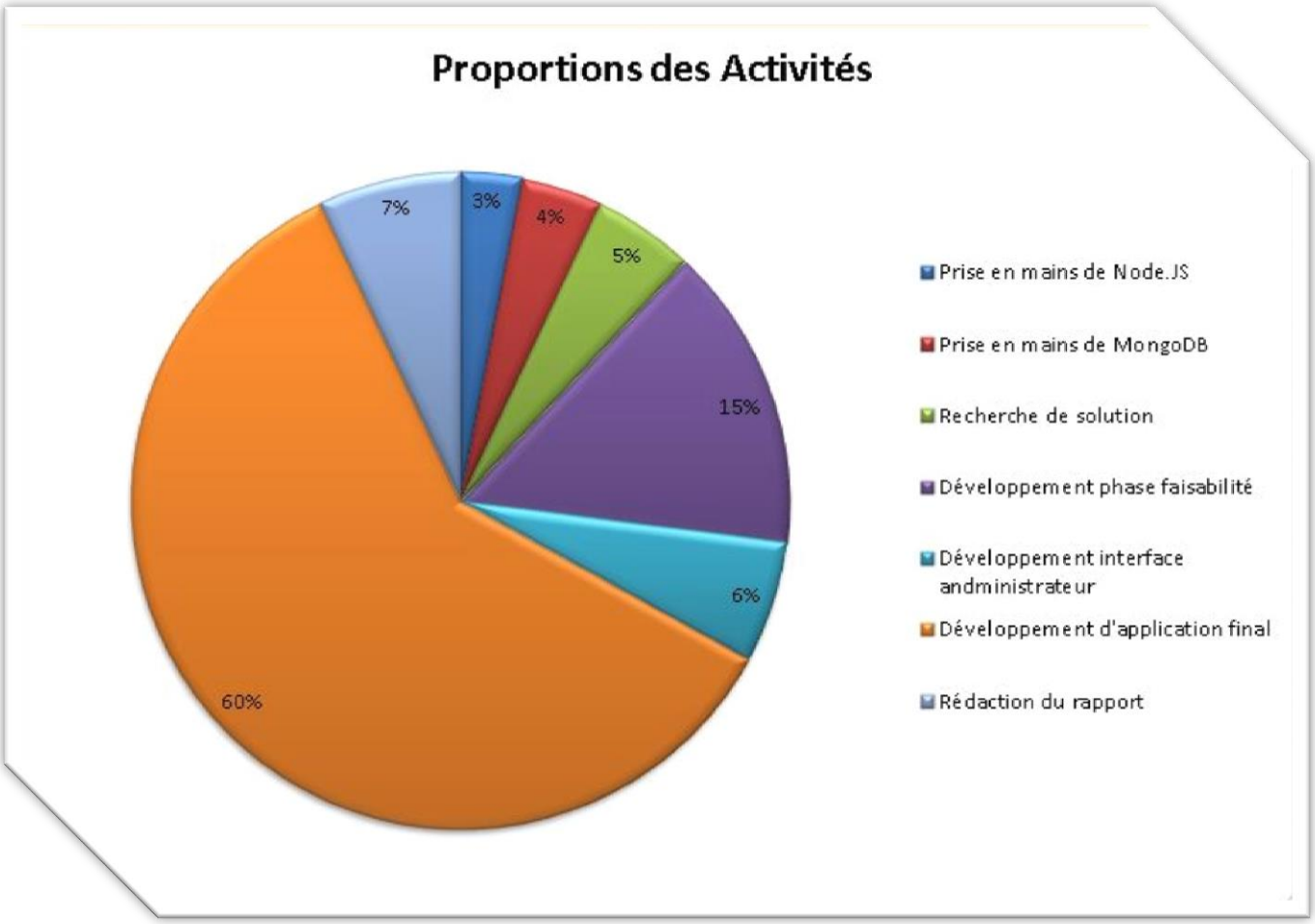


FIGURE 3 : REPARTITION DES DIFFERENTES ACTIVITE

II. Développement technique :

1. Stratégie

Au début du projet on n'avait pas totalement cerné le problème proposé, c'est pour ça que notre phase d'expression de besoin a été dédiée entièrement a la recherche de solutions et de technologie qui réponde à l'attente du cahier des charges.

Une fois qu'on a pu établir le but final du cahier des charges nous avons conclu à ces solutions techniques :

- Mise en place d'un serveur node.JS sur une carte Espruino, connexion en WiFi grâce à une carte CC3000
- Création d'une application mobile permettant la communication Espruino-smartphone via WiFi
- Création d'un boîtier de commande espruino + CC3000
- Création d'un site permettant d'accéder aux données en temps réel avec un ordinateur

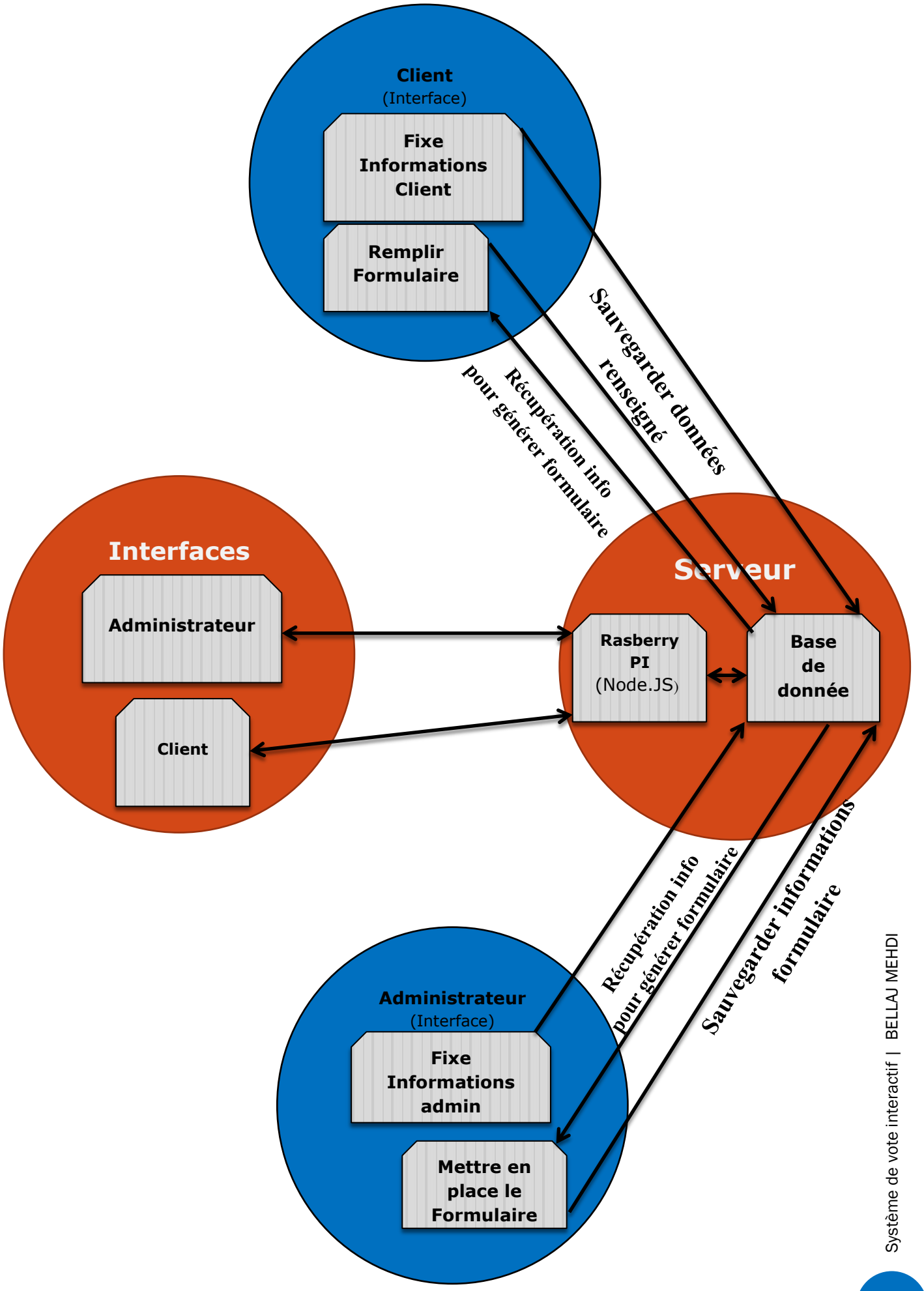
NB : au cours du projet nous et notre tuteur avons décidé d'utiliser une carte Raspberry Pi au lieu d'une carte Espruino.

A la suite de cette phase on a enchainé avec la phase de faisabilité qui consiste à créer une application de test simple du style client-serveur, ensuite on a mis en place une architecture générale qui reflète la structure du projet final ainsi que les taches capitales pour la réussite du projet :

Les Taches capitales :

- Création serveur (Node.js)
- Communication avec le serveur (Node.js)
- Traitement de données (Jade, Node.js, Express)
- Sauvegarde de données (BDD)

Figure 4 : Architecture général



Le serveur:

- Le serveur doit permettre de stocker les données reçues des interfaces Web, qui sont le formulaire créé par l'enseignant et les réponses de l'utilisateur.

- Le serveur doit permettre de renvoyer des données à la demande d'une Interface Web toutes en respectant la hiérarchie administrative.

- Le serveur doit permettre de calcul des scores de chaque formulaire et l'enregistrer dans l'emplacement correspondant.

Les Interfaces Web:

- Le site internet est l'interface utilisée par l'enseignant et l'utilisateur pour proposer ou répondre à un formulaire.

- Interface administrateur doit permettre de créer un questionnaire facilement et pertinent à l'orateur, peu de compétence informatique devront être nécessaire à l'utilisation de cette interface de création du questionnaire. A la fin de cette étape on envoie les données au serveur.

- Interface administrateur doit permettre à l'enseignant de voir les réponses des participants et leurs scores en lui affichant les formulaires remplis en les groupant par nom ou identifiant d'utilisateur. Pour cela il faut demander les informations au serveur.

- Interface client doit permettre à l'utilisateur de sélectionner et remplir le formulaire mis en disposition, avant cela le client doit entrer des informations personnelles pour l'identification (Nom, Prénom...).

- Interface Client doit envoyer à la fin les données du formulaire au serveur pour traitement.

2. Mise en œuvre

a. Pré configuration

Pour débuter le développement de notre projet il nous faut tout d'abord installer Node.js et MongoDB.

Une fois ces logiciels installés on ouvre « Node command prompt » pour l'installation des Framwork qui vont nous permettre d'architecturer notre projet. Pour ce faire on effectue les commandes suivantes:

```
//installation des paquets express

# npm install  -g express
# npm install  -g express-generator

//créer notre projet architecturé grâce à Express
//on se place ou on désire créer notre projet et on indique le nom du projet

# express MonProjet
```

```
//affichage des taches effectué par la commande
create : MonProjet
 create : MonProjet/package.json
create : MonProjet/app.js
create : MonProjet/public/stylesheets
create : MonProjet/public/stylesheets/style.css
create : MonProjet/public/javascripts
create : MonProjet/public
create : MonProjet/public/images
create : MonProjet/routes
create : MonProjet/routes/index.js
create : MonProjet/routes/users.js
create : MonProjet/views
create : MonProjet/views/index.jade
create : MonProjet/views/layout.jade
create : MonProjet/views/error.jade
create : MonProjet/bin
create : MonProjet/bin/www
```

Une fois cette étape franchie il nous reste plus qu'ajouter les dépendances du projet soit manuellement « npm install …» soit en ajoutant cela dans le fichier « package.json » créé par Express comme indiquer dans le tutorial [3] (avec Package.json on utilise juste la commande npm install)

Les principaux fichiers/dossiers de notre projet sont le fichier « app.js » qui configure les routes, le fichier « /bin/www » rassemble les configurations de connexion (Port de connexion, création du serveur…) qu'on ne modifie pas sauf si on veut changer de port d'écoute, le dossier routes qui contient le fichier index.js qui est le script principal de notre projet où pour chaque route on lui associe le traitement correspondant. Enfin, on a le dossier « views » qui regroupe les fichiers jade (page web) qui vont pouvoir rendre les informations visuelles à l'utilisateur.

Maintenant on peut démarrer notre serveur et commencer à développer les routes et les interfaces. Pour le faire on se place dans le dossier « MonProjet » et on utilise la commande :

```
# npm install
# npm start
```

Dans un navigateur on lance localhost:3000 et on obtient :

b. Développement détaillé

Notre fichier index.js contient une route qui ouvre le ficher index.jade

```
router.get('/', function(req, res, next) {
res.render('index', { title: 'Express' });
});
```

Index.js

```
extends layout
block content
  h1= title
  p Welcome to #{title}
```

index.jade

la fonction « router.get() » prend en paramètre la route (/ pour localhost :3000) qui déclenche le traitement correspondant qui est défini dans la fonction callback, cette dernière est envoyée en deuxième paramètre. La fonction callback prend deux ou trois paramètres (soit res,req ou res,req,next). Dans notre cas elle fait appel à la méthode « render() » qui va nous permettre d'indiquer le fichier d'interface à exécuter qui est « index » et qui est situé dans le dossier « views ». La fonction « render() » permet de faire passer en paramètre des variables qui pourront être utilisées dans les fichiers contenus dans le dossier « views », comme la variable « title » qui a comme valeur « express » et ce qui explique le résultat obtenue dans le navigateur web.

Le fichier « index.jade » comporte un appel d'un autre fichier jade « layout » qui contient l'entête d'une page HTML (voir l'image ci-dessous), comme j'ai pu le mentionner auparavant le dossier « views » regroupe les fichiers qui vont pouvoir rendre les informations visuelles à l'utilisateur, la syntaxe de programmation de jade est basé sur le langage HTML à la différence que sur jade on n'utilise pas de balise mais seulement la lettre de la balise HTML et les espaces sont pris dans la syntaxe et peuvent être source d'erreurs, néanmoins on peut utiliser la syntaxe HTML et du Javascript.

```
doctype html
html(lang="fr")
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    block content
```

Le bloque qui contiendra le code du fichier layout

layout.jade

Pour notre interface administrateur j'ai choisi d'utilise les routes suivantes:

```
router.get('/admin', function(req, res) {

router.post('/admin/info', function(req, res) {


router.get('/admin/fill', function(req, res, next) {

router.post('/admin/fill/ok', function(req, res, next) {
```

Ouvrir la page administrateur du dossier views

Recevoir en POST les informations saisies de l'administrateur pour générer le formulaire

Ouvrir la page de saisie du formulaire du dossier views

Envoi des informations saisies à la BDD

- La Première route va faire appel à la fonction « res.render » qui va demander l'affichage d'un des fichiers du dossier « views » :

```
res.render('AdminInfo', { title: 'Creation de Questionnaire!' });
```

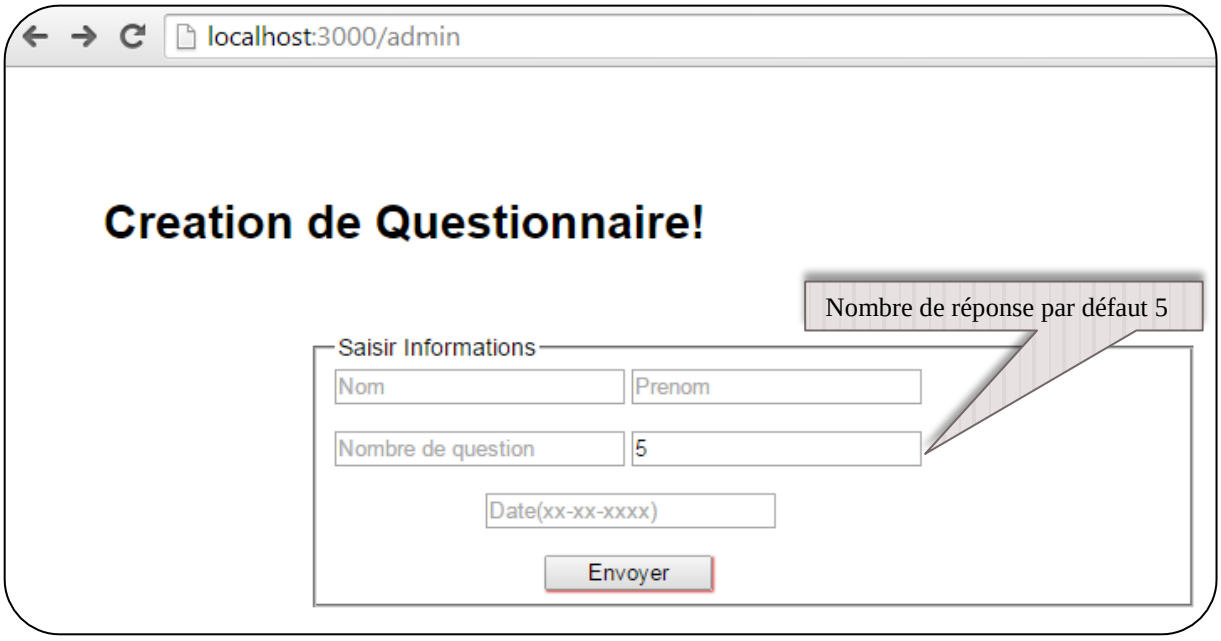


Figure 5 : Information nécessaire pour créer un formulaire

Le code source est un formulaire du type POST qui va être récupéré par la deuxième route de NodeJs.

```
extends layout

block content
  h1= title
  <form action="/admin/info" method="post">
  <fieldset>
  <legend> Saisir Informations</legend>
  <input type="text" name="Lastname" placeholder="Nom"  required="required"/>
  <input type="text" name="Firstname" placeholder="Prenom " required="required"  />   </br> </br>
  <input type="text" name="NBques" placeholder="Nombre de question"  required="required"/>
  <input type="text" name="NBrep" placeholder="Nombre Reponse (5 default)" value="5" /> </br> </br>
  <input type="text" name="Date" placeholder="Date(xx-xx-xxxx)" required="required" /> </br> </br>
  <input type="submit" value="Envoyer" />
  </fieldset>
  </form>
```

Les informations reçues par la deuxième route doivent être enregistrées dans la base de données, pour se faire il faut créer un dossier dans notre projet où on va sauvegarder la BDD (data) ensuite avec un invité de commande on se place sur C:\Program Files\MongoDB\Server\3.0\bin et on exécute la commande « mongod db –dbpath »

```
# cd c:\Program Files\MongoDB\Server\3.0\bin
# mongod –dbpath ~\MonProjet\data
```

- La deuxième route reçoit les informations en POST, la fonction CALLBACK sauvegarde les informations dans des variables locales ensuite on crée une collection dans notre base de données (req.db.get('NomDeLaCollection')) pour insérer ces données (collection.insert()). Si le traitement c'est bien passé on redirige vert la page ou l'administrateur va créer le formulaire sinon on affiche un message d'erreur causé par un problème avec l'insertion des données dans la base de données.

```
router.post('/admin/info',
       function(req, res)
      {
          var db = req.db;
          Nom=req.body.Lastname;
          Prenom=req.body.Firstname;
          Date=req.body.Date;
          NBques=req.body.NBques;
          NBrep=req.body.NBrep;
          var collection= db.get('Admincollection');
          collection.insert    (
          {
            "Nom":Nom,
            "Prenom":Prenom,
            "NBques":NBques,
            "NBrep":NBrep,
            "Date":Date
          } ,
           function (err, doc)
          {
             if (err) {
            // En cas de problème, on renvoie une erreur
            res.send("Il y a un problème pour insérer les données dans la base.");
            }
            else {
            // En cas de succès on redirige vert la page /fill
            res.location("fill");
            res.redirect("fill");
            }
          })
       });
```

Le formulaire vide est généré par rapport aux informations reçues de l'administrateur, ce qui impose à la troisième route d'aller récupérer ces informations à la base de données et ensuite l'envoyé à la page web concerné.

- La troisième route va donc se connecter à la base de donné (req.db.get('Admincollection')) pour récupérer les données (collection.find()) et les envoyés à la page web (jade) à l'aide de la fonction « render() » qui prend en paramètre le nom du fichier jade et la variable qui contiendra les résultats reçus par la base de données.

  La fonction « find » a la syntaxe suivante :

  collection.find(query[[[, fields], options], callback]);

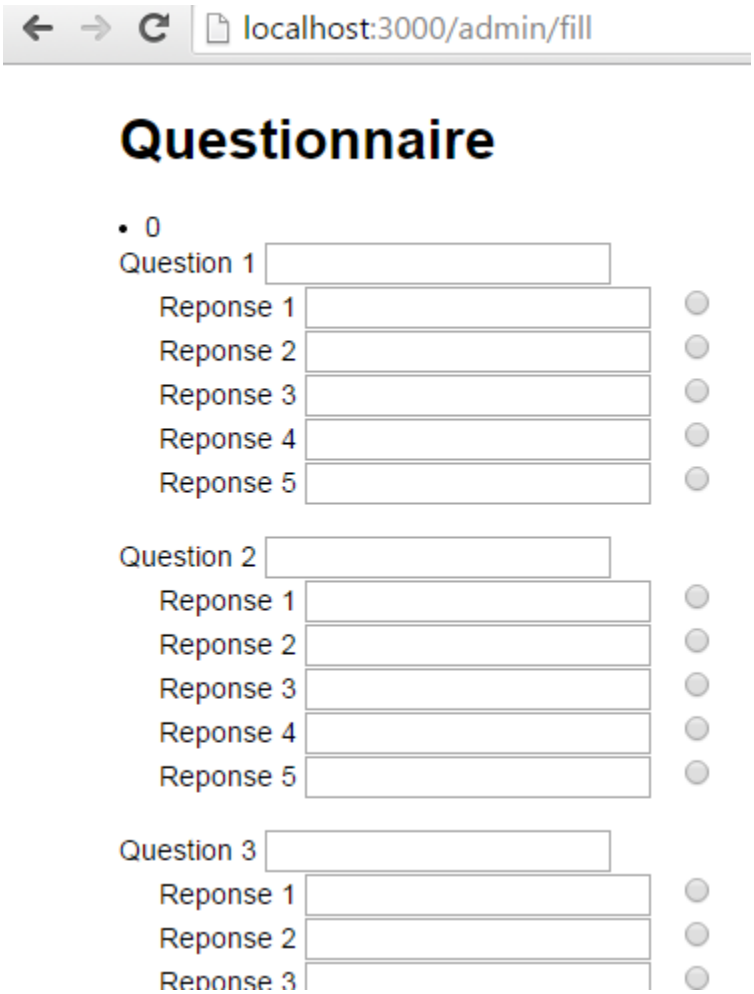
  Le paramètre « query » est utilisé pour visé un résultat spécifique, le paramètre « options » est utilisé pour le tri, min, max ou toute autre action sur la base de donnée.

  Le paramètre « callback » est la fonction qui va envoyer ou/et recevoir les résultats grâce à ces paramètre « req » (requête) et « res » (réponse).

```
router.get('/admin/fill', function(req, res, next) {
  var db = req.db;
  var collection = db.get('Admincollection');
   collection.find({},{},function(e,docs){
       res.render('AdminFill', {
          "_AdminInfo" : docs, title:'Questionnaire'
       });
    });
```

Les données récupérées de la BDD par la fonction « find() » sont stockées dans la variable « docs » (réponse) qui va initialiser la variable « _AdminInfo » qu'on utilisera dans le fichier jade pour exécuter le traitement d'affichage du formulaire vide.

Pour l'affichage de la page j'utilise le code source suivant, qui utilise une boucle par rapport au nombre de questions et à chaque question on affiche le nombre de réponses désirer.




```
extends layout

block content
  h1= title
  <form action="/admin/fill/ok" method="post">
  for user, k in _AdminInfo
    li= k
       -var nbr=user.NBques
       -for(var i=1;i<=nbr; i++)
          label Question #{i} <input type="text" name="q#{i}"
required="required"/>  </br>
          -for(var j=1;j<=user.NBrep;j++)
                  Reponse #{j}  <input type="text"
name="q#{i}r#{j}"/>  
             <input type="radio" name="state#{i}" autofocus/></br>
          </br>
  <input type="submit" value="Envoyer" />
  </form>
```

- La quatrième route envoie les informations à la base de données sous format JSON, le format qu'on a choisi pour le projet est le suivant :

```
{
   "ID ": 154531235,
   "Nom" : "XXXXXX",
   "Prenom" : "XXXXX"
   "NBques" : 5,
   "NBrep" : 5,
   "Date" : "18-04-2015",

   "Questionnaire" :
    {
         "Question1" :
          {
                   "Question" : "XXXXXXXX ? ",
                   "ReponseA " : "XXXXXX",
                   "EtatA" : "true",
                   "ReponseB " : "XXXXXX",
                   "EtatA" : "false",
                   "ReponseC " : "XXXXXX",
                   "EtatA" : "false",
                   "ReponseD " : "XXXXXX",
                   "EtatA" : "false",
                   "ReponseD " : "XXXXXX",
                   "EtatA" : "false",
                   .....
          },
    },

}
```

On récupère les informations envoyées grâce à l'attribut « req.body » et ensuite on l'envoie à la base de données :

```
router.post('/admin/fill/ok', function(req, res, next) {
 var db = req.db;
 var collection = db.get('Admincollection');
 var query;
 for(var i=1;i<=NBques; i++)
 {
    query=query+"{$push:{Questionnaire:{Question "+i+":{";
    for(var j=1;j<NBrep;j++)
    {
      query=query+"Question :"+q#i+",Reponse "+j+": "+q#ir#j+", Etat
"+j+": " +state#i"+ ", " ;
    }
         query=quer+"}, },";
 }
 collection.update(query);
});
```

CONCLUSION &&
PERSPECTIVE

Ce projet nous a permis de mettre en pratique ce que nous avons acquis tout au long de notre formation et aussi de faire face à des problématiques. En effet nous sommes partis d'un réel besoin, nous avons pu découvrir toutes les grandes étapes d'un projet, depuis la rédaction du cahier des charges jusqu'à la mise en en pratique.

Or ce travail était une occasion pour approfondir nos connaissances en langage informatique (Node.JS, HTML, PHP, JavaScript…), en plus des langages acquis et déjà étudier on a pu découvrir des nouveaux langages puissants, qui reviennent de plus en plus et que nous n'avions jamais utilisés (Node.JS, Express…).

Afin de mener à bien ce projet, plusieurs phases ont été réalisées pour atteindre les objectifs préalablement fixés. Avant de se lancer dans la phase du développement où nous avons essayé de mettre en place maximum des fonctionnalités pour répondre aux attentes des utilisateurs.

Il existe plusieurs aspects du projet qui peuvent être améliorés ou modifiés. On pourrait penser à une autre manière d'architecturer notre application ou encore développer une application mobile pour l'utilisation des clients ou même pour les deux parties concernées qui impliqueras une connexion pour identifier l'utilisateur. Nous espérons ainsi que notre travail sera mené à évoluer et que notre retour d'expérience sera bénéfique à l'avancée des recherches et des prochains développements sur des sujets semblables.

III. Bibliographie :

[4] Programmation avec Node.js, Express.js et MongoDB de Éric Sarrion, Groupe Eyrolles Diffusion Geodif

Webographie :

[1] http://fr.openclassrooms.com/informatique/cours/dynamisez-vos-sites-web-avec-javascript/introduction-au-javascript

[2] http://fr.openclassrooms.com/informatique/cours/des-applications-ultra-rapides-avec-node-js

[3] http://perso-laris.univ-angers.fr/~lhommeau/teaching/HTML5/TD/TD4NODEBISV2.html

[5] http://nodejs.org/api/

[6] http://expressjs.com/api.html#app.use

[7] http://fr.wikipedia.org/wiki/Node.js

[8] https://thinkster.io/mean-stack-tutorial/

[9] http://www.rfc-editor.org/rfc/rfc2616.txt

Table des illustrations

Système de vote interactif

Projet réalisé par :
BELLAJ Mehdi

Projet encadré par :
Mehdi Lhommeau

Résumé :

Dans le cadre du projet d'EI4 AGI à l'ISTIA nous avons été amenés à réaliser un système de vote interactif permettant de réaliser un sondage des participants d'une assemblée en leur posant des questions, et en recueillant leurs réponses, grâce à une application Web basée sur une carte Raspberry Pi. On peut imaginer utiliser ce système de vote dans un amphi lors d'un cours magistral pour recueillir des réponses à des questions de cours. Le système développé doit être très simple à déployer.

On peut utiliser les téléphones des étudiants comme système de vote dans la limite ou on a accès au site Web mis à la disposition. Le serveur permettant de recueillir les votes sera basé sur la Raspberry Pi.

Mots Clés :

Node.JS , Express , Jade , JavaScript , HTML , Vote interactif , Application Web

---

Summary :

As part of the project EI4 AGI ISTIA we were led to make a voting system for carrying out a survey of participants of a meeting by asking questions and collecting their answers thanks to a Web application based on a Raspberry Pi board.

we can imagine using this voting system during a lecture, to gather answers to questions of course. The developed system must be easy to deploy.

Students can use phones with voting system if they have access to the Web site.

Keywords :

Node.JS , Express , Jade , JavaScript , HTML , Vote interactif , Application Web