

ENSIBS – Cybersécurité du Logiciel

# Rapport Projet Web

Projet 9 : Élections cantonales de Rennes – 2011

BOËDA Malo & SOMMER Clément  
18/05/2020

## Table des matières

|   |    |
|---|----|
| Introduction.....   | 3  |
| I. Présentation de l'application Web.....                 | 4  |
| a) Choix des technologies.....                            | 4  |
| b) Visuels de l'application.....                          | 5  |
| II. Dans le cœur de l'application.....                    | 7  |
| a) Architecture générale du projet.....                   | 7  |
| b) Le fichier HTML.....                                   | 8  |
| 1. La barre de navigation.....                            | 9  |
| 2. La division des lieux d'une circonscription.....       | 9  |
| 3. La division des résultats du vote.....                 | 10 |
| c) Le contrôleur Javascript, pilier de l'application..... | 11 |
| 1. La fonction loadData().....                            | 11 |
| 2. La fonction loadCirconscription().....                 | 12 |
| 3. La fonction loadVille(nom).....                        | 13 |
| 4. Les fonctions loadLieu(lieu) et loadCandidats().....   | 14 |
| III. Un bilan.....  | 16 |
| a) Les difficultés.....                                   | 16 |
| b) Les améliorations possibles.....                       | 16 |

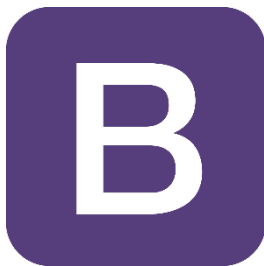
## Introduction

Ce projet est une partie majeure du module de Programmation Web. Le but de ce projet est de réaliser une application web permettant d'afficher les données utiles retournée par notre API. Pour notre groupe, il s'agit de traiter les données des élections cantonales de Rennes pour l'année 2011. Ce projet a pour principale contrainte d'être réalisée sous le Framework Javascript de notre choix. L'équipe projet se compose de BOËDA Malo et SOMMER Clément. Pour ce qui est de la répartition des tâches, nous avons tous les deux travaillés sur tous les aspects du projet mais chaque membre à une spécification. SOMMER Clément est le responsable Backend, BOËDA Malo le responsable Frontend.

## I. Présentation de l'application Web

### a) Choix des technologies

Pour ce projet, nous devons utiliser le Framework Javascript de notre choix entre Angular, Angular JS, React, VueJS. Nous avons choisi d'utiliser le Framework Angular JS. Nous avons choisi ce Framework car il possède une documentation complète. De plus, nous avons déjà eu une expérience avec ce Framework lors de notre DUT Informatique à Vannes. De plus, Angular JS permet de réaliser des applications web hautement interactives. Ceci est un critère très important car nous devons réaliser une application web qui privilégie la clarté ainsi que la simplicité.



Pour ce qui est de la partie Frontend, nous avons utilisé Bootstrap. Il s'agit d'un Framework CSS. Il permet de faire la présentation graphique d'un site. Son grand avantage est qu'il permet de rendre un site « responsive » de manière assez facile comparée à la création de son propre CSS. C'est un critère très important car la « responsivité » est un élément qui ne doit pas être mis à l'écart. Ce Framework est très populaire et c'est un incontournable.

Nous utiliserons bien sûr les technologies web HTML, CSS et Javascript qui sont la base de toute application web. Ces technologies ne sont plus à présenter.



Ce projet a été réalisé pendant le confinement. Le travail de groupe était donc plus difficile. Nous avons donc décidé d'utiliser GitHub pour ce projet. Ce git permet à tout le groupe de travailler sur le même projet en même temps et à distance. Il permet également de retracer le développement de l'application web et de revenir en arrière en cas de problème.

## b) Visuels de l'application

Pour le visuel de l'application, nous avons décidé d'utiliser un Template gratuit très simple et que nous avons adapté pour le besoin de notre application. Il s'agissait d'un Template basique, qui n'utilisait pas Angular JS. Il s'agit d'un site « One page ». Nous trouvons ce type de site très intuitif, qui facilite la navigation. Voici un comparatif entre le Template original et notre application web finale :

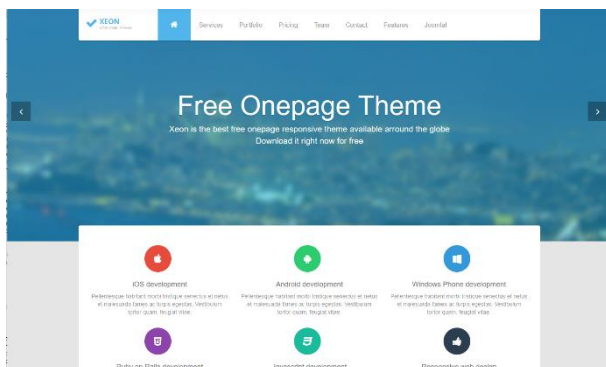


Figure 1 - Template de base



Figure 2 - Notre application web

Comme on peut le constater, la structure du document a été complètement adaptée afin d'afficher les données désirées. Utiliser un Template permet tout de suite d'avoir un site agréable et qui à l'énorme avantage d'être complètement responsive. Au premier abord, on constate que notre application affiche uniquement les circonscriptions dans la barre de navigation. Il suffit de cliquer sur une de ces circonscriptions afin d'avoir le choix parmi des lieux où se sont déroulés les votes pour les élections cantonales de Rennes en 2011.



Figure 3 - Une fois la circonscription choisie

Une fois la circonscription choisie, des informations apparaissent. On voit alors tous les lieux de la circonscription choisie il suffit alors de cliquer sur un des lieux qui apparaissent afin d'obtenir les informations des votes qui ont eu lieu à cet endroit.

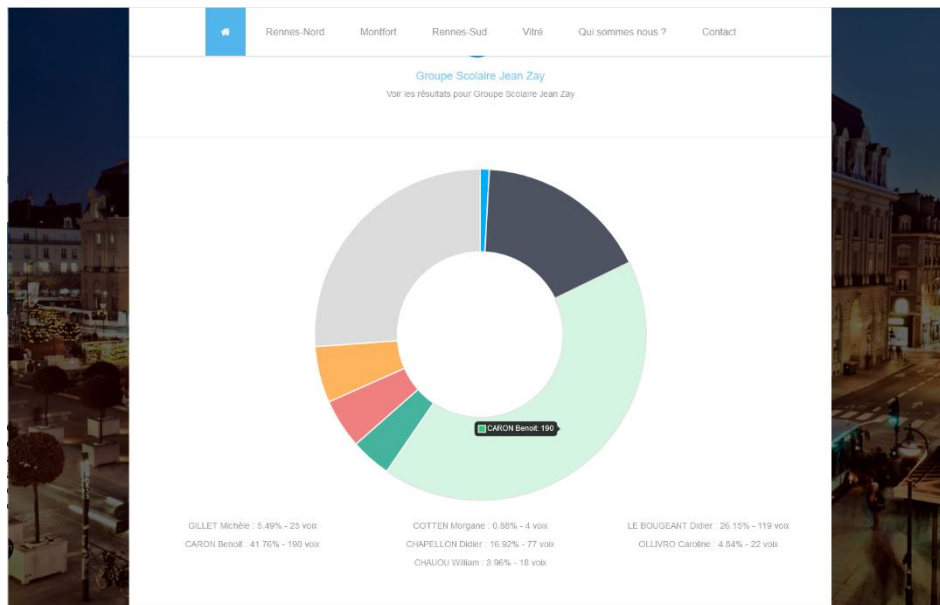


Figure 4 - Une fois le lieu choisi

Une fois le lieu choisi, les résultats des votes apparaissent grâce à un graphique. Ce graphique est géré par le Framework Chart.JS. C'est un Framework spécialisé dans l'affichage de graphique. Il nous est donc très utile.



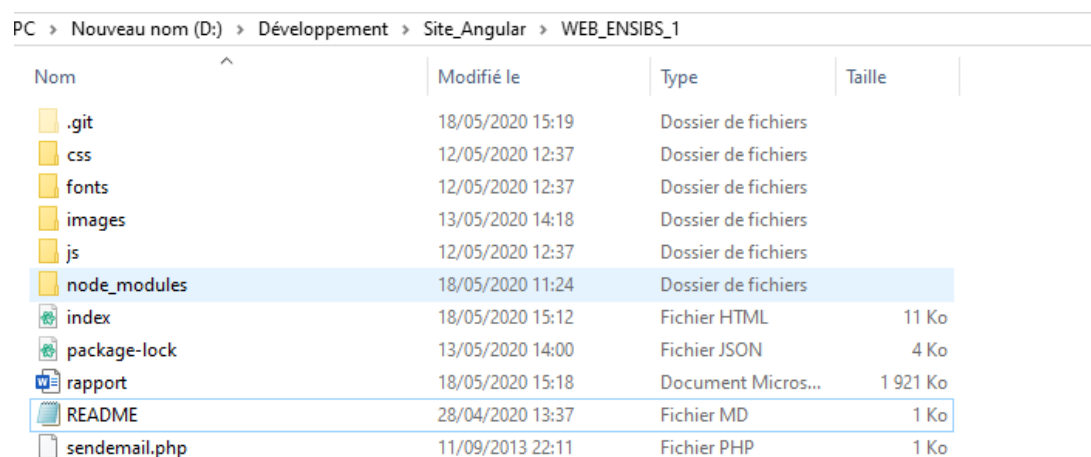
Figure 5 - Chargement d'une autre circonscription

Si on veut les informations d'une autre circonscription, il suffit de choisir dans la barre de navigation. Une fois ce choix fait, on remarque que le précédent graphique disparaît. Toutes ces apparitions de divisions HTML sont gérés grâce à Angular JS. Nous détaillerons ce fonctionnement par la suite.

## II. Dans le cœur de l'application

### a) Architecture générale du projet

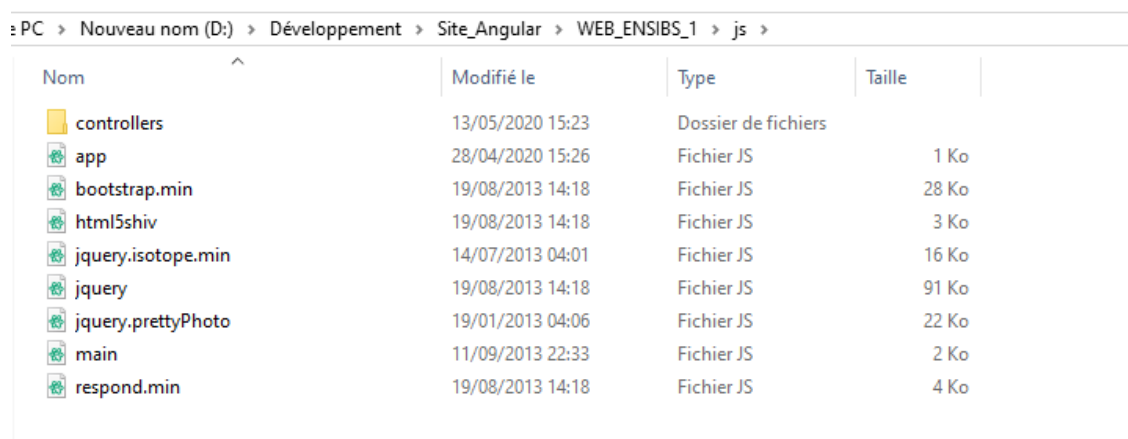
Voici ci-dessous l'architecture de notre application web. Il s'agit d'une architecture classique. On retrouve les différents dossier CSS, JS, fonts, ainsi que notre fichier HTML. Dans le dossier « node\_modules », on retrouvera tous les modules nécessaires au bon fonctionnement de l'application (Angular JS, Chart.JS notamment). Pour lancer notre application, il suffit d'ouvrir le fichier index.html dans le navigateur de votre choix.



| Nom           | Modifié le       | Type                | Taille   |
|---------------|------------------|---------------------|----------|
| .git          | 18/05/2020 15:19 | Dossier de fichiers |          |
| css           | 12/05/2020 12:37 | Dossier de fichiers |          |
| fonts         | 12/05/2020 12:37 | Dossier de fichiers |          |
| images        | 13/05/2020 14:18 | Dossier de fichiers |          |
| js            | 12/05/2020 12:37 | Dossier de fichiers |          |
| node_modules  | 18/05/2020 11:24 | Dossier de fichiers |          |
| index         | 18/05/2020 15:12 | Fichier HTML        | 11 Ko    |
| package-lock  | 13/05/2020 14:00 | Fichier JSON        | 4 Ko     |
| rapport       | 18/05/2020 15:18 | Document Micros...  | 1 921 Ko |
| README        | 28/04/2020 13:37 | Fichier MD          | 1 Ko     |
| sendemail.php | 11/09/2013 22:11 | Fichier PHP         | 1 Ko     |

Figure 6 - Architecture du projet

Ci-dessous, nous retrouvons le dossier JS en détail. Ce dossier contient tous les scripts nécessaires au bon fonctionnement de l'application. Les seuls qui vont nous intéresser sont les fichiers app.js et controleur.js (dans le dossier controllers). Tous les autres sont des scripts liés aux différents modules utilisés et qui sont natifs.



| Nom                | Modifié le       | Type                | Taille |
|--------------------|------------------|---------------------|--------|
| controllers        | 13/05/2020 15:23 | Dossier de fichiers |        |
| app                | 28/04/2020 15:26 | Fichier JS          | 1 Ko   |
| bootstrap.min      | 19/08/2013 14:18 | Fichier JS          | 28 Ko  |
| html5shiv          | 19/08/2013 14:18 | Fichier JS          | 3 Ko   |
| jquery.isotope.min | 14/07/2013 04:01 | Fichier JS          | 16 Ko  |
| jquery             | 19/08/2013 14:18 | Fichier JS          | 91 Ko  |
| jquery.prettyPhoto | 19/01/2013 04:06 | Fichier JS          | 22 Ko  |
| main               | 11/09/2013 22:33 | Fichier JS          | 2 Ko   |
| respond.min        | 19/08/2013 14:18 | Fichier JS          | 4 Ko   |

Figure 7 - Détails du dossier JS

Le fichier app.js est un fichier de « routing ». C'est-à-dire un fichier qui permet la bonne redirection vers le bon contrôleur lors d'une action sur la page HTML. Quant au fichier controleur.js, il contient toutes les fonctions utiles au fonctionnement de l'application web. Ce fichier sera détaillé par la suite.

## b) Le fichier HTML

Afin d'utiliser tous les Framework précédemment cités, il est nécessaire d'importer tous les scripts nécessaires à son bon fonctionnement. C'est grâce à ses balises scripts que les Framework peuvent être utilisés dans notre HTML.

```
<script src="js/jquery.js"></script>
<script src="js/bootstrap.min.js"></script>
<script src="js/jquery.isotope.min.js"></script>
<script src="js/jquery.prettyPhoto.js"></script>
<script src="js/main.js"></script>
<script src="node_modules\angular\angular.js"></script>
<script src="node_modules\angular-route\angular-route.js"></script>
<script src="js\app.js"></script>
<script src="js\controllers\controleur.js"></script>
<script src="node_modules\chart.js\dist\Chart.min.js"></script>
<script src="node_modules\angular-chart.js\dist\angular-chart.min.js"></script>
<script type="text/javascript">
```

Figure 8 - Importation des Framework

Pour utiliser AngularJS, il est nécessaire de donner certains attributs à notre « body » :

```
<!-- Définition du controleur pour le body de notre page ainsi que lancement de la fonction loadData() au chargement de la page -->
<body data-spy="scroll" data-target="#navbar" data-offset="0" ng-app='rennesData' ng-controller='controleur' ng-init='loadData()'
```

Figure 9 - Définition du contrôleur pour le body

On retrouve dans cette balise « body » trois attributs liés à AngularJS. Tout d'abord, on a l'attribut « ng-app » qui permet de définir le nom de l'application AngularJS. Ensuite, on retrouve l'attribut « ng-controller » qui permet de choisir le contrôleur lié à notre fichier HTML (controler.js entre autres). Enfin, on a l'attribut « ng-init » qui permet de lancer une fonction au chargement de la page. La fonction lancée est la fonction « loadData() ». Cette fonction permet de charger toutes les données issues de notre API. Elle ainsi que toutes les autres cités dans cette partie seront détaillées plus tard dans la partie du contrôleur Javascript.

Notre application web, on retrouvera trois grandes parties : la barre de navigation, la partie où les lieux d'une circonscription sont affichés et enfin la partie où les résultats du vote sont affichés. Nous allons détailler ces parties une à une dans la suite de ce rapport.



## 1. La barre de navigation

La barre de navigation est la partie centrale de notre application. Elle permet de charger tous les lieux en fonction de la circonscription choisie. Voici le code HTML pour cette partie :

```
<div class="collapse navbar-collapse">
  <ul class="nav navbar-nav barre">
    <li class="barre"><a href="#top"><i class="icon-home"></i></a></li>
    <!-- On boucle (ng-repeat) dans le JSON de base afin de mettre autant de li que de circonscriptions chargées -->
    <li class="barre" ng-repeat='data in circonscriptions'><a href="#{{data}}"' ng-click='loadVille(data)''>{{data}}</a></li>
    <li class="barre"><a href="#about-us">Qui sommes nous ?</a></li>
    <li class="barre"><a href="#contact">Contact</a></li>
  </ul>
</div>
```

Figure 10 - Barre de navigation

Cette division a une structure classique de barre de navigation. Une balise « ul » elle-même composée de plusieurs balises « li ». Mais ce qui nous intéresse est la deuxième balise « li ». Elle possède un attribut « ng-repeat ». Cet attribut permet de boucler dans une liste présente dans notre contrôleur. Dans notre cas, elle va boucler dans la liste des circonscriptions. Pour chaque circonscription trouvée, on va créer une balise « li » qui va contenir le nom de cette circonscription. Chaque balise « li » est cliquable. Lorsque l'on clique dessus, la fonction « loadVille(data) » sera lancée. Cette fonction permet de charger tous les lieux de la circonscription « data » passée en paramètre.

## 2. La division des lieux d'une circonscription

La division des lieux d'une circonscription affiche tous les lieux pour une circonscription donnée. Voici le code HTML de cette partie :

```
<!-- section des lieux avec un ng-class qui dépend de l'attribut visible1 (expliqué dans le rapport) -->
<section id="services lieux" ng-class="{ok:visible1}">
  <div class="containeur">
```

Figure 11 - Gestion de l'affichage de la section

Cette section n'est pas affichée lors du premier chargement de l'application. Cet affichage est géré grâce à l'attribut « ng-class ». Cet attribut permet d'affecter à la section une classe en fonction d'un booléen. Dans notre cas, si la variable « visible1 » est vraie, la section ne sera pas affichée car elle possèdera la classe « ok ». On peut voir dans ce CSS que la classe « ok » désactive l'affichage.

```
.ok{
  display: none !important;
}
```

```

<!-- section des lieux avec un ng-class qui dépend de l'attribut visible1 (expliqué dans le rapport) -->
<section id="services lieux" ng-class="{ok:visible1}">
  <div class="container">
    <div class="box first center">
      <h3 id="titre">Par lieux :</h3>
      <div class="row">
        <!-- On boucle (ng-repeat) dans les circonscriptions afin de mettre autant une div pour chaque circonscriptions chargée -->
        <div class="col-md-4 col-sm-6" ng-repeat="data in lieux">
          <div class="center">
            <i class="icon-tasks icon-md icon-color3"></i>
            <!-- Chargement des informations pour la circonscription choisie (fonction loadLieu(data)) -->
            <h4><a href="" ng-click='loadLieu(data)'>{data}</a></h4>
            <p>Voir les résultats pour {{data}}</p>
          </div>
        </div><!--/.col-md-4-->
      </div><!--/.row-->
    </div><!--/.box-->
  </div><!--/.container-->
</section><!--/#services-->

```

Figure 12 - Division des lieux d'une circonscription

Cette division affichera tous les lieux d'une circonscription. Ce qui nous intéresse principalement c'est la balise « div » contenant l'attribut « ng-repeat ». Comme dis précédemment, cette balise va permettre de créer autant de divisions que de circonscriptions. Elle va boucler dans les lieux. Chaque division sera cliquable. Lorsqu'on cliquera dessus, la fonction « loadLieu(data) ». Cette fonction permettra de générer toutes les informations sur le résultat des votes dans pour le lieu « data ».

### 3. La division des résultats du vote

Cette division contiendra les résultats du vote pour le lieu d'une circonscription. Comme la division précédente, elle gèrera son affichage à l'aide de l'attribut « ng-class » qui sera égale à « ok » ou non. Ceci permet de ne plus afficher le résultat de l'ancien lieu lorsqu'on change de circonscription. Voici le code HTML de cette partie :

```

<!-- section des lieux avec un ng-class qui dépend de l'attribut visible2 (expliqué dans le rapport) -->
<section id="portfolio resultats" ng-class="{ok:visible2}">
  <div class="container">
    <div class="box">
      <!-- Graphique qui représente les informations de votes pour un lieu d'une circonscription -->
      <canvas id="doughnut" class="chart chart-doughnut"
        chart-data="data" chart-labels="labels" chart-options="options">
      </canvas>
      <div class="row">
        <!-- On boucle (ng-repeat) dans les candidats afin de mettre autant une div pour chaque candidat chargé de la circonscription -->
        <div class="col-md-4 col-sm-6" ng-repeat="cand in candidats">
          <div class="center">
            <p id="donnees">{{dataLieu[cand[0]]}} : {{dataLieu[cand[1]]}}% - {{dataLieu[cand[2]]}} voix</p>
          </div>
        </div><!--/.col-md-4-->
      </div><!--/.row-->
    </div><!--/.box-->
  </div><!--/.container-->
</section><!--/#services-->

```

Figure 13 - Division des résultats du vote

Tout d'abord, on peut remarquer l'apparition d'une balise « canva » dans cette division. Cette balise est native au Framework Chart.JS et permet d'afficher un graphique selon la classe qu'on attribut à cette balise. On lui donnera également des données (stockées dans le fichier controler.js) pour permettre son affichage.

Ensuite, on remarque une structure presque équivalente avec la division précédente. C'est-à-dire que notre division possède un attribut « ng-repeat » qui, cette fois, va boucler dans les candidats de la circonscription. Pour chaque candidat, elle va afficher son nom, son nombre de voix ainsi que son pourcentage de votes (informations stockées dans le triplet « cand »).

### c) Le contrôleur Javascript, pilier de l'application

Dans cette partie, nous allons étudier en détail chaque fonction appelée dans le fichier HTML. On détaillera alors son fonctionnement. Notre fichier « controleur.js » contient les fonctions « loadData() », « loadLieu(lieu) », « loadCirconscription() », « loadVille(ville) », « loadCandidats() » et « onload() ». Nous allons uniquement nous intéresser aux cinq premières fonctions. La fonction « onload() » est uniquement une fonction permettant de parser dans un fichier JSON les données de notre API.

#### 1. La fonction loadData()

Cette fonction est très importante pour notre application web. Elle permet d'attribuer à une variable toutes les données de notre API. Voici ce que donne les données brutes reçues par l'API :

```
{
  "nhits": 178,
  "parameters": {
    "dataset": "resultats_c11",
    "format": "json",
    "rows": 10,
    "timezone": "UTC"
  },
  "records": [
    {
      "datasetid": "resultats_c11",
      "recordid": "4322a61cbf35d23a2e73fa17eef1e8c0d4bffb6e",
      "record_timestamp": "2019-05-29T10:02:39.075000+00:00",
      "record_latitude": 46.85,
      "record_longitude": 7.12
    },
    {
      "datasetid": "resultats_c11",
      "recordid": "3a9dad89d7a1fd029d902d74585d5fb86005f29",
      "record_timestamp": "2019-05-29T10:02:39.071000+00:00",
      "record_latitude": 46.85,
      "record_longitude": 7.12
    },
    {
      "datasetid": "resultats_c11",
      "recordid": "22571ee7cfdec9f12d32b9533dbf9fbb11c7c3",
      "record_timestamp": "2019-05-29T10:02:39.071000+00:00",
      "record_latitude": 46.85,
      "record_longitude": 7.12
    },
    {
      "datasetid": "resultats_c11",
      "recordid": "1a50ad71d903b01935888128ab13cfe041b5fa59",
      "record_timestamp": "2019-05-29T10:02:39.071000+00:00",
      "record_latitude": 46.85,
      "record_longitude": 7.12
    },
    {
      "datasetid": "resultats_c11",
      "recordid": "dbcb5806294eefcf987f605b2034d2e000ad5a33f",
      "record_timestamp": "2019-05-29T10:02:39.071000+00:00",
      "record_latitude": 46.85,
      "record_longitude": 7.12
    },
    {
      "datasetid": "resultats_c11",
      "recordid": "d7f1ecad4a1f55fb34e3944615ba4334f809f72",
      "record_timestamp": "2019-05-29T10:02:39.071000+00:00",
      "record_latitude": 46.85,
      "record_longitude": 7.12
    },
    {
      "datasetid": "resultats_c11",
      "recordid": "1927988b1c6a9aca43be6a49198eac44e998e265",
      "record_timestamp": "2019-05-29T10:02:39.071000+00:00",
      "record_latitude": 46.85,
      "record_longitude": 7.12
    },
    {
      "datasetid": "resultats_c11",
      "recordid": "5d9da5130c35d6608b32743fc1b648bf566c5ea",
      "record_timestamp": "2019-05-29T10:02:39.071000+00:00",
      "record_latitude": 46.85,
      "record_longitude": 7.12
    },
    {
      "datasetid": "resultats_c11",
      "recordid": "b491cf3001821fc8576d34d2d4f1950fcb7abc78",
      "record_timestamp": "2019-05-29T10:02:39.075000+00:00",
      "record_latitude": 46.85,
      "record_longitude": 7.12
    },
    {
      "datasetid": "resultats_c11",
      "recordid": "90b75dd514fc664420f2b5d14f156678bf5eaa",
      "record_timestamp": "2019-05-29T10:02:39.075000+00:00",
      "record_latitude": 46.85,
      "record_longitude": 7.12
    }
  ]
}
```

Figure 14 - Données brutes de l'API

L'API nous renvoi un tableau de données. Ce qui nous intéresse principalement est le tableau « records » contenant toutes les informations pour chaque lieu d'une circonscription. On va boucler sur ce tableau afin d'ajouter à la variable « dataPrint » tous les détails de chaque circonscription. Voici le code Javascript dédié à cette action :

```
/** Fonction qui va charger toutes les données de notre API et mettre à jour les circonscriptions */
function loadData(){
  for (i = 0; i < $scope.data.records.length; i++){
    $scope.dataPrint.push($scope.data.records[i].fields);
  }
  loadCirconscription();
}
```

Figure 15 - Fonction loadData()

On lancera ensuite la fonction « loadCirconscription » que je vais vous détailler par la suite.

## 2. La fonction loadCirconscription()

Cette fonction permet de charger tous les noms des circonscriptions présentes dans notre API. Cependant, la variable « dataPrint » contient tous les lieux de l'API de Rennes. Il contient donc plusieurs fois la même circonscription. Il faudra donc faire attention pour ne pas ajouter une circonscription deux fois pour ne pas avoir de doublon dans la barre de navigation. Voici le code Javascript correspondant à cette fonction :

```

/** Fonction loadCirconscription qui chargera tous les noms des circonscriptions présent dans le JSON de base */
function loadCirconscription(){
  let test = false; //variable pour checker si notre circonscription est déjà chargée
  /** On boucle sur toutes nos données */
  for (i = 0; i < $scope.dataPrint.length; i++){
    /** Boucle qui regarde si une circonscription a déjà été chargée (test = true si oui) */
    for(j = 0; j < $scope.circonscriptions.length; j++){
      if($scope.dataPrint[i].nom_circonscription == $scope.circonscriptions[j]){
        test = true;
        break;
      }
    }
    /** On ajoute notre circonscription uniquement si elle n'est pas déjà présente dans le tableau des circonscriptions */
    if(!test){
      $scope.circonscriptions.push($scope.dataPrint[i].nom_circonscription);
    }
    test = false; // On remet à faux pour le prochain tour de boucle
  }
}

```

Figure 16 - Fonction loadCirconscription()

La partie principale de notre fonction est la partie qui vérifie si une circonscription n'a pas déjà été ajoutée au tableau des circonscriptions. Pour cela, on boucle sur ce tableau qui est au départ vide car on n'a pas encore ajouté de circonscription. En fonction du booléen « test », on va donc ajouter ou non une circonscription à ce tableau. Voici ce que renvoi cette fonction à la fin de son exécution :



Figure 17 - Résultat de la fonction après exécution de la fonction loadCirconscription()

Une fois la fonction exécutée, on constate bien la présence de nos quatre circonscriptions sans présence de doublon. L'objectif est donc atteint.

### 3. La fonction loadVille(nom)

Cette fonction permet de charger tous les lieux d'une circonscription. Elle reprend en majeure partie le code principal de la fonction loadCirconscription() mais en variant ce que l'on veut obtenir. C'est pourquoi je ne vais pas détailler son fonctionnement. Voici le code Javascript de cette fonction :

```

/** Fonction loadVille qui charge tous les lieux d'une circonscription dans le tableau des lieux */
$scope.loadVille = function (nom){
    $scope.visible1 = false; // On rend notre section/div visible (car on lui retire la ng-class "ok")
    $scope.visible2 = true; // On rend notre section/div non visible (car on lui remet la ng-class "ok")
    // Remise à 0 de tous les attributs d'un lieu afin de ne pas avoir de doublon
    $scope.lieux = [];
    $scope.candidats = [];
    $scope.labels = [];
    $scope.data = [];
    let test = false; //variable pour checker si notre lieu est déjà chargé
    /** On boucle sur toutes nos données */
    for (i = 0; i < $scope.dataPrint.length; i++){
        /** Boucle qui regarde si un lieu a déjà été chargé (test = true si oui) */
        for(j = 0; j < $scope.lieux.length; j++){
            if($scope.dataPrint[i].nom_lieu == $scope.lieux[j]){
                test = true;
                break;
            }
        }
        /** On ajoute notre nom de lieu uniquement si elle n'est pas déjà présente dans le tableau des circonscriptions */
        if($scope.dataPrint[i].nom_circonscription == nom && !test){
            $scope.lieux.push($scope.dataPrint[i].nom_lieu);
        }
        // On remet à faux pour le prochain tour de boucle
        test = false;
    }
}

```

Figure 18 - Fonction loadVille(nom)

On remarque cependant une différence avec la précédente fonction. En effet, on peut voir qu'on vide tous les tableaux en début de fonction. Ceci est réalisée afin de ne pas ajouter à une liste de lieux déjà initialisée avec une autre circonscription des lieux qui ne sont pas dans cette circonscription. On retrouve cela pour la liste des données, des labels mais aussi des candidats. Voici le résultat de cette fonction après exécution :



Groupe Scolaire Pascal Lafaye

Voir les résultats pour Groupe Scolaire Pascal Lafaye



Groupe Scolaire Eugène Guillevic

Voir les résultats pour Groupe Scolaire Eugène Guillevic



Groupe Scolaire Carle Bahun

Voir les résultats pour Groupe Scolaire Carle Bahun

► Array(3) [ "Groupe Scolaire Pascal Lafaye", "Groupe Scolaire Eugène Guillevic", "Groupe Scolaire Carle Bahun" ]

»

Figure 19 - Résultat après exécution de la fonction loadVille(nom)

#### 4. Les fonctions loadLieu(lieu) et loadCandidats()

Je vais détailler ces fonctions ensemble car la fonction « loadLieu() » est juste une fonction qui charge toutes les informations pour un lieu pour ensuite faire appel à la fonction loadCandidats(). Cette dernière va quant à elle trier les données du lieu en question pour en ressortir une liste de candidats avec leur nombre de voix ainsi que leur pourcentage de vote. Voici ce que ressort la fonction loadLieu(lieu) après exécution :

```

{
  candidat_1: "GILLET Michèle"
  candidat_2: "COTTEN Morgane"
  candidat_3: "LE BOUGEANT Didier"
  candidat_4: "CARON Benoit"
  candidat_5: "CHAPELLON Didier"
  candidat_6: "OLLIVRO Caroline"
  candidat_7: "CHAUOU William"
  code_election: "C11"
  date_election: "20110320"
  nb_bulletins: 459
  nb_emargements: 459
  nb_express: 455
  nb_inscrits: 1231
  nb_nuls: 4
  nb_voix_1: 25
  nb_voix_2: 4
  nb_voix_3: 119
  nb_voix_4: 190
  nb_voix_5: 77
  nb_voix_6: 22
  nb_voix_7: 18
}

```

Figure 20 - Résultat après exécution de la fonction loadLieu(lieu)

Sur ce résultat, il n'y a pas toutes les informations car cela prendrait trop de place. Mais on comprend vite la nécessité de trier ces informations afin d'en sortir les candidats ainsi que les résultats. On remarque alors que tout est mélangé. On va donc utiliser une REGEX afin de prendre tous les candidats. Voici le code Javascript de cette fonction :

```

/** Fonction loadCandidats qui charge tous les informations d'un candidat d'une circonscription dans le tableau des candidats */
function loadCandidats(){
  // Remise à 0 de tous les attributs d'un lieu afin de ne pas avoir de doublon
  $scope.candidates = [];
  $scope.labels = [];
  $scope.data = [];
  $scope.visible2 = false; // On rend notre section/div non visible (car on lui remet la ng-class "ok")
  regex1 = /candidat_\d/gm; // Utilisation d'une REGEX afin de détecter le pattern 'candidat_' afin de récupérer tous les candidats d'un lieu
  let cand, nbvoix, pourcentage, indice; // création des variables locales à cette fonction
  /** On boucle dans les tous les lieux de notre circonscription */
  for(const obj in $scope.dataLieu){
    /** Test pour voir si notre info est bien un candidat */
    if(obj == obj.match(regex1)){
      cand = obj; // attribution du nom de candidat
      indice = obj[obj.length-1]; // Récupération de l'indice du candidat
      /** On boucle dans les tous les lieux de notre circonscription */
      for(const obj in $scope.dataLieu){
        /** si notre ligne est le nombre de voix pour notre candidat actuel (valeur de l'indice) */
        if(obj == 'nb_voix_' + indice){
          nbvoix = obj; // on attribut le nombre de voix du candidat
        }
        /** si notre ligne est le pourcentage pour notre candidat actuel (valeur de l'indice) */
        if(obj == 'pourcentage_' + indice){
          pourcentage = obj; // on attribut le pourcentage de voix du candidat
        }
      }
      $scope.candidates.push([cand,pourcentage,nbvoix]); // on ajoute à notre tableau un triplet candidat, pourcentage et nbvoix d'un candidat
      $scope.labels.push($scope.dataLieu[cand]); // Labels pour notre graphique (nom de candidat)
      $scope.data.push($scope.dataLieu[nbvoix]); // data pour notre graphique (nombre de voix pour le candidat)
    }
  }
  $scope.candidates.sort();
}

```

Figure 21 - Fonction loadCandidats()

Au début de cette fonction, on remet les listes candidats, labels, data à zéro afin de ne pas avoir d'éventuels candidats d'autres circonscriptions par exemple. Dans un premier temps, je vais boucler sur ma liste créée par « loadLieu(lieu) ». Je vais tout d'abord regarder si mon item de la liste est un candidat à l'aide de la REGEX. Si s'en est un, on assigne à notre variable « cand » le nom du candidat. Ensuite, on va encore boucler sur la liste sortie par la fonction « dataLieu(lieu) » afin de récupérer le nombre de voix ainsi que le pourcentage de ce même candidat. A la fin de ces deux boucles, on ajoute le triplet à la liste de candidats. Voici le résultat de cette fonction après exécution :

```

(7) [-]
  0: Array(3) [ "candidat_1", "pourcentage_1", "nb_voix_1" ]
  1: Array(3) [ "candidat_2", "pourcentage_2", "nb_voix_2" ]
  2: Array(3) [ "candidat_3", "pourcentage_3", "nb_voix_3" ]
  3: Array(3) [ "candidat_4", "pourcentage_4", "nb_voix_4" ]
  4: Array(3) [ "candidat_5", "pourcentage_5", "nb_voix_5" ]
  5: Array(3) [ "candidat_6", "pourcentage_6", "nb_voix_6" ]
  6: Array(3) [ "candidat_7", "pourcentage_7", "nb_voix_7" ]
  length: 7
  <prototype>: Array []

```

Figure 22 - Résultat après exécution de la fonction loadCandidats()

Afin de récupérer les informations, il suffira uniquement de charger dans la variable « dataLieu » la ligne désirée. Si je veux le nombre de voix du candidat 6, je taperai alors « \$scope.dataLieu.candidat\_6 ». Voici comment ces données sont ensuite affichées par notre application web :

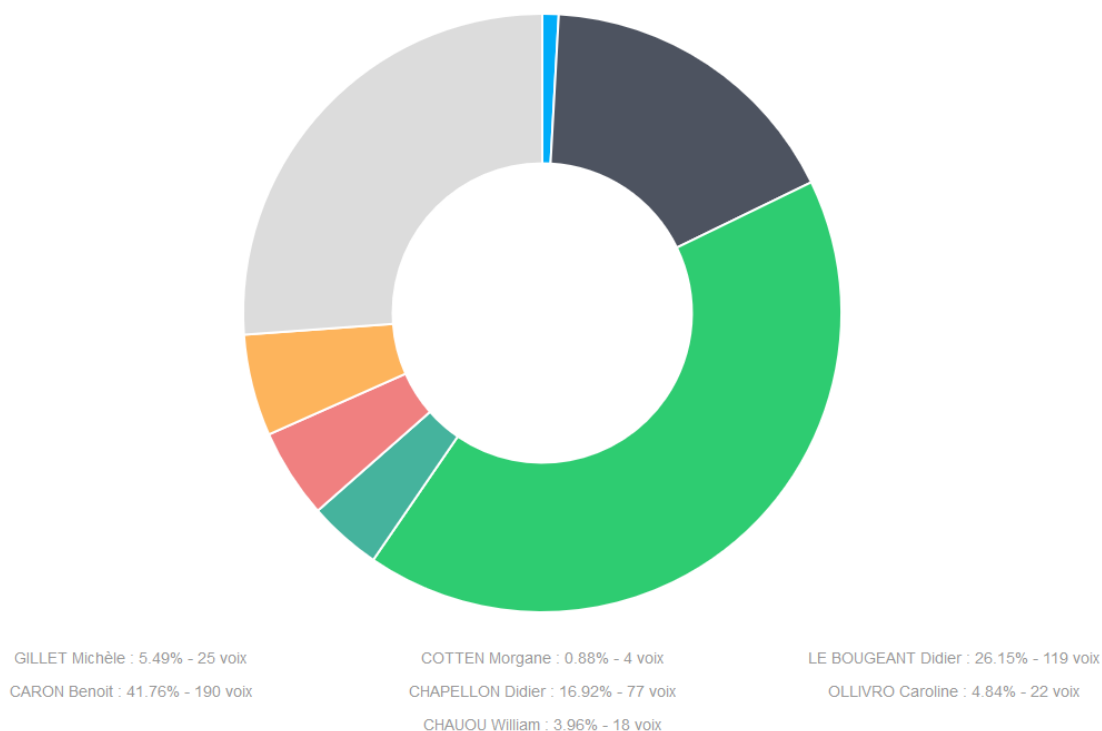


Figure 23 - Affichage des données après exécution de la fonction loadCandidats()

### III. Un bilan

#### a) Les difficultés

Comme on l'avais dit en début de rapport, la prise en main du Framework Angular JS n'a pas été une difficulté pour nous puisqu'on avait déjà eu l'occasion de l'utiliser, de le mettre en pratique. Cependant, ce qui a été pour nous une réelle difficulté était de comprendre la structure et les données de notre API. En effet, il fallait déjà comprendre que l'API nous renvoyait une liste de lieux mais qui pouvait être rattaché à la même circonscription. Il y a donc eu tout un travail de tri ainsi que d'organisation des données.

Une autre difficulté a été l'affichage des données à l'aide du Framework Chart.JS. C'est un Framework que nous n'avions jamais utilisé. Il a donc fallu comprendre comment il fonctionnait et comment on pouvait le mettre en place avec notre API et nos différents modules.

Ce projet nous a donc permis de mettre en pratique de connaissances que nous avions déjà. Cela nous permet aujourd'hui d'avoir des bases et des connaissances solides en programmation web, que ce soit pour le design mais également pour tout ce qui se cache derrière une page web. Il nous a également permis d'apprendre de nouvelles choses comme l'utilisation du Framework Chart.JS. Le web est aujourd'hui quelque chose de très intéressant et qui ne cesse d'évoluer. Il faudra donc rester attentif au fil des années afin de rester efficace dans ces domaines.

#### b) Les améliorations possibles

Nous avons intégré un formulaire de contact à notre application web. Cependant, elle ne fonctionne pas car elle doit être intégrée à un server pour pouvoir effectuer des requêtes en ligne. On pourrait donc facilement imaginer mettre en place un docker hébergeant notre application. Cela pourrait être en continuité direct avec le module de Virtualisation.

De plus, chaque groupe de la promotion avait un sujet différent mais qui traitait également de l'API de Rennes. On pourrait donc imaginer une grosse mise en commun afin d'afficher sur une application commune énormément de données sur la ville de Rennes.