Système de sondage en ligne

Aurélien CURTI
Elia DITMANN
Jérémy FROMENT

JÉRÉMY FROMENT
jeremyfroment@yahoo.fr

Technologie

Ionic 2

Application

Système de sondage en ligne

Description

Un système de sondage en ligne permettant de voter directement depuis notre smartphone à travers une liste de sondages disponibles.

Sur cette application, l'utilisateur a accès à deux menus :

- le premier donne sur la page des sondages :

  Cette page affiche tous les sondages disponibles dans une liste. Si la géolocalisation est activée, l'utilisateur voit aussi une liste de sondages disponibles autour de lui. Chaque sondage est cliquable afin d'y accéder (envoi vers la page du sondage).
  Sur la page du sondage est affiché les représentants à travers un *slider* permettant de les faire défiler en balayant de gauche à droite ou de droite à gauche. Sur chaque *slide*, le nom du représentant, sa photo et un bouton permettant de voter pour lui sont disponibles. Lorsque l'utilisateur vote, il ne peut plus re-voter. Le représentant choisi s'affiche donc à la place du *slider.*

- le second donne sur la page « à propos » :

  Cette page affiche les informations sur l'application et sur les membres l'ayant réalisée.

Adaptations

- S'adapte aux différents smartphones (Windows Phone / Android / IOS) ;
- Position GPS ;

Installation des outils

Le principal outils à installer est Ionic 2. Ionic s'installe en ligne de commande à partir de NPM.
NPM est un gestionnaire de package en CLI que propose Node, et peut être téléchargé en se rendant sur le site web de NodeJS : https://nodejs.org/fr/

Une fois celui-ci installé, il faut donc lancer cette ligne de commande :

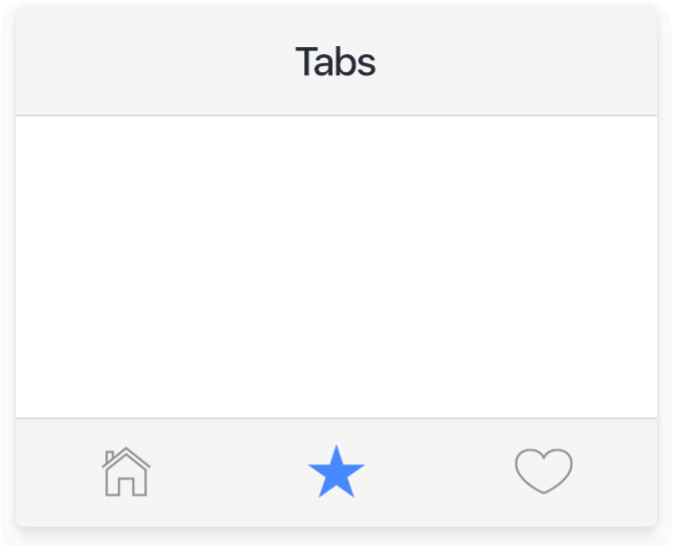
```
$ npm install –g ionic cordova
```

Une fois l'installation terminée, Ionic est prêt, et il n'y a pas d'autres outils à installer.

Architecture

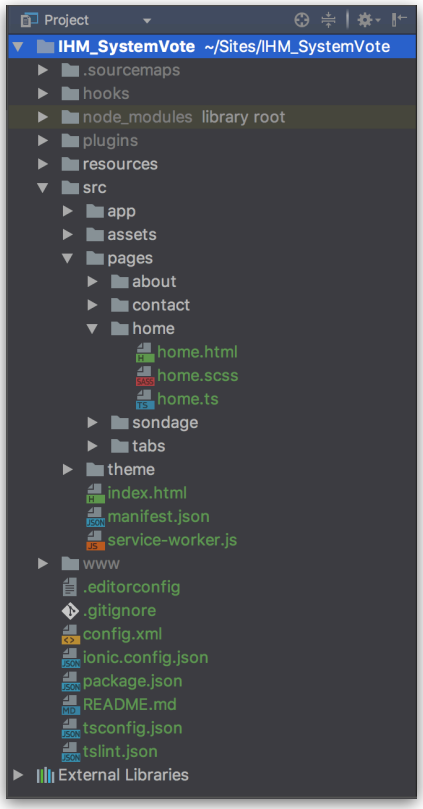
L'architecture de l'application dépend de celle proposée par défaut dans Ionic. En effet, Ionic propose, à travers une ligne de commande, un setup de démo pour l'application.

J'ai donc choisi de commencer avec le modèle « tabs » qui propose une IHM correspondant à mes besoins :




```
$ ionic start myApp tabs
```

Cette commande m'a donc créé le projet de démarrage, avec l'architecture de fichier suivante :



Le dossier « resources » contient les ressources qui vont être utilisées pour les smartphones, comme les icons et les splashscreens.

Le dossier « src » contient tous fichiers de développement, à savoir les « assets » qui contient les ressources (images) dans l'application, le dossier « app » qui contient les fichiers javascript et html principaux de l'application (imports, initialization, etc…). Ensuite il y a le dossier « pages » qui contient les dossiers des pages. Chaque page contient un fichier .html, un fichier .scss (pré-processeur de css) et un fichier .ts qui contient le contenu javascript.

Enfin, il y a divers autres fichiers qui permettent de configurer l'application.

Implémentation du code

Concernant l'implémentation de la géolocalisation :

Dans le fichier « app.module.ts » qui contient la déclaration des choses qui composent notre module, j'ai commencé par effectuer l'import de la librairie.

```
import {Geolocation} from '@ionic-native/geolocation';
```

Puis l'injecter dans les providers.

```
providers: [
    StatusBar,
    SplashScreen,
    Geolocation,
    {provide: ErrorHandler, useClass: IonicErrorHandler}
]
```

Ensuite, il faut l'ajouter au composant principal de l'applisation, à savoir « app.component.ts ».
Donc comme précédemment, commencer par importer la librairie.

```
import {Geolocation} from '@ionic-native/geolocation';
```

Puis l'ajouter aux injections de dépendances dans le « constructor ».

```
constructor(platform: Platform, statusBar: StatusBar, splashScreen: SplashScreen, geolocation: Geolocation) {
```

Ensuite, il faut implémenter le code qui permet de récupérer la longitude et latitude (toujours dans le fichier « app.component.ts ») dans « constructor ».

```
geolocation.getCurrentPosition().then( onfulfilled: (resp) => {
    // resp.coords.latitude
    // resp.coords.longitude
}).catch( onrejected: (error) => {
    console.log('Error getting location', error);
});
```

Enfin, afin de garder la localisation en temps réel, et donc de mettre à jour les sondages selon sa position, j'ai implémenté un watcher qui récupère les données.

```
let watch = geolocation.watchPosition();
watch.subscribe( next: (data) => {
    //
});
```

Comment tester les adaptations ?

Les adaptations peuvent êtres testées à travers un smartphone, lorsqu'on lance l'application, celle-ci demande à l'utilisateur d'accepter ou non d'être localisé. Si sa réponse est positive, une liste de sondages proche de sa position apparaissent.

Difficultés rencontrées

Pour commencer, la première difficulté a été le bug présent dans Google Chrome pour le rafraîchissement de l'application après une modification. Effectivement, l'application ne semblait pas être modifiée alors qu'en vérité oui. Le problème fut résolu en utilisant Safari plutôt que Chrome pour le développement.

L'une des plus grosses difficultés a été de s'adapter à l'architecture qu'impose Ionic, mais aussi des éléments graphiques qui sont proposés par défaut.

L'ajout de la localisation a été aussi un obstacle important, il fallait pouvoir tester sur un smartphone car elle ne fonctionne par sur un navigateur d'ordinateur (il n'y a pas le périphérique de localisation qu'utilise Cordova). J'ai donc utilisé Ionic View sur mon iPhone afin de simuler le déploiement de l'application et enfin pouvoir utiliser la géolocalisation.

Conclusion personnelle et améliorations

Je pense que l'application pourrait être plus « design » notamment en utilisant d'autres composants que celui utilisé pour représenter les sondages (qui est une liste). Le système permettant de slider entre les choix du sondage peut poser des problèmes ergonomiques, car d'un premier coup d'oeil, on ne voit pas que c'est un slider.

AURÉLIEN CURTI
aurelien.curti@etu.unice.fr

Technologie

Angular 2

Application

Système de sondage en ligne

Description

L'utilisateur se connecte sur le site puis est redirigé vers l'interface de vote qui est l'affichage par défaut du site, qui lui permet de voter pour une liste de sondages qui lui sont proposés et d'exprimer son vote parmi une liste de propositions données.

Si l'utilisateur dispose des privilèges d'administrateur, il aura également accès à une autre interface qui va lui permettre de gérer les différents sondages disponibles, c'est-à-dire d'en créer de nouveaux ou d'en supprimer, de modifier ceux déjà existants et de gérer les propositions de réponses associées à chacun des sondages.
Il peut également visualiser des informations sur les votes déjà effectués comme le nombre de personnes ayant voté pour un sondage et également le nombre de votes attribués à chacune des propositions du sondage.
Enfin, il a la possibilité de réinitialiser les statistiques de vote des différents sondages.

Adaptations

- S'adapter au type d'utilisateur visé (rôle utilisateur) ;

Environnement de l'installation

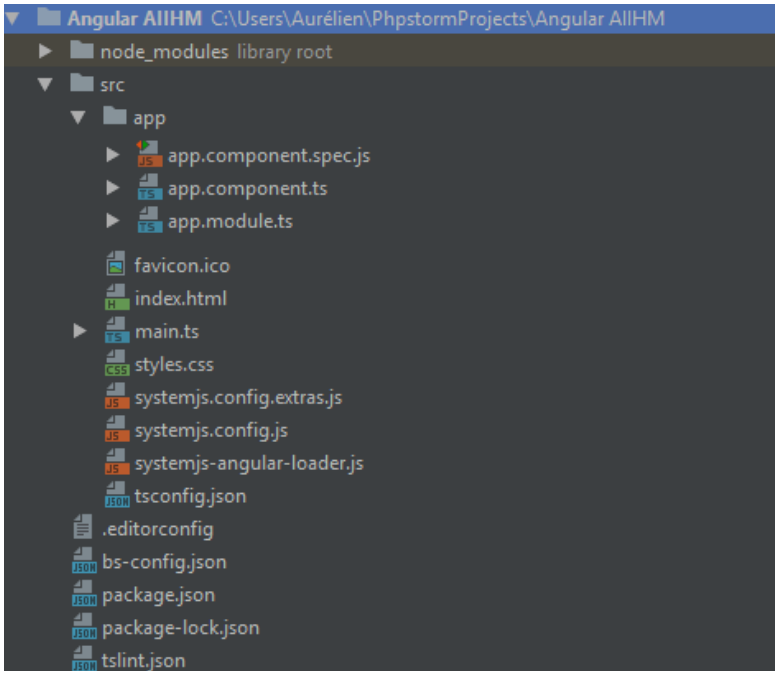
La technologie Angular 2 sera déployé sous un environnement Raspbian, où node et npm ont été préalablement installés et mis à jour.

Installation d'Angular 2

Les fichiers de déploiement d'Angular 2 étant sous un répertoire GitHub, nous clonons ce répertoire dans un répertoire local que nous nommerons 'angular-quickstart' et qui deviendras la racine de notre projet Angular 2, cette opération est réalisée par les commandes suivantes :

git clone https://github.com/angular/quickstart.git angular-quickstart
cd quickstart
npm install
npm start

A ce stade, l'arborescence de notre projet sera décrite comme ceci :



Dans un terminal, à la racine de notre projet angular 2 nous exécutons la commande :
npm start

Ce qui aura pour effet de lancer le serveur et de nous permettre d'accéder à l'interface WEB du projet depuis l'URL http://localhost:3000 et de nous assurer que l'installation s'est correctement déroulée :

Développement de la solution sous Angular 2

Mise en place du modèle globale

Le dossier 'node_modules' contiendra tous nos modules externes qui seront installés via la commande npm et les autres modules nécessaires au fonctionnement d'angular 2, comme typescript par exemple.
Ce dossier étant géré automatiquement, nous nous en occupons plus.

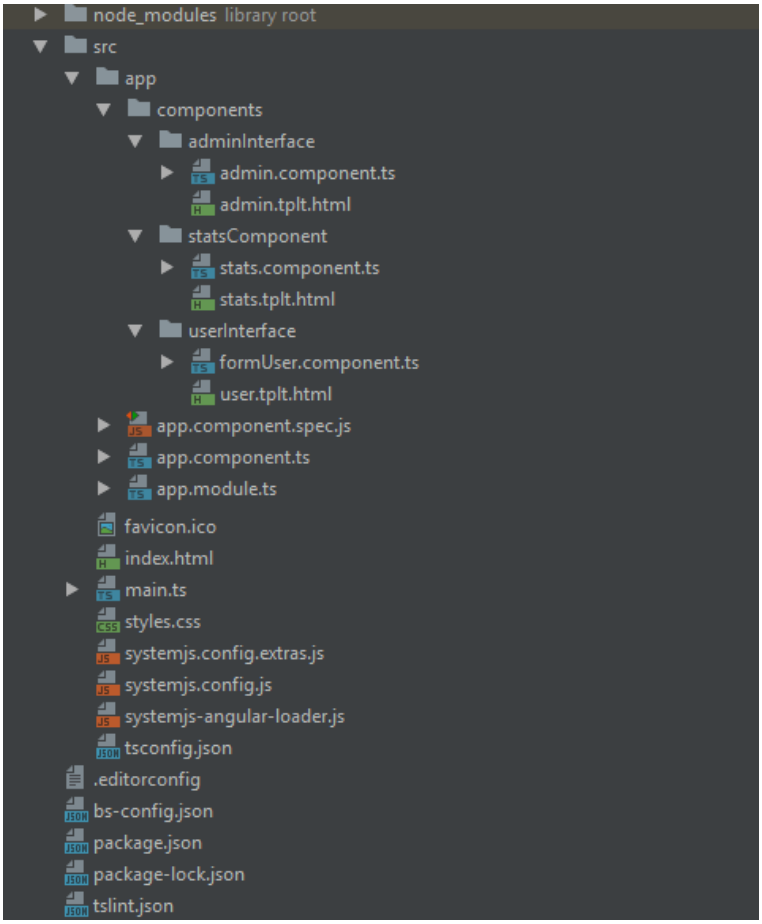
En ce qui concerne le dossier 'src', c'est dans ce dossier que nous allons développer notre solution, une des principales utilité d'Angular 2 est de permettre un développement en brique logiciel que sont les composants. L'avantage de ces composants est qu'ils répondent à un besoin spécifique et peuvent être réutilisable à souhait, tout dépend de la manière dont ils ont été implémentés, permettant ainsi un gain de temps conséquent.

Le but de notre adaptation est de permettre à un composant de s'adapter en fonction du rôle de l'utilisateur qui est en train d'utiliser l'application de vote en ligne.
Nous avons décrit deux composants distincts qui gèrerons de manière individuel l'utilisateur et l'administrateur du système de vote.
Ces deux composants seront le composant 'adminInterface' pour le manager et le composant 'userInterface' pour l'utilisateur.
Le troisième composant qui nous intéresse fortement sera le composant 'statsInterface' qui, en fonction des données que vont lui transmettre les autres composants qui vont interagir avec lui, s'adapter en conséquence.

Ajout et configuration de nos composants sous Angular 2

Création des composants

Sous Angular 2, un composant ne peut être utilisable que s'il est correctement importé et bien-sûr correctement créé.
Dans notre projet, sous le répertoire 'src' nous créons un sous-répertoire 'components', qui va contenir les dossiers de tous les composants de notre application.
Un composant est constitué principalement d'un fichier typescript et d'un template HTML que nous créons également. Nous nous retrouvons ainsi avec l'arborescence suivante :



Une fois que la structure est définie, il nous reste à indiquer à Angular 2 qu'il s'agit de composants, pour ce faire, nous importons Component depuis Angular 2 et utilisons le décorateur @component qui va nous permettre cela.
Puis nous indiquons quelle balise HTML nous pourrons utiliser afin d'invoquer le composant en question.
Enfin, nous renseignons le chemin menant vers le fichier du template HTML que le composant sera chargé d'utiliser.
Par exemple, notre composant userInterface sera déclaré de la manière suivante :

```
import { Component } from '@angular/core';
```

```
@Component({
    selector: 'survey',
    templateUrl: './user.tplt.html'
})
```

Afin de pouvoir l'utiliser dans un autre composant, il est important d'exporter la classe qu'il implémente.
Le corps de notre composant sera donc le suivant :

```
export class FormUser {

}
```

A cette étape, il nous reste plus qu'à définir les attributs de classe de notre composant et les méthodes qu'il contient.

Importation de nos composants

Afin qu'un composant puisse entrer en intéraction avec d'autre composants au sein d'Angular 2, il est nécessaire de l'importer dans le fichier app.module.ts.
Dans notre cas, voici les lignes qu'il est nécessaire d'ajouter ou de modifier depuis le fichier app.module.ts :

```
import {FormUser}  from './components/userInterface/formUser.component';
```

```
declarations: [ AppComponent, FormUser],
```

Sous Angular 2, la procédure reste identique pour tous les autres composants.

De plus, j'ai choisi d'utiliser le routeur d'Angular 2 pour spécifier des routes distinctes vers le composant utilisateur et une autre vers le composant du manager.
Nous créons dans le répertoire app, un fichier app.routing.ts dont le contenu sera le suivant :

```
import {ModuleWithProviders} from '@angular/core';
import {Routes, RouterModule} from '@angular/router';

import {FormUser} from './components/userInterface/formUser.component';
import {AdminEditor} from './components/adminInterface/admin.component';

const appRoutes: Routes = [
    {
        path: '',
        component: FormUser
    },
    {
        path: 'adminPanel',
        component: AdminEditor
    }
];

export const routing: ModuleWithProviders = RouterModule.forRoot(appRoutes);
```

Puis nous modifions le template utilisé dans le fichier app.component.ts pour pouvoir utiliser le routeur :

```
    template: `<router-outlet></router-outlet>`
```

Enfin, on modifie le fichier app.module.ts et importons le module routing :

```
import {routing} from './app.routing';
@NgModule({
  imports:      [ BrowserModule, FormsModule, routing, Ng2GoogleChartsModule ],
  declarations: [ AppComponent, FormUser, AdminEditor, Stats ],
  providers: [CookieService],
  bootstrap:    [ AppComponent ]
})
```

Implémentation des composants et mise en place de l'adaptation

Implémentation des composants

Chaque composant étant maintenant correctement configuré et importé dans notre projet Angular 2, il nous reste à définir le corps de nos classes.
Commençons par le composant userInterface, son but est de permettre à l'utilisateur de visualiser un sondage et les propositions qu'il contient afin d'y répondre.

Voici par exemple le code de classe FormUser du fichier formUser.component.ts, ainsi que son template HTML (user.tplt.html):

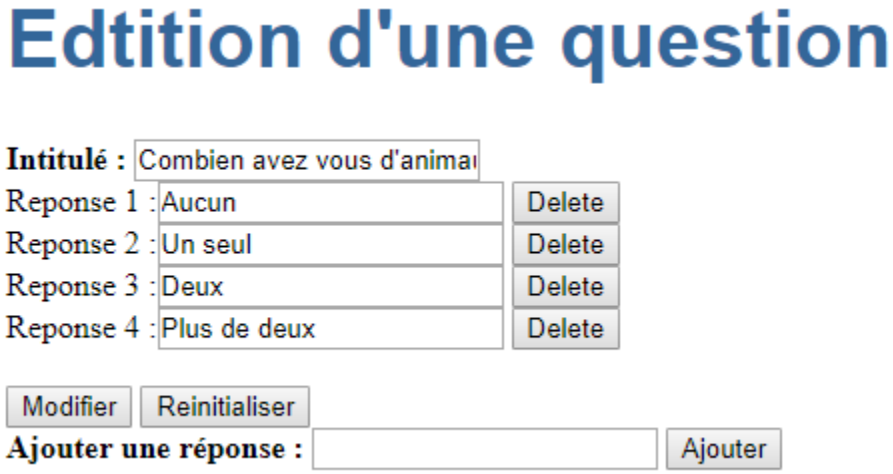
```
export class FormUser {
    title_page: string;
    survey_question: string;
    survey_proposals: string[];

    constructor(){
        this.title_page = 'QUESTIONNAIRE';
        this.survey_question = 'Combien avez vous d\'animaux de compagnie ?';
        this.survey_proposals = [
            'Aucun',
            'Un seul',
            'Deux',
            'Plus de deux'
        ];
    }

    sendResponse(){
        alert('Reponse soumise');
    }
}
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1> {{title_page}}</h1>
    <h3> {{survey_question}}</h3>
    <div *ngFor="let proposal of survey_proposals">
        <input name="response" id="{{proposal}}" value="{{proposal}}" type="radio">
        <label for="{{proposal}}">{{proposal}}</label>
    </div>
    <button (click)="sendResponse()">Soumettre</button><br />
    <h2 ><a href="adminPanel">Acceder à l'interface de gestion</a></h2>
</body>
</html>
```

Grace à Angular 2, il nous est possible de « binder » des variables directement dans le template HTML pour en récupérer leurs valeurs provenant du fichier typescript du composant.
Typiquement, cela se fait grâce aux doubles accolades {{MA_VAR}} comme nous le voyons dans le code ci-dessus, pour la variable title_page par exemple.
A cet instant, si nous nous rendons sur la page http://localhost:3000, nous obtenons le rendu suivant :

Le composant du manager, lui, a pour finalité de permettre à une personne avec des droits spéciaux d'y accéder, pour lui permettre de gérer les différents sondages du site.
Ajouter ou supprimer des sondages, ou encore modifier des sondages déjà existants font partie des fonctionnalités de ce composant.
Nous implémentons donc la classe du composant admin.component.ts ainsi que son template HTML admin.tplt.html

Une fois implémenté, le rendu suivant est obtenu sur l'url qui utilise cette fois ci une nouvelle route http://localhost:3000/adminPanel :



Détermination des rôles avec les cookies

Afin de déterminer s'il s'agit d'un simple utilisateur ou d'un manager, nous allons utiliser le module externe angular2-cookie.
Nous l'installons depuis le terminal sous le répertoire racine de notre projet angular 2 grâce à la commande : npm install angular2-cookie
Une fois que toutes les dépendances sont installées, il nous reste à l'importer dans notre projet, en y modifiant le fichier app.module.ts comme ceci :

```
import {CookieService} from 'angular2-cookie/services/cookies.service';

import {routing} from './app.routing';
@NgModule({
  imports:      [ BrowserModule, FormsModule, routing, Ng2GoogleChartsModule ],
  declarations: [ AppComponent, FormUser, AdminEditor, Stats ],
  providers: [CookieService],
  bootstrap:    [ AppComponent ]
})
```

Et également le fichier systemjs.config.js comme ceci :

```
// other libraries
'rxjs':                      'npm:rxjs',
'angular-in-memory-web-api': 'npm:angular-in-memory-web-api/bundles/in-memory-web-api.umd.js',
'angular2-cookie': 'npm:angular2-cookie',
```

```
rxjs: {
  defaultExtension: 'js'
},
'angular2-cookie': {
    main: './core.js',
    defaultExtension: 'js'
}
```

Maintenant, le module externe angular2-cookie est utilisable et nous pouvons l'utiliser dans nos composants après l'avoir importé au préalable dans les composant en question à l'aide de la ligne suivante:

```
import {CookieService} from 'angular2-cookie/core';
```

Le code de notre composant utilisateur a ainsi évolué de la manière suivante en tenant compte du cookie qui va nous permettre d'identifier un simple utilisateur d'un utilisateur qui sera un manager des sondages.

```
export class FormUser {
    title_page: string;
    survey_question: string;
    survey_proposals: string[];
    status_usr: string;

    constructor(private _cookieStatus: CookieService){
        this.title_page = 'QUESTIONNAIRE';
        this.survey_question = 'Combien avez vous d\'animaux de compagnie ?';
        this.survey_proposals = [
            'Aucun',
            'Un seul',
            'Deux',
            'Plus de deux'
        ]
        _cookieStatus.put( key: 'status', value: 'admin');
        this.status_usr = _cookieStatus.get('status');
    }

    sendResponse(){
        alert('Reponse soumise');
    }
}
```

Mise en place de l'adaptation

Enfin, concernant l'adaptation, un troisième composant a été ajouté à l'application, il s'agit du composant des statistiques des votes et qui implémente le module externe ng2-google-charts.

Son installation dans notre projet angular 2 s'effectue depuis le terminal grâce à la commande suivante :
npm i –save ng2-google-charts

Une fois installé, nous devons le déclarer dans le fichier de configuration systemjs.config.js :

```
// other libraries
'rxjs':                      'npm:rxjs',
'angular-in-memory-web-api': 'npm:angular-in-memory-web-api/bundles/in-memory-web-api.umd.js',
'angular2-cookie': 'npm:angular2-cookie',
'ng2-google-charts': 'npm:ng2-google-charts/bundles/ng2-google-charts.umd.min.js'
```

Nous sommes maintenant dans la capacité d'utiliser ce module externe, grâce aux balises HTML suivantes placés dans notre template HTML stats.tplt.html :

```
<google-chart [data]="surveyData"  *ngFor="let surveyData of extsurveyDatasTabs"></google-chart>
```

La classe typescript des statistiques étant implémenté de la manière suivante :

```
import {Component, Input} from '@angular/core';

@Component({
    selector: 'stats',
    templateUrl: './stats.tplt.html'
})

export class Stats {
    public title_page: string;

    constructor() {
        this.title_page = 'Statistiques';
    }

    @Input()
     extsurveyDatasTabs: any;
}
```

Afin d'avoir un affichage et une adaptation des statistiques, nous avons simplement à lui passer des données de façon structurés.
C'est-à-dire, en respectant une certaine syntaxe que le module ng2-google-charts est capable d'interpréter, pour en donner un affichage adapté de celles-ci.
Dans l'application, ces données correspondent à la variable extsurveyDatasTabs qui dans le composant des statistiques, utilisera le système d'Input d'angular 2, pour permettre ce transfert de données entre composants.

C’est cette variable qui contiendra la totalité des données transmises depuis le composant de l’utilisateur ou du manager, ainsi seules les données passées au composant chargé d’affiché les statistiques diffèreront, mais le composant lui ne changera pas.
Afin de transmettre des données au composant des statistiques, nous déclarons respectivement dans le composant de l’utilisateur et du manager les données suivantes, que nous stockons dans la variable surveyDatas :

```
surveyDatas =  [{
    chartType: 'PieChart',
    dataTable: [
        ['Reponse', 'Nombre de votes'],
        ['Aucun',     11],
        ['Un seul',      8],
        ['Deux',  2],
        ['Plus de deux', 1],
    ],
    options: {'title': 'Nombre de votes par réponses'},
}];
```

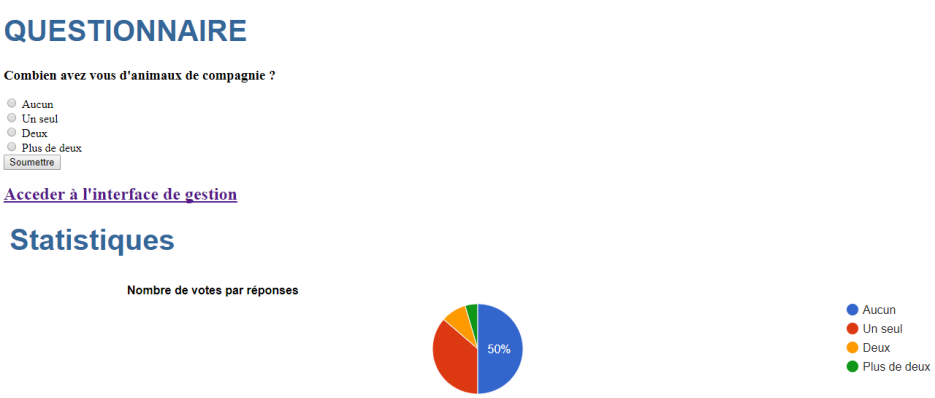
```
surveyDatasTabs =  [
    {
        chartType: 'PieChart',
        dataTable: [
            ['Reponse', 'Nombre de votes'],
            ['Aucun',     11],
            ['Un seul',      8],
            ['Deux',  2],
            ['Plus de deux', 1],
        ],
        options: {'title': 'Répartition des votes', is3D: true},
    },
    {
        chartType: 'ColumnChart',
        dataTable: [
            ['Task', 'Homme', 'Femme'],
            ['Aucun',     9, 2],
            ['Un seul',      5, 3],
            ['Deux',  1, 1],
            ['Plus de deux', 0, 1],
        ],
        options: {'title': 'Comparaison hommes/femmes', colors: ['#1E90FF', '#FF1493']},
    },
    {
        chartType: 'LineChart',
        dataTable: [
            ['Mois', 'Nombre de votants'],
            ['Juillet',     3],
            ['Août',     7],
            ['Septembre',  4],
            ['Obctobre', 8],
        ],
        options: {'title': 'Evolution du nombre de votes au cours des mois'},
    }
];
```

Il nous reste plus qu’à appeler le composant des statistiques depuis nos templates utilisateur et manager de la façon suivante et de lui transmettre la variable qui contient les données à afficher :

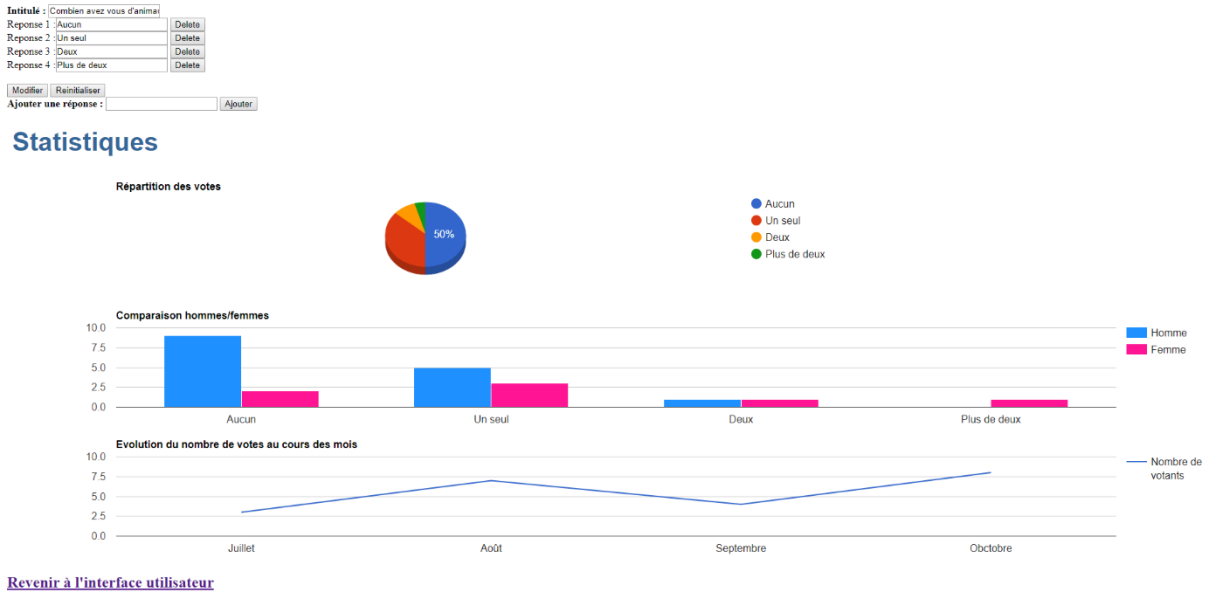
```
<stats [extsurveyDatasTabs] = "surveyDatasTabs"></stats>
```

Voici les deux interfaces que cela nous donne sur nos pages :

Pour l'interface utilisateur



Pour l'interface du manager



Nous voyons que malgré le fait que le composant statistique qui est appelé depuis le composant utilisateur ou manager soit le même, celui-ci n'affiche pas la même chose depuis la page de l'utilisateur que celle du manager, car les deux composants ne lui envoient pas les mêmes statistiques à afficher et donc le composant statistique doit s'adapter en fonction des données qui lui sont envoyés.

ELIA DITMANN
elia.ditmann@etu.unice.fr

Technologie

BootStrap

Application

Système de sondage en ligne

Description

Dans la version Bootstrap de l'application, l'utilisateur qui se connecte sur la page Web a accès à différents boutons permettant de visualiser différentes catégories de votes grâce à un scroller situé sur le côté de la fenêtre. Il peut également visualiser les sondages actuels les plus importants, situés au milieu haut de l'écran sous forme de graphes de proportion.

Une section regroupant les sondages précédents par ordre chronologique décroissant est située sous les sondages actuels, sous forme de tableau.

Pour voter, l'utilisateur clique sur le nom d'un sondage et il est redirigé vers une page lui proposant différents choix de votes. Il valide ensuite son choix grâce à un bouton, son vote est pris en compte et il est redirigé vers la page d'accueil du site.
Un vote est unique et n'est pas révocable, un utilisateur ne peut donc pas voter deux fois ou modifier son vote

Adaptations

- S’adapte aux différents supports d’affichage (responsive) ;
- Accessibilité aux non voyants ;

Une page web contient du texte et des images, mais aussi un certain nombre d'éléments très fréquents : listes, tableaux, formulaires, icônes, boutons. Bootstrap permet de créer un rendu visuel cohérent pour que tous ces éléments cohabitent de façon esthétique.

Installation des outils

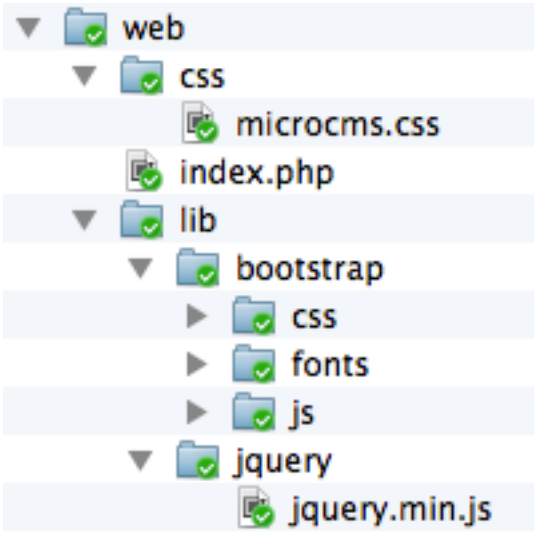
Bootstrap est un framework client (frontend), orienté CSS mais pas seulement puisqu'il embarque également des composants HTML et JavasScript. Il comporte un système de grille simple et efficace pour mettre en ordre l'aspect visuel d'une page web. Il apporte du style pour les boutons, les formulaires, la navigation… Il permet ainsi de concevoir un site web rapidement et avec peu de lignes de code ajoutées.

L'installation est simple : Il suffit de télécharger le framework sur le site de Bootstrap, puis de récupérer les principaux fichiers .css et .js. Les fichiers bootstrap-theme.css.map et bootstrap.css.map permettent de retrouver l'emplacement original d'une ligne de code à partir du code minifié. Cette fonctionnalité est utilisable avec les dernières versions de Chrome et Firefox. Ces fichiers ne sont cependant pas indispensables au fonctionnement de Bootstrap.

Architecture

L'architecture de BootStrap est basée sur LESS, un outil bien pratique qui étend les possibilités de CSS (un portage sur SASS existe également).

L'arborescence de BootStrap ressemble à ceci :



Pour que Bootstrap fonctionne il faut le déclarer dans les pages HTML, qui doivent être impérativement au format HTML 5, il faut donc prévoir le bon DOCTYPE.
Il faut ensuite déclarer au minimum le fichier bootstrap.min.css (ou bootstrap.css) dans l'en-tête de la page web :

```
<head>
        …
   <link href="bootstrap/css/bootstrap.min.css" rel="stylesheet">
</head>
```

À partir de là toutes les classes sont accessibles, à condition d'adapter le lien selon la localisation de vos fichiers.

```
<body>
  <nav class="navbar navbar-toggleable-md navbar-inverse fixed-top bg-inverse" style="background-color: #292b2c!important;">
    <button class="navbar-toggler navbar-toggler-right hidden-lg-up" type="button" data-toggle="collapse" data-target="
    #navbarsExampleDefault" aria-controls="navbarsExampleDefault" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <a class="navbar-brand" href="#">SondeMoi</a>

    <div class="collapse navbar-collapse" id="navbarsExampleDefault">
      <ul class="navbar-nav mr-auto">
```

Implémentation du code

BootStrap propose une mise en page basée sur une grille de 12 colonnes de bases (nombre modifiable en fonction des besoins). Il faut donc parfois réserver un espace afin de jouer avec les contraintes de cette grille, par exemple pour une barre de navigation en haut de page avec "padding-top" afin d'éviter que le texte du début ne se retrouve sous la barre.

On trouve ici une classe qui fixe un espace interne et un alignement centré du texte. On trouve aussi dans la page une trame de la barre de navigation

Le contenu de la page est ensuite inséré dans une DIV comportant la classecontainer:

```
<div class="container-fluid">
  <div class="row">
    <nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
      <ul class="nav nav-pills flex-column">
        <li class="nav-item">
          <button aria-label="Catégorie Overview" type="button" class="btn btn-link">Overview</button>
        </li>
        <li class="nav-item">
          <a aria-label="Catégorie Reports" class="nav-link" href="#">Reports</a>
        </li>
        <li class="nav-item">
          <a aria-label="Catégorie Analises" class="nav-link" href="#">Analytics</a>
        </li>
        <li class="nav-item">
          <a aria-label="Catégorie Export" class="nav-link" href="#">Export</a>
        </li>
      </ul>
```

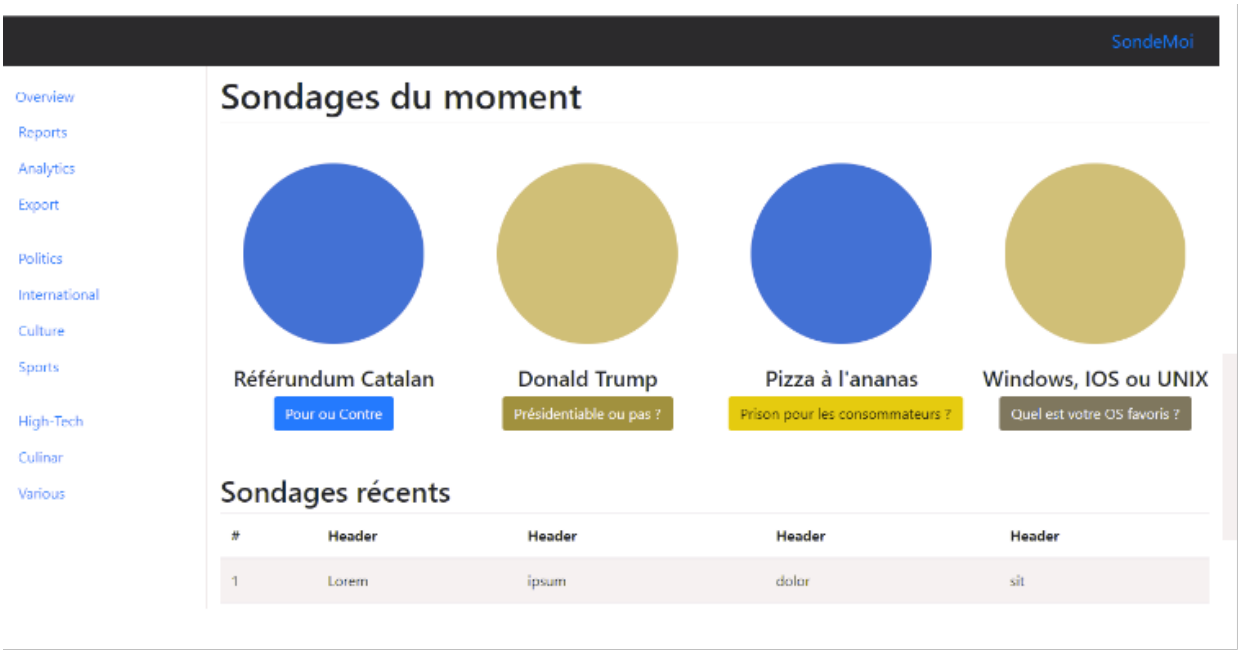
Comment tester les adaptations ?

Afin de tester l'adaptation du site aux personnes mal voyantes, il convient avant tout d'activer le narrateur audio, ici celui intégré au système d'exploitation Windows 10. Pour cela, aller dans le "Panneau de configuration", cliquer sur "Facilité d'accès" puis "Options d'ergonomie". "Sélectionner Explorer tous les paramètres" et "Utiliser l'ordinateur sans moniteur".

Une fois le narrateur activé, il est possible d'interagir avec les éléments cliquables tels que des boutons ou des liens, et le narrateur lira le contenu de la balise Label, donnant ainsi des informations vocales pour les personnes ayant des difficultés à lire des textes.

Les autres adaptations étant de nature visuelle, par exemple le contraste plus élevé des boutons ou le code couleur adapté à une meilleur visualisation, il convient de faire tester l'interface à des personnes atteintes de déficience visuelle et d'apporter des modifications en fonction des retours utilisateurs.

Ayant personnellement des problèmes de vues, j'ai pu me charger moi-même tester les différents composants et apporter des modifications adaptées, rendant ainsi les élément plus visibles. Il existe cependant des outils permettant de tester l'adaptation d'éléments visuels, tels que le CBC (ColorBlindessCheck) qui permet de simuler la dyschromatopsie (mauvaise perception de certaines tonalités, telles que le vert ou le bleu). Le site "http://www.color-blindness.com/coblis-color-blindness-simulator/" m'a permis de visualiser ma page avec différentes anomalies visuelles telles que la monochromie ou le daltonisme. J'ai ainsi pu procéder au choix des couleurs de ma page afin de la rendre la plus accessible possible.
Ci-dessous, un exemple de page avec simulation de dichromatopsie sur la couleur rouge.



Le fait d'avoir doublé l'information visuelle avec une aide sonore (ici le narrateur vocal) permet une compréhension maximale des différents éléments interactifs.

Difficultés rencontrées

Comme tout framework, il est nécessaire de bien connaitre BootStrap pour l'utiliser efficacement. La courte période d'implémentation a nécessité de se focaliser sur certains éléments importants, en passant à côté d'autres fonctionnalités du framework. BootStrap est cependant rapide à prendre en main, et il a donc été

rapide de débuter l'implémentation des différentes fonctionnalités, telles que l'ajout des labels pour le narrateur audio intégré au système d'exploitation.

La difficulté principale a été, ici, de s'adapter à la contrainte des 12 colonnes afin de rendre esthétique l'aspect global de mes pages web, il convient donc de réfléchir en terme de découpage et de prendre en compte la totalité des éléments affichés à l'écran. Il est cependant possible de modifier cette contrainte, c'est à dire de modifier le nombre de colonnes, mais il faudra dans tous les cas adapter la mise en page.

Conclusion personnelle et améliorations

La majeur partie du côté "Adaptation" a consisté pour ma part à rechercher ce qui existait dans le domaine des interfaces orientées aux mal voyants afin de partir d'une base cohérente. Quelles couleurs, quels éléments ou fonctionnalités importantes sont requises ? J'ai testé de nombreux échantillons de couleurs, principalement sur le fond et les boutons, afin le rendu soit assez contrasté compréhensible. Par exemple, les boutons jaunes avec écriture noire ont été sélectionnés pour la page principale car parmi les différents tests que j'ai effectué, c'est le résultat qui m'a semblé le plus judicieux car plus lisible. Certains de ces éléments ont été implémentés, d'autres pas, que ce soit pour des raisons de priorités du cahier des charges ou de pertinence. J'ai par exemple choisi de ne pas intégrer de système de grossissement des textes puisque les navigateurs possèdent déjà un système de grossissement (généralement avec ctrl + molette).

Le contenu final de la page web n'étant qu'un prototype, il est évidemment perfectible et pourrait être amélioré de diverses façons, par exemple en implémentant un système de configuration en fonction de l'utilisateur, pour que le contenu soit adapté à leur déficience éventuelle (Daltoniens par exemples). Cependant, le fait d'avoir doublé un élément visuel tel qu'un bouton coloré avec un synthétiseur vocal me semble à la fois adapté et pertinent en vue de l'adaptation envisagée.