Adaptation des interfaces à l'environnement

Rapport n°2

Groupe 3

Sommaire :

Introduction

Créer ou faire évoluer une IHM et l'adapter au contexte d'usage est l'un des enjeux majeurs du développement d'IHM. Pour cela, l'interface réalisée doit s'adapter aux différents supports, d'utilisateurs ou même environnements afin d'avoir la meilleure expérience utilisateur possible. Enfin d'expérimenter sur les changements nécessaires demandés par l'adaptation nous étudierons ici plusieurs façons de réaliser une interface pour l'organisation et la visualisation d'événements comme des manifestations ou des fêtes privées.
Tous les membres du groupe étudierons une technologie différente en cherchant à tester les capacités d'adaptation et leurs limites. Les interfaces créées devront permettre une continuité de service peu importe le contexte d'utilisation et pouvoir s'adapter au système d'exploitation utilisé ainsi qu'aux capteurs présents ou non sur les terminaux.

Tutoriaux

Tutoriel sur la partie Cross Platform (PhoneGap) :

L'exemple réalisé correspond à un scénario dans lequel un utilisateur va utiliser plus d'un smartphone (car le smartphone principal n'aura plus de batterie au bout d'un moment) afin de savoir où se rendre pour un événement organisé donné (donc une destination fixe) en fonction de sa position actuelle. Il a donc été nécessaire de situer l'utilisateur sur une carte du monde ainsi qu'afficher un itinéraire pour qu'il puisse se repérer, et ce de manière adaptée à l'OS en réalisant une application Cross Platform.

1) Réalisation de l'exemple

1.1) Installation de PhoneGap Desktop

La première étape pour pouvoir réaliser l'exemple est d'ouvir un navigateur sur son ordinateur et d'aller sur le lien http://docs.phonegap.com/getting-started/1-install-phonegap/desktop/ afin de télécharger le logiciel PhoneGap Desktop, qui permet de fournir un serveur web et de lancer l'exemple sur ce serveur.
Il faut après accès au site choisir son OS, télécharger le logiciel d'installation correspondant à l'OS et suivre les étapes indiquées sur le site web qui consistent principalement à accepter l'accord de licence et à appuyer sur le bouton Next (Suivant) jusqu'au bouton Finish (Terminer) pour conclure l'installation.

1.2) Installation de PhoneGap Developer

La deuxième étape consiste à ouvrir l'App Store ou le Google Play Store sur son appareil mobile et de rechercher le mot clé "PhoneGap" dans la partie recherche d'applications, puis d'installer l'application PhoneGap Developer qui est à ce jour le premier résultat dans la liste.

1.3) Création d'un projet PhoneGap

Pour la troisième étape il faut lancer l'application PhoneGap Desktop (ordinateur); une fois lancée se trouve un signe + en haut à gauche de la fenêtre sous l'onglet PhoneGap, qui permet de créer un nouveau projet phonegap avec ses dépendances et plugins de base. Il faut après avoir appuyé sur le + appuyer sur "Create new PhoneGap project" > "Blank" > entrer un nom de projet et un emplacement pour le disque. Appuyer ensuite sur le bouton "Create Project" pour terminer la création d'un projet vide.

1.4) Ajout des différents plugins

PhoneGap met à disposition dans ses templates de projet des plugins par défaut, mais pour l'exemple choisi il faut rajouter certains plugins afin de faire fonctionner la localisation sur les différents OS majeurs. Il va falloir ainsi installer Cordova qui est le moteur sur lequel tourne PhoneGap, mais qui ici sera seulement utilisé pour installer des plugins en ligne de commande.
Ceci nécessite d'avoir Node.js car cordova s'installe en tant que module avec npm; si Node.js n'est pas installé sur l'ordinateur, il se trouve sur https://nodejs.org/en/. La version 6 est suffisante pour installer Cordova qui s'installe après avoir installé Node.js (en acceptant la licence et en appuyant sur Next) par la commande *npm install -g cordova*.

Une fois Cordova installé, il faut se rendre en ligne de commande via la commande *cd nomdurépertoire* jusqu'au répertoire où se trouve le projet PhoneGap, et entrer la commande *cordova plugin add cordova-plugin-geolocation.* Ce plugin sera installé pour les différents OS tels que iOs, Android et Windows.

1.5) Codage d'une carte basique avec l'API Google Maps

Une fois le plugin installé il est nécessaire de créer une carte, le choix s'est porté ici sur Google Maps pour sa facilité d'utilisation. Il est nécessaire avant de pouvoir utiliser l'API Google Maps d'aller sur le lien https://developers.google.com/maps/documentation/javascript/get-api-key#key afin de pouvoir générer une clé d'API qui permettra l'utilisation gratuite des fonctionnalités Google Maps, il suffit pour se faire de suivre les étapes indiquées jusqu'à obtenir une clé sur la console développeur de Google. La carte s'initialise ensuite dans le code HTML comme suit :

```
<div id="map"></div>
<script async defer
    src="https://maps.googleapis.com/maps/api/js?key=AIzaSyChTBmYeyWhmqqNnMRtFB2L_anoudcpbEo">
</script>
```

La clé (personnelle) se trouve ici après *key=*, et permet dans l'exemple ci-dessus de faire des appels asynchrones au serveur pour pouvoir récupérer des informations quand demandées.
La carte n'est pas encore affichable avec ce seul bout de code, il faut d'abord lui donner un affichage par défaut via un script au cas où les requêtes venaient à échouer :

```
var map = new google.maps.Map(document.getElementById('map'), {
    center: {lat: 44, lng: 7},
    zoom: 7
});
```

Ce code permet d'instancier une carte centralisé sur les Alpes-Maritimes, qui est le lieu où sera testée l'application.

1.6) Utilisation du service de géolocalisation pour trouver sa position

La première étape pour réaliser le scénario imaginé est de situer l'utilisateur sur la carte, en effectuant une requête de géolocalisation au navigateur via le plugin geolocation. Le partie clé du code demande la position au navigateur et en extrait les coordonnées de latitude et longitude :

```
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(function(position) {
        departurePos.push(position.coords.latitude);
        departurePos.push(position.coords.longitude);
        // console.log("Departure found");
    });
}
```

Ces coordonnées sont ensuite ajoutées à un tableau contenant les informations sur le point de départ de l'itinéraire.

1.7) Utilisation du service de conversion d'adresse en coordonnées

La deuxième étape pour réaliser le scénario imaginé est d'identifier les coordonnées d'une adresse fournie (l'adresse où se situe l'événement) grâce au service de décodage de l'API :

```
geocoder.geocode({'address': "10 Bd Comte de Falicon 06100 Nice"}, function(results, status) {
    if (status === 'OK') {
        arrivalPos.push(results[0].geometry.location.lat());
        arrivalPos.push(results[0].geometry.location.lng());
        // console.log("Arrival found");
    }
});
```

Ces coordonnées sont ensuite ajoutées à un tableau contenant les informations sur le point d'arrivée de l'itinéraire.

1.8) Utilisation du service de direction pour établir l'itinéraire

Avant de réaliser la troisième étape il est nécessaire de réaliser que les deux requêtes ci-dessus sont asynchrones, et si une troisième requête est effectuée juste après il est possible que la position de départ ne soit pas encore dans le tableau. Il faut alors bloquer le code jusqu'à la terminaison des deux requêtes; plusieurs méthodes sont possibles et j'ai choisi de bloquer l'exécution du code jusqu'à terminaison des requêtes. Il faut pour se faire déclarer la fonction dans laquelle ce code se trouve comme *async,* et insérer deux variables locales de verrou : lock1 et lock2 initialisées à true en début de fonction, qui passeront à false une fois chaque requête complétée avec *lock1 = false;* au dessus de *console.log("Departure found");* et *lock2 = false;* au dessus de *console.log("Arrival found");*. Une fois ces modifications effectuée il suffit de rajouter la boucle bloquant l'exécution :

```
while (lock1 == true || lock2 == true) {
    await sleep(1000);
}
```

Une fois l'attente terminée, il ne reste plus qu'à effectuer la requête d'itinéraire avec les positions obtenues :

```
directionsDisplay.setMap(map);

directionsService.route({
    origin: {lat: departurePos[0], lng: departurePos[1]},
    destination: {lat: arrivalPos[0], lng: arrivalPos[1]},
    // Note that Javascript allows us to access the constant
    // using square brackets and a string value as its
    // "property."
    travelMode: google.maps.TravelMode["DRIVING"]
}, function(response, status) {
    if (status == 'OK') {
        directionsDisplay.setDirections(response);
    } else {
        window.alert('Directions request failed due to ' + status);
    }
});
```

Ce code permet d'afficher non seulement l'itinéraire mais aussi des marqueurs sur la carte indiquant où se trouve l'utilisateur et sa destination, ainsi que leurs adresses.

1.9) Utilisation du service de géolocalisation pour la boussole

La mise en place de la boussole est plus simple à mettre en oeuvre : à partir d'une * * dans le code HTML, il suffit de demander des informations sur la position courante afin de recevoir les informations sur le heading de l'appareil conformément à la nouvelle norme du W3C depuis Mai 2017. Les informations sont ensuite affichées dans la balise div.

```
var onSuccess = function(position) {
    var direction = position.coords.heading;
    var text;
    if (direction>0 && direction<90) text = "NE";
    else if (direction>90 && direction <180) text = "SE";
    else if (direction>180 && direction<270) text = "SW";
    else if (direction>270 && direction<360) text = "NW";
    else text = "";
    document.getElementById('heading').innerHTML = "Direction: " + direction + " " + text;
}

function onError(error) {
    document.getElementById('heading').innerHTML = "No compass";
}

navigator.geolocation.getCurrentPosition(onSuccess, onError);
```

2) Outil de développement et de tests

2.1) Outils de développement

Le développement du code de l'exemple s'est fait entièrement avec un simple éditeur texte : Sublime Text 3, en effet il n'y a pas vraiment d'IDE nécessaire pour développer en Javascript/HTML5/CSS3 mais certains éditeurs facilitent l'écriture et la reconnaissance de bugs liés à l'écriture du code avant de compiler comme ST3.

2.2) Outils de test

Les outils de test ont été les suivants : tout d'abord un ordinateur portable Lenovo Legion Y720 (écran 38.0 x 27.7 cm - navigateurs Firefox et Chrome) et un iphone 5 (écran 6.40 x 11.36 cm - navigateur Safari) pour tester la version de base avec la géolocalisation qui ne demande que la position courante de l'utilisateur. A été utilisé ensuite un Motorola G3 2015 (écran 7.24 x 14.2 cm - navigateur Chrome) sous Android afin de vérifier la compatibilité et l'efficacité du Cross Platform avec les OS majeurs du marché.

3) Déploiement et exécution de l'exemple

Pour pouvoir déployer l'exemple sur n'importe quelle plateforme il faut tout d'abord lancer le programme PhoneGap Desktop sur ordinateur et appuyer sur la flèche dans la zone à droite du

nom du projet; cet appui déclenche le lancement du serveur web local qui fera tourner l'application.

3.1) Déploiement sur ordinateur

Pour pouvoir déployer l'exemple sur ordinateur il faut lancer un web serveur local autre que PhoneGap Desktop, car le navigateur n'autorise les requêtes de géolocalisation que si la requête est effectuée dans un contexte sécurisé ce que ne fait pas PhoneGap Desktop. Il faut donc utiliser un web serveur plus adapté, comme par exemple l'extension du navigateur Chrome appelée Web Server for Chrome (qui une fois lancée fonctionne pour tous les navigateurs, malgré le nom). Une fois le web server positionné sur le dossier de l'exemple, la page web de l'exemple est alors accessible depuis l'IP fournie par le web server.

3.2) Déploiement sur mobile

Pour pouvoir déployer l'exemple sur mobile il faut connecter le mobile sur le même réseau WiFi que l'ordinateur, puis lancer l'application PhoneGap Developer sur mobile et entrer l'IP du serveur indiquée sur PhoneGap Desktop. Une fois l'IP entrée et validée, l'application est prête à être exécutée.

3.3) Exécution de l'exemple

L'exécution de l'exemple se fait alors naturellement, la page web de départ (par défaut index.html, changeable dans le fichier config.xml) se lance dans le navigateur web à l'adresse IP locale indiquée et/ou dans le viewport sur mobile une fois l'IP validée.

4) Comment tester les capacités d'adaptations

Les capacités d'adaptations étudiées ici, liées aux problématiques du Cross Platform qui fournit des solutions pour l'adaptation à la conception et à la plateforme, seront étudiées selon deux axes différents permettant de montrer avec le premier un des avantages puis avec le deuxième un des inconvénients de PhoneGap.

4.1) Premier axe d'adaptation : géolocalisation

En partant de la page web de départ présentant un calendrier, il faut cliquer sur le jour numéro 10 en rouge pour ouvrir la page correspondant à l'événement du jour numéro 10. En bas de la page qui s'affiche se trouve un bouton "Localize Event" qu'il faut cliquer et qui ouvre une page où va s'afficher la carte montrant la position courante sur la carte de l'appareil utilisé, la position de l'adresse indiquée dans l'événement et la visualisation du chemin à prendre en voiture pour aller de la position courante jusqu'à la destination.
Cet exemple comprend trois requêtes : une pour la position courante, une pour la destination, une pour le tracé de l'itinéraire sur la route et chacune de ces requêtes est asynchrone ce qui implique qu'il n'y a pas d'attente de la terminaison du calcul du point de départ et d'arrivée avant de commencer le calcul de la trajectoire ce qui peut causer des erreurs, c'est pourquoi il est nécessaire d'avoir une adaptation pour chaque plateforme en fonction du temps de réponse

à la requête de position courante (la conversion d'une adresse en une position se terminant toujours avant) afin de toujours afficher le résultat désiré.
Dans un premier temps une solution a été d'avoir une valeur différente pour chaque plateforme sur setTimeout qui permettait de retarder la requête de trajectoire, mais une façon plus globale de gérer le problème est d'attendre le temps que les deux premières requêtes se finissent avant de lancer la dernière sur la trajectoire.
On remarque aussi ici que pour d'autres technologies Cross Platform (par exemple Xamarin) il est nécessaire pour accéder aux capteurs de l'appareil d'utiliser du code spécifique à l'OS visé, or il n'est pas nécessaire avec PhoneGap dans ce cas d'adapter le code de cette manière car l'accès aux capteurs de géolocalisation se fait à travers le navigateur qui lui a accès aux capteurs des différents OS après une demande de confirmation à l'utilisateur par un message d'alerte.

4.2) Deuxième axe d'adaptation : boussole

Un deuxième aspect de l'adaptation a été ici étudié : il est accessible via le bouton GO sous la carte qui s'affiche avec l'exemple précédent. Ce bouton GO était dans un premier temps un appel au plugin device-orientation de phonegap mais qui a été déprécié, c'est donc à présent un appel au moyen recommandé par la norme du W3C depuis Mai 2017 pour obtenir le heading de l'appareil : le plugin geolocation, via le champ heading de la réponse retournée par *navigator. getCurrentPosition( ).*
La particularité de cette spécification est qu'elle est très récente et n'est pas implémentée par tous les appareils, ce qui donne un résultat null sur iPhone 5 par exemple. Sur l'ordinateur, le dispositif est reconnu en tant que stationnaire et le résultat est alors Not a Number (NaN); on voit ici qu'en fonction du support, il faudrait adapter le code pour soit implémenter une autre solution (par exemple le plugin cordova déprécié) ou alors adapter l'affichage selon si oui ou non la fonctionnalité est implémentée car avoir un bouton qui renvoie null ou NaN n'est pas utile et peut même apporter de la confusion à l'utilisateur.

Tutoriel sur la partie Web Components (Angular) :

Ce tutoriel a pour but de montrer comment réaliser une application web Angular codée en typescript, "EventManager", qui est capable de s'adapter à l'appareil utilisé.

À la fin de ce tutoriel, l'application doit être capable de répondre au scénario d'utilisation suivant:

Marc participe à plein d'événements. À la maison, il organise son emploi du temps en fonction de tous ces événements, du coup il aime bien avoir une vue d'ensemble sur son dashboard (browser PC). Au travail, où il n'a pas accès à un ordinateur, il veut simplement pouvoir vérifier ou consulter des informations sur deux événements en particulier (parce qu'ils arrivent bientôt ou parce qu'il veut en parler à ses collègues) - ça ne l'arrange donc pas d'avoir à les trouver

parmi tous les autres, particulièrement sur le petit écran de son mobile. Du coup, il voudrait pouvoir personnaliser sa vue mobile pour qu'elle affiche les éléments qui lui sont pertinents en premier.

On va utiliser npm start pour déployer l'application. Pour tester l'adaptation sur mobile, en utilise la vue adaptative de l'outil de développement du browser.

1) EventManager: l'application de base

Le tutoriel présuppose que vous avez déjà une application "EventManager" de base qui fonctionne. Pour mettre en place l'environnement Angular, vous pouvez suivre les instructions de ce guide: https://angular.io/guide/quickstart.

L'application de base sur laquelle nous allons travailler est construite de la manière suivante:

- Le dossier src/app contient les éléments suivant:
    - Le dossier components : ce dossier comprend tous les fichiers où l'on défini les différents composants de l'application.
    - Le dossier entities : ce dossier comprend tous les fichiers .ts où sont défini les objets avec lesquels nous allons travailler. Pour l'instant, il ne s'agit que de "event.ts".
    - Le dossier services : ce dossier comprend tous les fichiers .service.ts où sont défini les services utilisés par l'application.
    - Le dossier styles : ce dossier comprend tous les fichiers css où sont défini les styles.
    - Le dossier view : ce dossier comprend tous les fichiers html qui représente les vues de l'application.
    - Le fichier app.module.ts : ce fichier représente le module "root" où l'on définit comment assembler l'application. C'est là que sont déclarés tous les composants de l'application (ainsi que les imports et les providers).
    - Le fichier app-rooting.module.ts : c'est dans ce fichier que sont déclarées toutes les routes de l'application - à quel composant est associé un path particulier. Pour cette application, nous avons 3 paths: dashboard, events, detail/:id, qui permettent respectivement d'accéder au dash, à la liste des événements de l'utilisateur, et aux détails d'un événement en particulier.
- Le dossier components contient les fichiers suivant:
    - app.component.spec.ts et app.component.ts: définissent le composant principal de l'application - la racine.
    - dashboard.component.ts: défini le composant qui correspond au dashboard (la vue d'entrée de l'application).
    - event-detail.component.ts: défini le composant qui correspond à la page d'un événement.
    - event-search.component.ts: défini le composant qui correspond à la barre de recherche d'un événement.

    - events.component.ts: défini le composant qui correspond à la vue des événements de l'utilisateur en liste.
- Le dossier styles et view contiennent respectivement les fichiers de styles et les htmls qui correspondent à chaque composant:
    - app.component.html contient les liens vers la page du dashboard et celle de la liste des événements.
    - dashboard.component.html contient les événements "top" et une barre de recherche (liée à EventSearchComponent).
    - event-search.component contient le html de la barre de recherche.
    - events.component.ts contient la liste des événements de l'utilisateur.
    - event-detail.component contient les détails d'un événement, et permet d'ajouter un événement dans la liste des favoris.
- Le dossier entities contient les fichiers suivant:
    - event.ts: défini ce qu'est un événement.
- Le dossier services contient les fichiers suivant:
    - event-search.service.ts: défini le service de recherche d'un événement.
    - event.service.ts: défini le service qui permet de récupérer les événements à afficher.
    - in-memory-data.service.ts: défini le service qui permet de stocker les événements en mémoire.

Une fois cette application construite, vous pouvez passer à l'étape suivante de ce tutoriel: l'adaptation.

2) Mise en place de l'adaptation

2.2) Mobile Component

Pour ajouter à l'application la barre de favoris qui ne sera visible que sur mobile, nous avons décidé de créer un nouveau composant plutôt que d'utiliser les directives vues ci-dessus, puisqu'il s'agit de créer une vue séparée avec un comportement (des fonctionnalités) spécifique à cette vue (le principe même d'un composant Angular), au lieu de simplement modifier légèrement (afficher ou ne pas afficher) les éléments d'une vue.

Pour le développeur, cela signifie qu'il pourra coder le traitement, la vue et le style de la barre de favoris pour être adapté aux smartphones, sans avoir à se soucier du comportement des autres composants séparés de l'application. Ce composant pourra évoluer indépendamment, l'ajout de nouvelles fonctionnalités est donc plus aisé et n'ira pas surcharger un autre composant.

On crée donc un nouveau fichier "mobile-view.component.ts" dans le dossier components, et un nouveau fichier "mobile-view.component.html" pour la vue associée dans le dossier view (pour le style, on peut réutiliser celui défini dans dashboard.component.html pour l'instant).

Il faut ajouter MobileViewComponent à app.module, pour qu'il soit reconnu par l'application.

MobileViewComponent doit afficher les favoris de l'utilisateur. Il a donc besoin d'importer EventService pour récupérer ces derniers. Il a aussi besoin du Router, afin que l'utilisateur puisse accéder à la page d'un événement s'il clique sur ce-dernier. Finalement, il implémente l'interface OnInit pour pouvoir initialiser sa liste d'événements.

On va donc les ajouter dans le fichier mobile-view.component.ts:

```
import { Component, OnInit } from '@angular/core';
import { Location } from '@angular/common';
import { Router } from '@angular/router';

import { Event } from '../entities/event';

import { EventService } from '../services/event.service';
```

Une fois qu'on a les bons imports, on peut ajouter les fonctions dont on a besoins:

```
<p>Vos événements favoris!</p>

<div class="grid grid-pad">
  <div *ngFor="let event of evts" (click)="gotoDetail(event)" class="col-2-2">
    <div class="module event">
      <h4>{{event.name}}</h4>
    </div>
  </div>
</div>
```

(La fonction getFavoriteEvents() doit être ajouté à EventService.)

On est maintenant capable de récupérer la liste des événements favoris de l'utilisateur. Il faut donc l'afficher. On ajoute le code html dans mobile-view.component.html (lié au MobileViewComponent dans mobile-view.component.ts):

```
<p>Vos événements favoris!</p>

<div class="grid grid-pad">
  <div *ngFor="let event of evts" (click)="gotoDetail(event)" class="col-2-2"
  >
    <div class="module event">
      <h4>{{event.name}}</h4>
    </div>
  </div>
</div>
```

Maintenant, il faut qu'on puisse accéder à la vue du MobileViewComponent. On veut que la barre de favoris soit visible sur mobile, avant toutes choses y-compris les onglets (ou liens de navigation) vers le Dashboard ou la liste des événements de l'utilisateur. On va donc l'intégrer dans app-component avec un simple sélecteur (défini dans mobile-view.component.ts): <mobile-view"></mobile-view>.

Cependant, pour qu'il ne soit visible que sur mobile, il faut ajouter une directive conditionnelle.

2.3) Directives adaptatives

Pour choisir quels éléments afficher, on pourrait utiliser la directive ngIf comme celà: <mobile-view *ngIf="(IsMobile)"></mobile-view>.

Il faudrait à ce moment-là ajouter du code dans app.component pour initialiser la variable "IsMobile" avec le bon booléen:

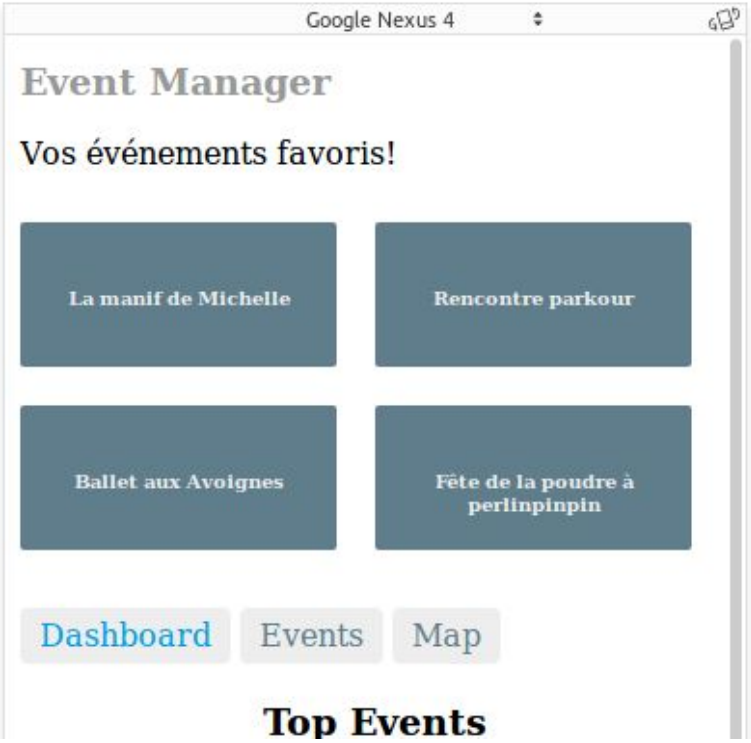
```
title = 'Event Manager';
IsMobile: Boolean = false;

IsPhone() {
  this.IsMobile = window.matchMedia('(max-width: 700px)').matches;
};

ngOnInit(): void {
  this.IsPhone();
}
```

(Note: importer "Location" de '@angular/common'.)

J'ai opté pour utiliser les directives responsives du paquet "ng2-responsive" (lien: https://www.npmjs.com/package/ng2-responsive). Ce paquet fournit une foule de directives responsives (créées par des personnes avec plus de temps et d'expérience que moi pour les développer) qui permettent de s'adapter non-seulement à la taille de l'écran, mais aussi le type d'appareil, le browser, le type de Smarphone (IPhone ou Android), l'orientation de l'écran, etc. Comme notre but est de développer une application responsive et adaptative, il est important de penser aux futures fonctionnalités que nous aimerions implémentés: coder chaque adaptation dont nous pourrions avoir besoin nous-même du début prend du temps et invite à l'erreur.

Ces directives sont simple à les utiliser, on les ajoutes aux divisions html tout comme les directives usuelles d'angular (ngIf). Pour traiter notre scénario, nous n'avons besoin que d'une seule - *isMobile: <mobile-view *isMobile></mobile-view>.

Nous pouvons donc à présent visualiser la barre de favoris sur les écrans mobile:



Elle ne s'affiche pas sur desktop.

2.4) Adapter le css utilisé à la taille d'écran

L'avantage de ces directives responsives, c'est qu'on peut les utiliser sur le même composant afin d'avoir un affichage différent pour des situations différentes et améliorer l'expérience de l'utilisateur.

L'affichage de la liste des événements, par exemple, n'est pour l'instant pas optimal pour une utilisation sur mobile:

Les barres sont trop grandes et ne rentrent pas dans le petit écran. On est obligé de scroller pour voir la fin.

On peut donc créer des nouvelles classes css pour l'affichage des événements dans events-component.css, et modifier le html dans events-component.html pour que l'affichage s'adapte à la résolution d'écran (utilise la classe adaptée), en utilisant ces directives:

```
<ul class="eventsMoyen" *responsive="{ sizes:{min:900,max:1400} }">
  <li *ngFor="let event of evts" (click)="onSelect(event)"
      [class.selected]="event === selectedEvent">
    <span class="badge">{{event.id}}</span>
    <span>{{event.name}}</span>
    <button class="delete"
      (click)="delete(event); $event.stopPropagation()">x</button>
  </li>
</ul>
```

L'utilisateur peut ainsi visualiser sa liste d'événement sur son mobile ou sur son desktop confortablement grâce à cette simple adaptation, et pourtant nous n'avons pas eu besoin de modifier le corps composant (EventsComponent):

Pour le développeur, cette approche est préférable dans ce cas à la création de différents composants pour mobile et desktop, puisque les fonctionnalités sont partagées et seul l'affichage change. On évite de faire du redondant et profite pleinement de la puissance de l'outils WebComponent, dont l'atout majeur et la réutilisation de composants.

Tutoriel sur la partie RWD (Bootstrap) :

1) Comment avez vous réalisé l'exemple ?

1.1) Créer le site web de l'implémentation

Pour commencer dans la réalisation de l'exemple il est nécessaire de créer des fichiers html et css pour l'implémentation RWD. Une fois les fichiers html créés, nous devons intégrer dans nos pages via la balise link de html la librairie permettant l'utilisation de Bootstrap. L'intégration de cette librairie nous permettra d'avoir accès à toutes les fonctionnalités offertes par Bootstrap.
La balise d'intégration de la librairie pourra être trouvée à l'adresse suivante :
http://getbootstrap.com/docs/4.0/getting-started/introduction/

1.2) Visualisation de l'exemple

Pour la visualisation de l'exemple sur ordinateur il suffit d'ouvrir les fichiers dans son navigateur internet. Pour la visualisation sur mobile il peut être utile de mettre son site sur un serveur afin d'y accéder via une url. Une autre solution pourrait être d'installer l'extension chrome web server disponible à l'adresse suivante : https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb?hl=en
Cette extension permet de mettre son site sur un serveur local et d'y accéder ensuite sur son mobile. On peut enfin également utiliser phonegap disponible à l'adresse suivante : https://phonegap.com/ voir la partie Cross Plateform de mon collègue Loïc pour plus d'informations sur l'installation et l'utilisation de phonegap.

1.3) Une première page pour situer le problème

J'ai commencé par créer un première page html pour pouvoir situer la limite de l'utilisation de Bootstrap dans le cas d'un tableau avec beaucoup d'éléments. L'objectif est de montrer qu'une adaptation uniquement en Bootstrap pose des problèmes aussi bien du côté de la qualité de visualisation lors du changement de contexte (passage de l'ordinateur au portable lors du déplacement) que de performance toujours dans le cadre d'un changement de contexte (un tableau trop rempli devrait être long à charger).
En ce qui concerne l'implémentation en Bootstrap j'ai utilisé la grille de Bootstrap pour placer mon calendrier. Ce dernier fait la taille complète de l'écran soit taille de 12 (la grille de Bootstrap est découpée en 12 colonnes qui s'adaptent en fonction de la taille de l'écran).

1.4) Une seconde page pour la solution

Une seconde page html est nécessaire pour montrer la solution qui sera proposée afin de corriger le problème posé par une implémentation uniquement en Bootstrap. Pour cela j'utilise du Javascript pour déporter l'affichage d'événements du calendrier à une partie de l'écran qui n'est affichée que lorsqu'on le souhaite en cliquant sur le jour du calendrier sur lequel on souhaite avoir plus d'informations.
Pour l'implémentation, j'utilise toujours la grille de Bootstrap mais avec cette fois ci une colonne de taille 3 qui sera dédiée à l'affichage des informations des événements et une colonne de taille 9 pour l'affichage du calendrier. Lorsqu'on change de contexte d'affichage en passant par exemple d'un ordinateur à un smartphone la colonne d'affichage des événements va passer au dessus du calendrier grâce à l'organisation de notre grille. Sur un écran suffisamment grand il est possible d'avoir tout l'affichage d'un seul coup d'oeil.

2) Avec quel outil de développement, de tests ?

2.1) Outil de développement

Pour le développement il m'a suffit d'utiliser un éditeur de texte simple comme Notepad++ qui est téléchargeable en cliquant sur ce lien :
https://notepad-plus-plus.org/

Ce dernier est suffisant pour éditer les fichiers html et css de mon exemple.

2.2) Outil de test

Pour les tests j'ai utilisé un ordinateur portable de marque Acer de diagonale d'écran de 15 pouces mais aussi un smartphone de type Motorola Moto G3 possédant un écran de 4,7 pouces afin de tester l'adaptation lors du changement de contexte (déplacement par exemple). Dans les deux cas le navigateur utilisé est Chrome.

3) Comment déploie-t-on et exécute-t-on l'exemple ?

3.1) Déploiement et exécution sur pc

Pour déployer et exécuter l'exemple sur ordinateur il suffit de se rendre dans le dossier du projet et de cliquer sur index.html ou solution.html qui sont les deux pages html du projet. Cela suffit pour pouvoir visionner l'exemple et tester les capacités d'adaptations dans n'importe quel navigateur.

3.2) Déploiement et exécution sur smartphone

Pour déployer et tester l'exemple sur smartphone il est possible d'utiliser l'url suivante :
https://www.e-meta.fr/up/M2/index.html
On accède alors à la première page html qui définit le problème. Il est possible de cliquer sur le lien "Voir la solution" pour accéder au second fichier html représentant la solution.
Il est également possible d'installer Phonegap sur son ordinateur et sur son téléphone. Il faut être connecté au même réseau et créer un nouveau projet Phonegap où l'on mettra les fichiers html. Une fois le serveur lancé il suffit d'utiliser l'url fournie pour accéder au site.

4) Comment teste-t-on les capacités d'adaptations ?

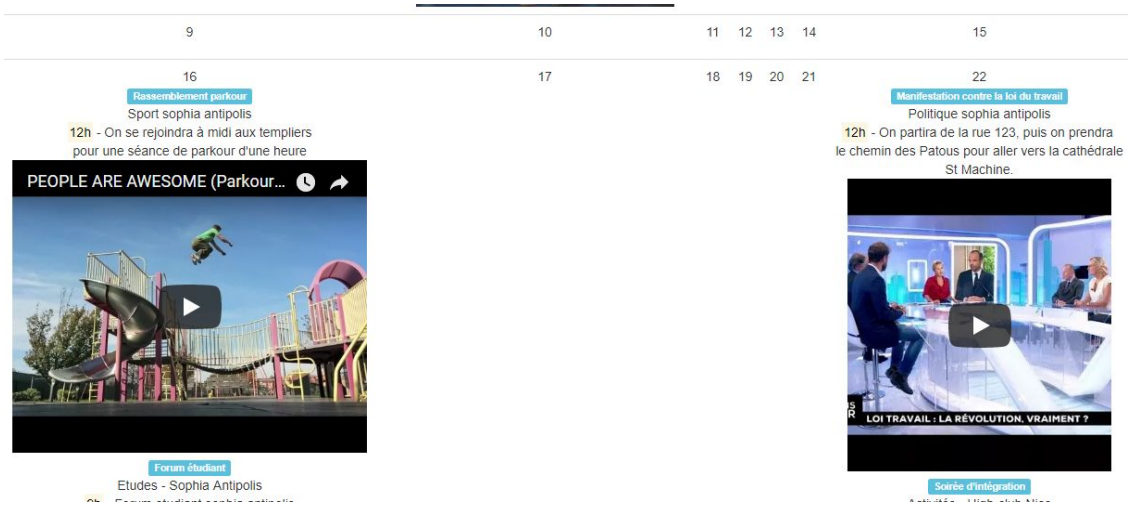
L'objectif est de tester si lors du changement de contexte d'utilisation (par exemple en mobilité), les capacité d'adaptation de la technologie choisie (ici Bootstrap), nous permettent de garder une expérience utilisateur comparable que ce soit en terme de visibilité du contenu ou en terme de rapidité d'affichage de l'information.

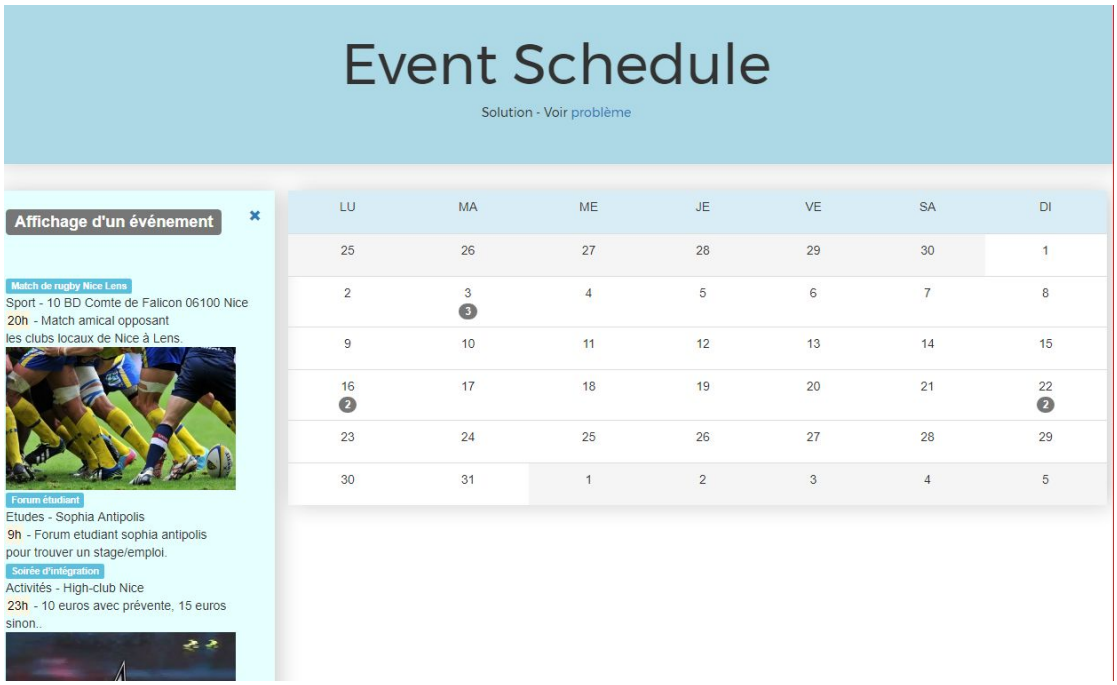
4.1) Test des capacités d'adaptation sur la visibilité du contenu

4.1.a) Sur ordinateur

Sur la première image (représentant le problème), on peut voir que même si le calendrier apparaît en entier, la visibilité de l'information est mauvaise et il est difficile de trouver ce que l'on cherche.

La seconde image représente la solution. Ici le calendrier se trouve à droite de l'écran et un espace est réservé à gauche pour l'affichage des informations sur les événements. La visibilité est grandement améliorée et il est beaucoup plus simple de trouver l'information recherchée.



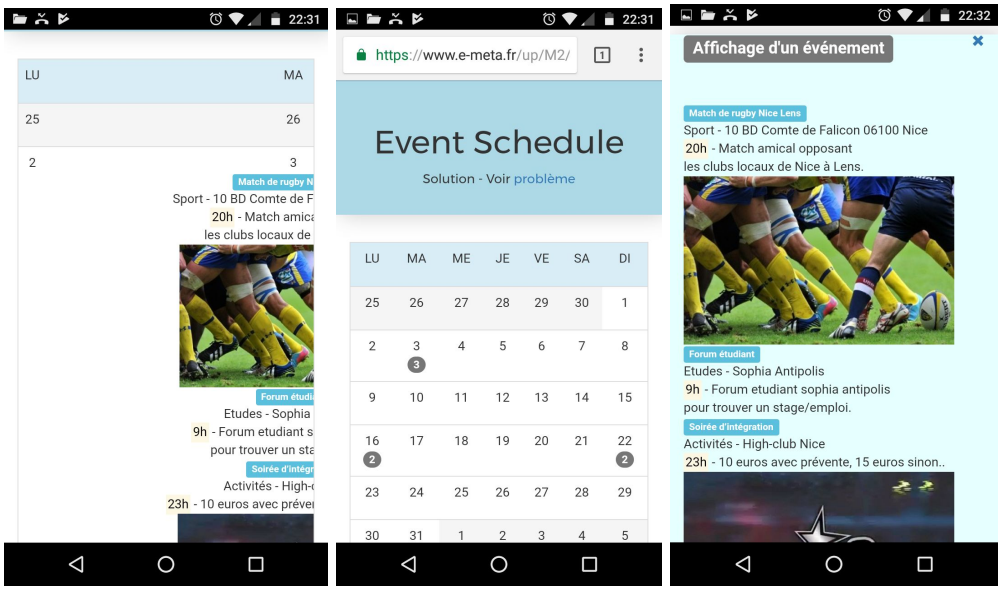


4.1.b) Sur smartphone

La première image correspond à l'affichage de la page index.html représentant l'implémentation uniquement en Bootstrap. Nous voyons rapidement un problème apparaître. En effet, il est très difficile de naviguer dans le calendrier qui est beaucoup trop grand pour un écran de smartphone.

Les deux images suivantes montrent l'implémentation de la page solution.html qui montre l'implémentation en Boostrap plus Javascript pour afficher plus d'informations sur les événements. Ici la visibilité du contenu est grandement améliorée et le calendrier apparaît en

entier dans l'écran de l'utilisateur qui n'a plus qu'à cliquer sur les bulles situées en dessous de chaque jour où se déroulent des événements pour avoir plus d'informations à leurs sujets.



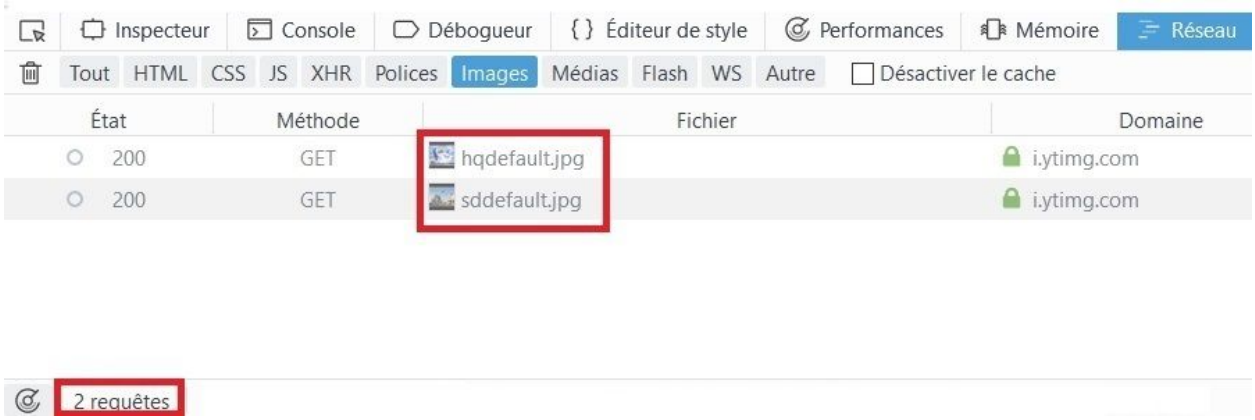
Au final on observe ici que l'implémentation en Boostrap uniquement n'est pas suffisante pour proposer une expérience utilisateur correcte en terme de visibilité. Il est nécessaire d'utiliser d'autres outils comme je l'ai fais avec du Javascript afin de déporter l'affichage des informations sur les événements.

4.2) Test des capacité d'adaptation sur la rapidité d'affichage de l'information

Pour tester la rapidité d'affichage des mes deux fichiers html et voir laquelle des deux pages est la plus rapide à charger, j'utilise simplement l'onglet "Network" du menu de Google Chrome/Firefox qui peut être ouvert en appuyant sur la touche F12.

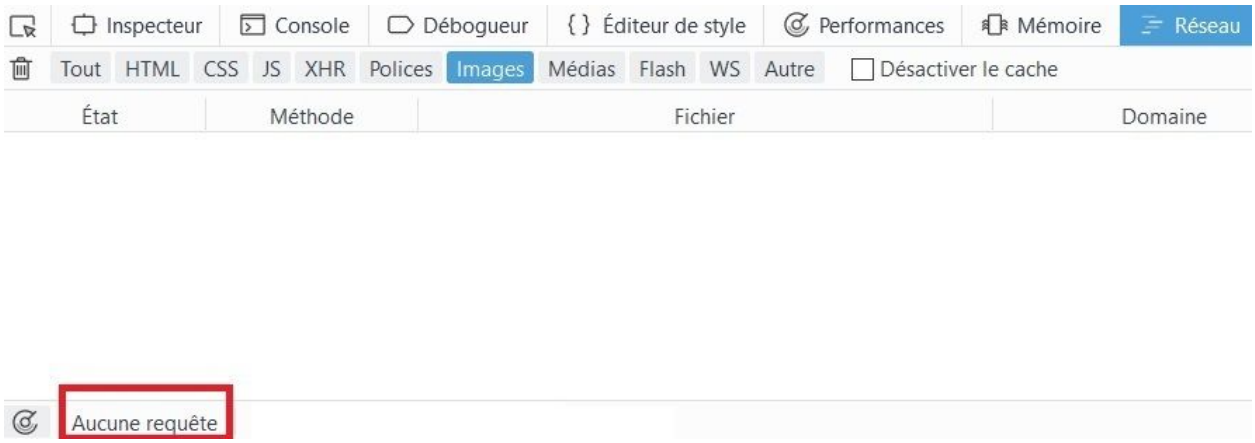
Temps d'affichage de la page index.html symbolisant le problème ou l'adaptation en Bootstrap uniquement :

Nombre de requêtes effectuées pour les images lors du chargement de la page :



Temps d'affichage de la page solution.html symbolisant la solution ou l'adaptation en Bootstrap plus Javascript pour avoir plus d'informations sur les événements :

Nombre de requêtes effectuées pour les images lors du chargement de la page :



On peut voir que la page solution.html est légèrement plus rapide à s'afficher alors que le contenu des deux pages est le même. Cela est dû au fait que certains navigateurs comme Firefox ne chargent pas les images de fond des blocs lorsqu'ils sont en display:none pour accélerer le chargement de la page. On peut voir que dans la page du problème le navigateur fait deux requêtes pour charger les deux images de fond des vidéos alors que dans la page de la solution les requêtes ne sont pas effectuées.

Attention cependant, cela ne fonctionne pas avec tous les navigateurs :

Par exemple Chrome ou Safari qui utilisent le moteur de rendu Webkit chargent dans tous les cas toutes les images du dom. (pour plus d'informations voir https://stackoverflow.com/questions/12158540/does-displaynone-prevent-an-image-from-loading). Si le temps gagné semble ridicule (moins d'une seconde) pour un ordinateur fixe, cela peut

sembler crucial pour un smartphone dont la connexion est souvent instable et le processeur moins puissant.

Tutoriel sur la partie au choix (PWA) :

Pour cette technology il suffit d'avoir n'importe quelle éditeur de text pour créer la page html, pour ma part j'utilise webStorm de jetBrains. Une fois la page crée on peut soit la consulter sur le navigateur de la machine, soit le maître sur le réseaux pour le consulter depuis d'autre machine.
Pour les test j'utilise les même outil que la partie RWD.

Conclusion

Nous avons pu voir à travers les implémentations réalisées à l'aide des différentes technologies choisies que chacune présente des avantages et des possibilités pour répondre aux besoins d'adaptations en fonction du contexte dans lequel ou lesquels peuvent se trouver notre application ou même la plupart des applications.
Il existe en revanche aussi certains inconvénients et limites qu'imposent une technologie particulière; c'est pourquoi il peut être utile de s'intéresser à l'ensemble des technologies disponibles ou tout du moins stable pouvant répondre aux besoins, et qu'avant d'implémenter une solution avec une technologie pour laquelle le développement est plus rapide ou facile il serait plus intéressant de choisir à la place la technologie la mieux adaptée à la place.