Jean-Yves Delmotte
Fabien Pinel

SI5-IHM

Rapport

Adaptation des interfaces à l'environnement

Cordova / Polymer

Comment avez vous réalisé l'exemple ?

Voici un tutoriel reprenant pas à pas les étapes nécessaire pour réaliser cet exemple.

Installation des outils

Pour commencer ce tutoriel, nous allons installer le célèbre gestionnaire de paquet NodeJS npm : https://nodejs.org/en/.

Puis, grâce à cette fameuse commande (npm), nous allons installer les dépendances nécessaires pour réaliser ce projet (à savoir : Cordova et Bower) :

```
npm install -g cordova bower
```

Petite explication : Cordova va nous permettre de déployer notre application web sous la forme d'une application Android et Bower va nous permettre de gérer toutes les dépendances web de notre application c'est à dire télécharger les librairies nécessaires.

Il va ensuite falloir installer Android SDK si ce n'est pas déjà fait sur votre machine et préciser dans votre variable globale PATH le chemin vers ANDROID_HOME. Nous supposons ici que Java est installé sur votre machine.

Création du projet

On va maintenant initialiser le projet grâce à la commande :

```
cordova create app
```

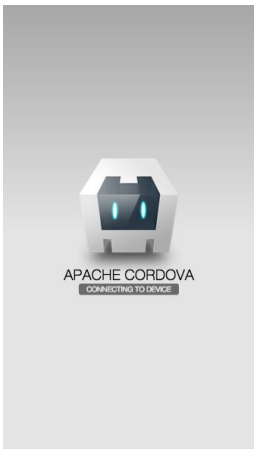
Cordova va alors créer toute l'arborescence de dossier nécessaire au développement et au déploiement de votre application. Nous travaillerons uniquement dans le dossier www/ car c'est là que nous développerons notre application.

Afin de s'assurer que tout fonctionne correctement, nous allons dès le début déployer l'application sur un device physique ou un émulateur, pour cela :

```
cd app/
cordova platform add android
cordova run android
```

Si un appareil Android reconnu par votre ordinateur est branché et qu'il accepte les options pour les développeurs, l'application sera déployée sur cet appareil, sinon un émulateur sera démarré (il faudra précédemment en ajouter un).

Si tout se passe bien vous devriez obtenir un résultat similaire à celui ci sur votre téléphone ou émulateur. Si ce n'est pas le le cas, copiez coller votre erreur sur internet. Un des points forts de Cordova est sa communauté, et vous trouverez sûrement votre réponse sur des forums. La plupart du temps, les bugs sont liés à des variables manquantes dans le PATH, mais ce n'est pas le sujet de ce tutoriel.



Restructuration des fichiers

Nous allons maintenant créer l'architecture de dossier pour notre projet, rendez vous dans le dossier www/ de notre application. Nous allons commencer par initialiser le bower (gestionnaire de dépendance pour le web).

```
cd WhatSnap/www
bower init
```

Ensuite, nous allons tout de suite effacer les fichiers inutiles :

```
rm -rf css/*
rm -rf js/*
mkdir components
```

Et adapter le fichier index.html comme suis :

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="format-detection" content="telephone=no" />
    <meta name="msapplication-tap-highlight" content="no" />
    <meta name="viewport" content="width=device-width, minimum-scale=1.0, initial-scale=1.0, user-scalable=yes">

    <title>WhatSnap</title>

    <style>body { background-image: url(http://fonds.toutimages.com/fo_beige/beige159.jpg); }</style>
</head>
<body style="margin:0px;">

Hey this is a test

<!— 404 in web browser —>
<script type="text/javascript" src="cordova.js"></script>

</body>
</html>
```

Notre Premier Composant Polymer

C'est parti pour construire notre premier composant avec Polymer ! Nous allons tout d'abord installer toutes les librairies Polymer nécessaires au projet à l'aide de bower :

```
bower install PolymerElements/paper-toolbar PolymerElements/paper-icon-button PolymerElements/paper-item PolymerElements/paper-badge PolymerElements/paper-fab PolymerElements/paper-menu PolymerElements/paper-ripple --save
```

Créez votre premier composant dans le dossier components/ et appelez le *my-home.html*. Les données que nous allons utilisé dans cet exemple sont disponibles dans le git, il s'agit du fichier *app/www/data.js*. Importez ce fichier dans votre html de sorte à pouvoir accéder à la variables 'talks' en global.

Le fichier *my-home.html* va se découper en **3 parties distinctes** :

1. **<u>Les imports</u> : ils servent à récupérer les composants de la librairie native de Polymer ou les composants que nous avons nous même créés :**

```html
<link rel="import" href="../bower_components/polymer/polymer.html">
<link rel="import" href="../bower_components/iron-icons/iron-icons.html"/>
<link rel="import" href="../bower_components/paper-icon-button/paper-icon-button.html"/>
<link rel="import" href="../bower_components/paper-toolbar/paper-toolbar.html"/>
<link rel="import" href="../bower_components/paper-item/paper-item-body.html"/>
<link rel="import" href="../bower_components/paper-item/paper-item.html"/>
<link rel="import" href="../bower_components/paper-badge/paper-badge.html"/>
<link rel="import" href="../bower_components/paper-ripple/paper-ripple.html"/>
```

2. **<u>Le script du composant</u> : c'est ici que va se trouver toute la logique de la vue, à savoir : les données, les listeners et les actions :**

```
<script>
    Polymer({
        is: 'my-home',

        properties: {
            conversations: {
                type: Array,
                value: talks
            }
        },

        downAction: function(e) { this.$.ripple.downAction({x: e.x, y: e.y}); },
        upAction: function() { this.$.ripple.upAction(); }

    });
</script>
```

3. La description du composant en html et en css dans laquelle on peut discerner un certain nombre d'inconnus qui font référence au script du composant :

```
<dom-module id="my-home">

    <style...>

    <template>

        <!-- menu bar -->
        <paper-toolbar>
            <paper-icon-button icon="menu"></paper-icon-button>
            <div class="title">WhatSnap</div>
        </paper-toolbar>

        <!-- Display all conversations -->
        <div class="talks-container">
            <template is="dom-repeat" items="{{ conversations }}">

                <paper-item class="grid-item">
                    <div class="avatar green" item-icon>
                        <img src="{{item.image}}"
                             alt=""
                             style="..."/>
                    </div>
                    <paper-item-body two-line class="layout vertical">
                        <div>{{ item.name }}</div>
                        <div secondary><span>{{item.place}}</span>, <span>{{item.date}}</span></div>
                    </paper-item-body>
                    <div><paper-badge label="{{ item.notif }}"></paper-badge></div>
                    <paper-ripple fit id="ripple" style="..."></paper-ripple>
                </paper-item>

            </template>
        </div>

    </template>

</dom-module>
```

Maintenant que nous avons notre composant *my-home.html* il faut l'importer dans l'*index.html* de notre application. Pour cela rien de plus simple :

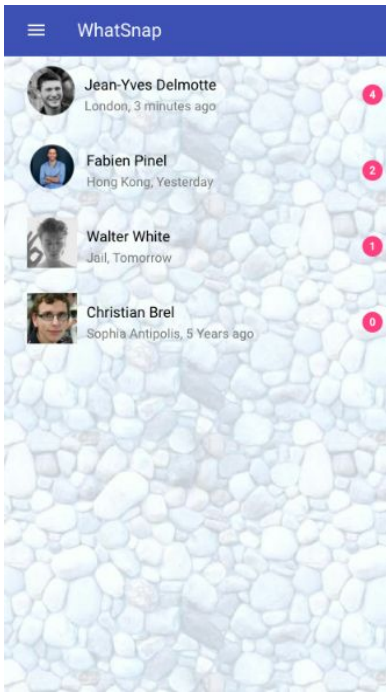
```
    <script src="data.js"></script> <!-- Les fameuses données dont nous parlions tout à l'heure ! -->
    <script src="./bower_components/webcomponentsjs/webcomponents.min.js"></script>
    <link rel="import" href="./components/my-home.html"/>
```

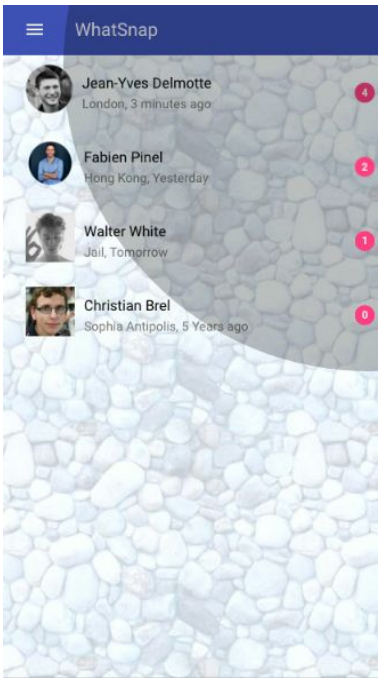
Ensuite, il faut l'instancier directement après le <body>, en remplacement le 'Hey this is a test' par : 19 `<my-home></my-home>`

Il ne faut pas oublier d'ajouter le plugin suivant avant exécution, sans quoi le navigateur refusera de télécharger tout ce qui se trouve hors de l'application (image, fichier, ...) :

```
cordova plugin add whitelist
```

Voici le résultat que vous obtiendrez après un ‘cordova run android’ :



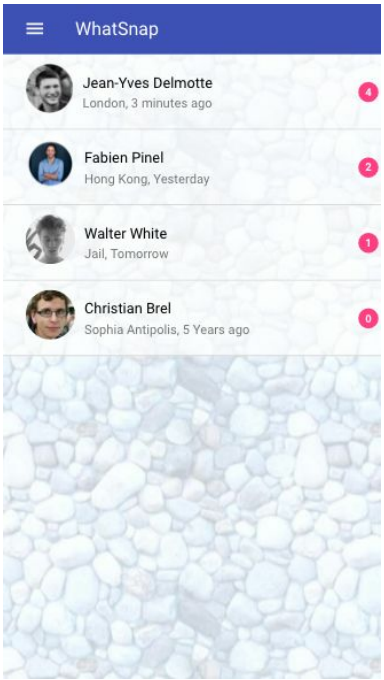


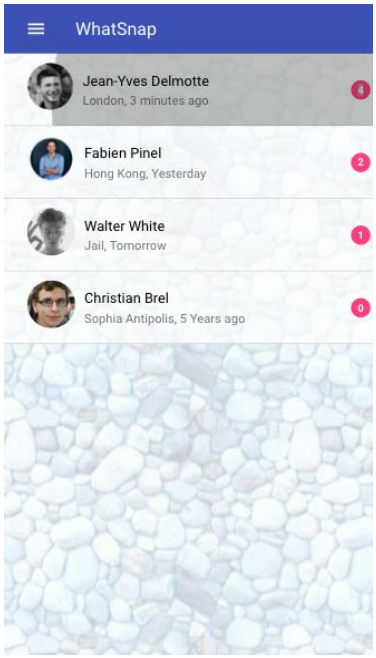
Et voilà ! Le premier bug ! Quand on clique sur un élément le ‘paper-ripple’ dépasse de son cadre ! De plus la vue n’est absolument pas responsive. En effet, lorsqu’on augmente la largeur de la fenêtre, la liste ne fait que s’étirer et le badge sur la droite s’éloigne de plus en plus…

Allez, un peu de CSS et nous allons corriger tout ça :

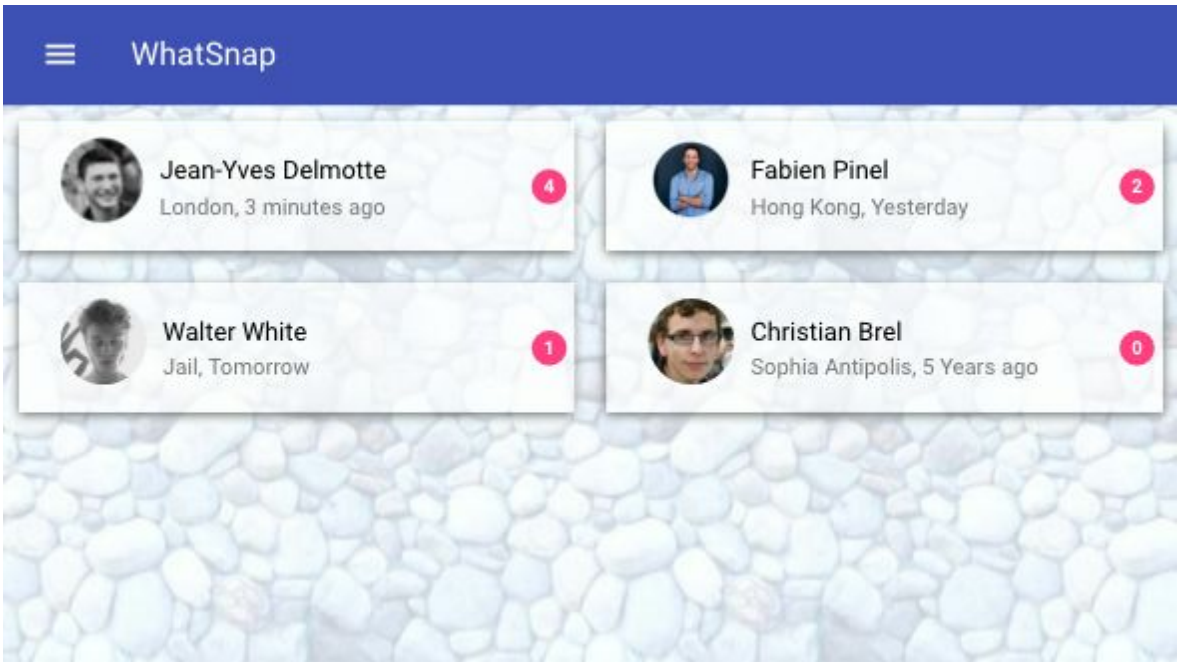
```
<style>
    .avatar img {
        -webkit-border-radius: 10000px;
        -moz-border-radius: 10000px;
        border-radius: 10000px;
    }

    .grid-item {
        float:left;
        position:relative;
        outline:0px!important;
        background-color:rgba(255,255,255,0.7)
    }

    @media (max-width: 600px) {
        .grid-item {
            width: calc(100% - 30px);
            border-bottom:1px lightgrey solid;
        }
    }
    @media (min-width: 600px) and (max-width: 800px) { .grid-item { width: calc(50% - 52px); } }
    @media (min-width: 800px) and (max-width: 1000px) { .grid-item { width: calc(33% - 52px); }}
    @media (min-width:1000px) { .grid-item { width: calc(25% - 52px); }}
    @media (min-width: 600px) {
        .grid-item {
            margin:10px;
            -moz-box-shadow: 0px 3px 10px 0px #656565;
            -webkit-box-shadow: 0px 3px 10px 0px #656565;
            -o-box-shadow: 0px 3px 10px 0px #656565;
            box-shadow: 0px 3px 10px 0px #656565;
            filter:progid:DXImageTransform.Microsoft.Shadow(color=#656565, Direction=180, Strength=10);
        }
    }
    .talks-container { -webkit-perspective: 500px; }
    .talks-container paper-item { -webkit-animation-fill-mode: forwards; }
</style>
```

Maintenant vous devriez obtenir quelque chose de similaire à ceci :





et en plus c'est responsive !



Il est à noter ici que Polymer ne dispose pas de système de grille pour adapter facilement l'affichage en fonction de la taille de la fenêtre. Polymer est un framework qui propose de nombreux composants graphiques et ce système de grille est assez classique donc il est vraiment dommage que la librairie ne l'inclue pas à l'origine. Cela nous oblige à l'implémenter nous même ou à en importer une existante qui alourdira l'application. C'est un vrai manque dans Polymer.

Optimisation de la vue avec l'accéléromètre

Avez vous déjà utilisé un iPhone ? Cette question n'est pas anodine car si vous avez déjà pratiqué iOS vous avez sans doute remarqué l'animation qui se joue lorsqu'on penche le téléphone. Les applications du menu principal effectuent une légère rotation pour rester parallèle à vos yeux. Cela donne un effet 3D plutôt réussi. Nous allons essayer de reproduire cet effet sur la liste des conversations grâce à l'accéléromètre. Pour que Cordova nous donne accès à l'accéléromètre il faut installer un plugin.

Installation du plugin :

```
cordova pluggin add cordova-plugin-device-motion
```

On ajoute pour cela quelques lignes de code dans le constructeur Polymer :

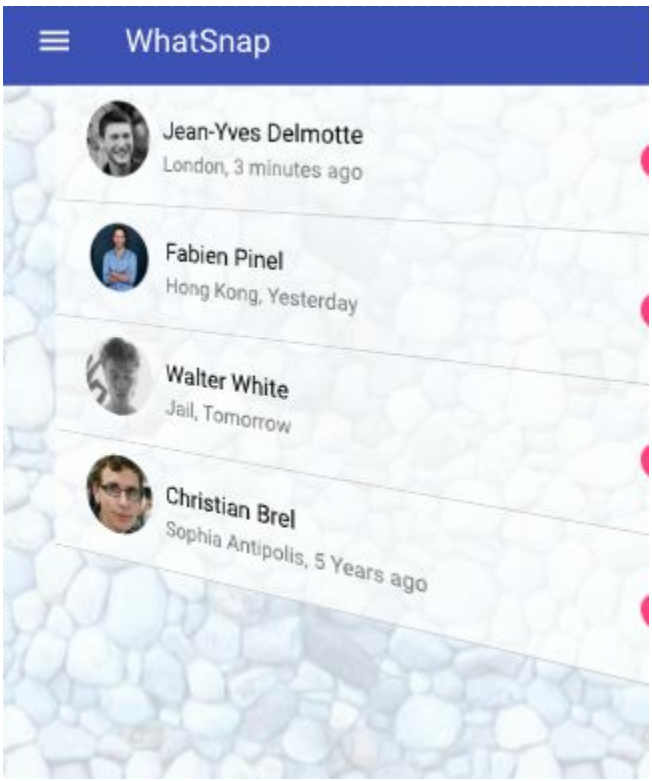
```
93      attached: function () {
94          this._onAccelerometer();
95      },
96
97      _onAccelerometer: function (){
98          var self = this;
99          if (navigator.accelerometer) {
100             navigator.accelerometer.getCurrentAcceleration(function success(acceleration) {
101                 self._rotateY(acceleration.x);
102                 self._onAccelerometer();
103             }, function error() {
104                 console.log('error');
105             });
106         }
107     },
108
109     _rotateY: function (deg) {
110         var paperItems = document.querySelectorAll('.talks-container paper-item');
111         for (var i in paperItems) {
112             paperItems[i].style['-webkit-transform'] = 'rotate3d(0, 1, 0, ' + (deg*4) + 'deg)';
113         }
114     },
115
116     detached: function () {
117         if (navigator.accelerometer) navigator.accelerometer.clearWatch();
118     },
```

Explication :

Lorsque le composant est créé la fonction *'attached'* est appelée. On commence donc à faire appel à l'accéléromètre. A l'inverse c'est dans la fonction *'detached'* (qui est appelé avant la destruction du composant) qu'on arrête l'utilisation de l'accéléromètre.

Ensuite, lorsque l'accéléromètre est lancé, on lance une récursivité infinie qui à chaque fois remet à jour le valeur de la rotation que doivent effectuer les éléments de notre liste de conversations.

Déployez sur votre appareil et vous devriez obtenir quelque chose d'horriblement non réactif qui prend cette apparence :



Cette fois, nous constatons que Cordova semble démontrer ses limites lors de transformations 3D couplés à l'écoute de l'accéléromètre en permanence. Les ralentissements se font sentir et la fluidité est un lointin souvenir.

Lier ses composants avec le router

Avoir un composant c'est bien, en avoir deux c'est très bien, mais les lier entre eux c'est mieux ! Dans cette partie nous allons mettre en place un *router* !

Nous commençons cette partie avec une critique : Polymer n'a pas de composant *router* natif. Il faut donc en installer un :

```
bower install app-router --save
```

Dans l'*index.html* nous allons remplacer notre déclaration de *my-home* par :

```
19  <app-router>
20      <app-route path="/" import="./components/my-home.html"></app-route>
21      <app-route path="/home" redirect="/"></app-route>
22      <app-route path="*" redirect="/"></app-route>
23  </app-router>
```

Il faut également remplacer notre import de *my-home.html* par :

```
13        <link rel="import" href="./bower_components/app-router/app-router.html"/>
```

Et nous retrouvons le même résultat que précédemment !

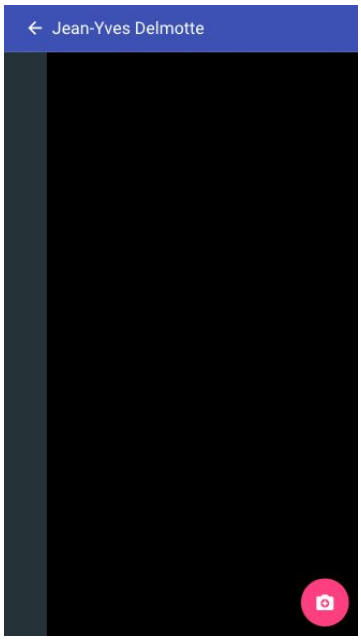
Explication :

Nous avons mis en place un système de routing qui pour le moment redirige tout le page vers l'url '/' qui correspond à notre composant "my-home". Lorsque nous ajouterons un nouveau composant nous devrons l'ajouter à notre liste de *app-route* et nous verrons que nous pouvons aussi passer des paramètres dans les URLs.

Notre deuxième composant : une gallerie

Maintenant que nous avons notre premier composant qui affiche les conversations, nous allons faire notre deuxième composant qui va afficher le contenu d'une conversation.

Voilà le rendu global de ce que nous désirons obtenir :



En haut, la barre de navigation, sur le côté nous mettrons les miniatures des images, dans le centre l'image sélectionné et en bas à droite un bouton pour ajouter une nouvelle photo.

Nous allons créer un fichier *my-conversation.html* dans le dossier *components*, puis le référencer dans notre router. Pour cela il suffit d'ajouter la ligne :

```
21      <app-route path="/conversation/:conversationId" import="./components/my-conversation.html"></app-route>
```

On remarque ici que cette route possède un paramètre 'conversationId'. Il sera disponible à tout moment dans le composant '*my-conversation*'.

La première chose à faire est de raccorder les deux vues, c'est à dire que lorsqu'on clique sur une conversation présente dans *'my-home'*, on soit redirigé vers le composant '*my-conversation*' correctement paramétré.

On ajoute cette fonction dans le constructeur polymer de *'my-home'* :

```
120         goConversation: function (item) {
121             document.querySelector('app-router').go('/conversation/' + item.model.item.id);
122         },
```

et on l'appelle lorsque notre clique / doigt se relève :

```
61                  <paper-item on-up="goConversation" class="grid-item">
```

Ceci étant fait nous pouvons construire notre composant *'my-conversation'*
Pour cela, voici les imports que nous devrons réaliser :

```
1   <link rel="import" href="../bower_components/polymer/polymer.html">
2   <link rel="import" href="../bower_components/iron-icons/iron-icons.html"/>
3   <link rel="import" href="../bower_components/paper-icon-button/paper-icon-button.html"/>
4   <link rel="import" href="../bower_components/paper-toolbar/paper-toolbar.html"/>
5   <link rel="import" href="../bower_components/paper-fab/paper-fab.html"/>
```

Le script :

```
<script>
    Polymer({
        is: 'my-conversation',

        properties: {
            messages: {
                type: Array,
                value: []
            },
            userName: {
                type:String,
                value: 'Contact Name'
            }
        },

        attached: function () {
            for (var i in talks) {
                if (talks[i].id == this.conversationId) {
                    this.set('userName', talks[i].name);
                    this.set('messages', talks[i].messages);
                    break;
                }
            }
        },

        goBack: function () { document.querySelector('app-router').go('/home'); }

    });
</script>
```

et le template :

```
<dom-module id="my-conversation" attributes="conversationId">
    <style...>

    <template>

        <!-- menu bar -->
        <paper-toolbar>
            <paper-icon-button icon="arrow-back" on-up="goBack"></paper-icon-button>
            <div class="title">{{ userName }}</div>
        </paper-toolbar>

        <!-- Side bar -->
        <div class="content side-thumbnail" style="padding-bottom: 67px">
            <template is="dom-repeat" items="{{ messages }}" >
                <div class="imageContainer" >
                    <div id="{{ index }}">
                        <!-- Houston, we have a problem ... -->
                    </div>
                </div>
            </template>
        </div>

        <!-- image container -->
        <div class="image-container" id="messages"></div>

        <!-- add photo from camera -->
        <div class="add-photo">
            <input type="file" id="upload_input" accept="image/*" style="..."/>
            <paper-fab id="upload_btn" icon="camera-enhance" on-click="_sendPicture"></paper-fab>
        </div>

    </template>

</dom-module>
```

Ici nous avons rencontré un problème. Nous voulions mettre en image de fond l'image contenu dans chaque message, mais nous ne pouvons le faire avec Polymer. En effet, il n'est pas prévu de pouvoir rajouter un attribut "style" sur les balises HTML en lui injectant des données. Nous avons donc du contourner le problème (Polymer). La technique que nous proposons n'est pas très sûr mais fonctionne: nous allons mettre en classe l'url de l'image à afficher pour ensuite avec le Javascript la récupérer et changer le style.

Pour information, la balise style est passée sous silence mais voici son contenu :

```
<style>
    @media (max-width: 639px) {
        .side-thumbnail{ top: 56px !important; }
        .image-container { padding-top: 56px!important; }
    }
    paper-toolbar {  z-index:100;  position:fixed;  top:0px;  width:100%;  }
    .add-photo {  position:fixed;  bottom:0px;  display:block;  width:100%;  text-align:right;  }
    .add-photo paper-fab {  top:-12px;  right:12px;  float:right;  }
    .side-thumbnail{
        position: fixed;
        left: 0;
        top: 64px;
        z-index: 1000;
        height: 100%;
        background-color: #263238;
        width : 12%;
        overflow: auto;
    }
    .img_container{
        height : 100%;
        background-position: center center !important;
        background-repeat: no-repeat;
        background-size: cover !important;
    }
    .imageContainer{  height: 60px;  margin : 3px 3px 0px 3px;
    }
    @media (max-width: 639px) {  .image-container { top:56px!important; }  }
    .image-container {
        top: 64px;
        bottom:0px;
        left: 12%;
        width:88%;
        position: absolute;
        display:block;
        background-color: black;
    }
    .side-thumbnail::-webkit-scrollbar { display: none; }
</style>
```

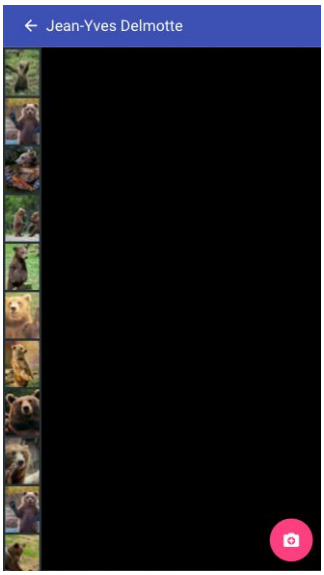
Pour cela il suffit de rajouter le code suivant à notre constructeur Polymer :

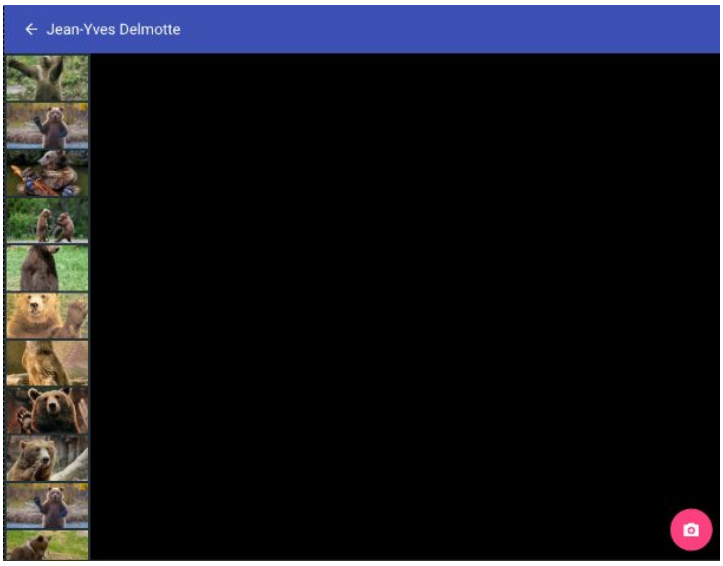
```
172         ready: function () {
173             setTimeout(function () {
174                 var imgs = document.querySelectorAll('.img-unique');
175                 for (var i = 0; i < imgs.length; i++) {
176                     imgs[i].style.background = 'url(' + (''+imgs[i].className).split(' ')[2] + ')';
177                 }
178             });
179
180         },
181
182         buildClass: function (image_url) {
183             return 'img_container img-unique ' + image_url ;
184         },
```

Et à notre HTML :

```
127                     <div id="{{ index }}" class$="{{buildClass(item.image_url)}}">
128                         <!-- Houston, we have a problem ... -->
129                     </div>
```

Nous avons maintenant un rendu très acceptable, mais lorsqu'on change de taille d'appareil on se rend vite compte que l'adaptation n'est pas aussi bien réussi que ce qu'on pensait :





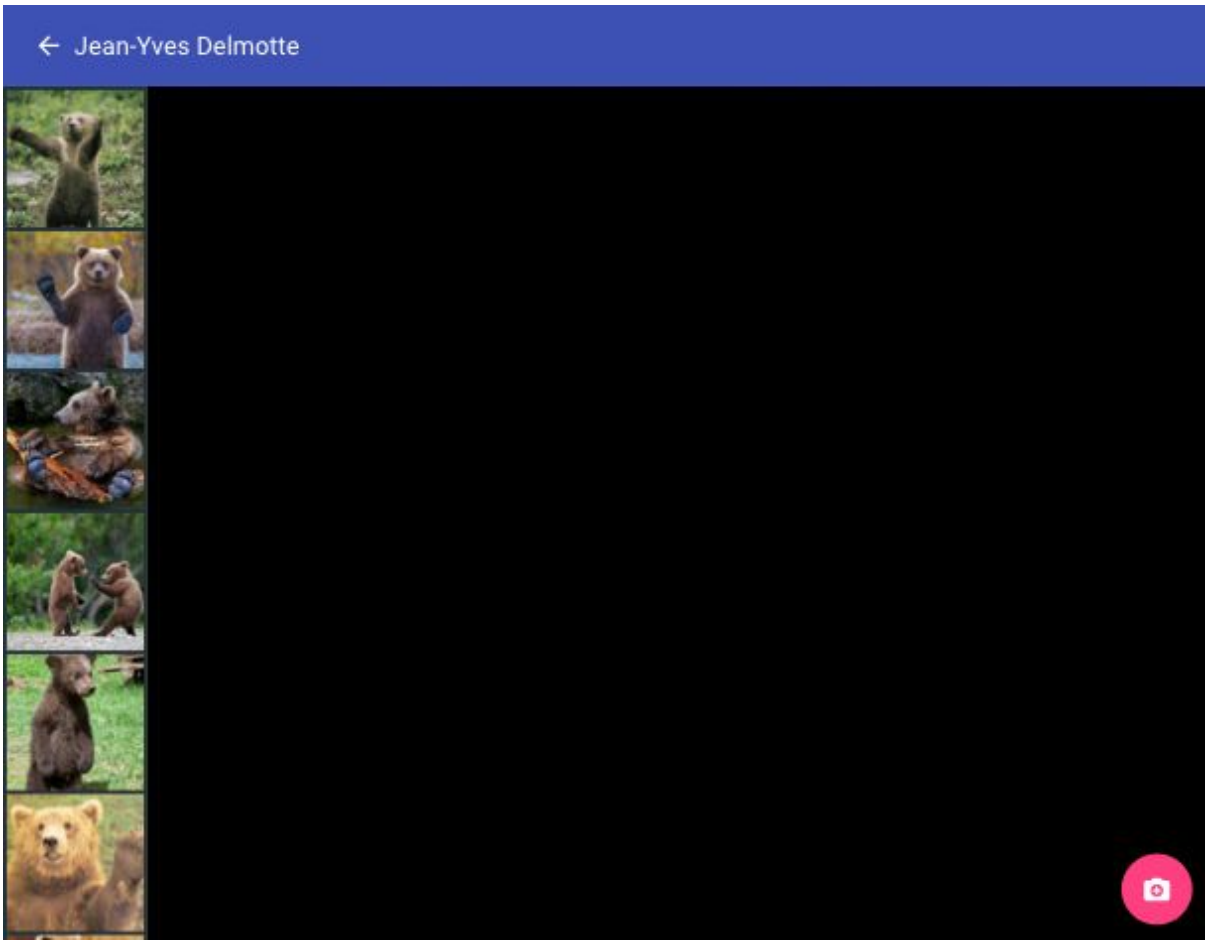
C’est pour cela que nous avons décidé de mettre les aperçus en forme de carré.

```
174         _updateWidthSlider: function () {
175             var newHeight =window.getComputedStyle(document.querySelector(".imageContainer"), null).getPropertyValue("width");
176             var imgs = document.querySelectorAll(".imageContainer");
177             for (var i in imgs) {
178                 try {
179                     imgs[i].style.height = newHeight;
180                 } catch (e) {}
181             }
182         },
```

Ensuite, il faut appeler cette fonction qui adapte les aperçus dans le *‘attached’* du composant, le tout dans un timeout qui permet d’attendre que le DOM ait fini de charger:

```
171                 setTimeout(this._updateWidthSlider, 100);
```

Remarque : On peut se permettre cette pratique car nous utilisons Cordova qui considère les fichiers en local. Il serait vraiment déconseillé de faire cela sur un serveur web dont le temps de réponse peut varier du tout au tout.
Le rendu est nettement meilleur sur l’appareil et on ne voit plus de déformations :



Maintenant nous allons attaquer la partie centrale : nous allons afficher l'image sélectionné (la première de la liste par défaut), la sélection sur l'événement "clique" sur la barre de gauche et le zoom/dézoom sur l'évènement 'double tap'.

La première chose est d'afficher la première image (par défault). Pour cela nous allons télécharger iviewer qui permet de mettre l'image à 100% de sa largeur ou de sa hauteur en fonction de la taille de l'écran et qui permet également de zoomer/dézoomer. Ce module dépend de jquery et de jquery-ui, c'est pourquoi nous les incluons également.

```
bower install jquery jquery-ui iviewer --save
```

On fait les inclusions nécessaires dans notre *index.html*, ce qui nous donne :

```
11      <script src="data.js"></script> <!-- Les fameuses données dont nous parlions tout à l'heure ! -->
12      <script src="./bower_components/webcomponentsjs/webcomponents.min.js"></script>
13      <script src="./bower_components/jquery/dist/jquery.min.js"></script>
14      <script src="./bower_components/jquery-ui/jquery-ui.min.js"></script>
15      <script src="./bower_components/iviewer/jquery.iviewer.min.js"></script>
16      <link rel="import" href="./bower_components/app-router/app-router.html"/>
17      <link rel="stylesheet" href="./bower_components/iviewer/jquery.iviewer.css"/>
```

il nous reste plus qu'à transformer notre *setTimeout* dans la fonction '*attached'* de notre composant *'my-conversation'* en :

```
171             var self = this;
172             setTimeout(function () {
173                 if(self.messages.length > 0) {
174                     $('#messages').iviewer({src: self.messages[0].image_url, ui_disabled: true});
175                 }
176                 self._updateWidthSlider();
177             }, 100);
```

Maintenant vous devriez obtenir quelque chose qui ressemble à ça :

Le statique c'est bien, mais le dynamique c'est mieux, on aimerait pouvoir changer la photo en cliquant sur la miniature. Allons y !

Pour la première fois, nous allons écrire du code en dehors du constructeur Polymer car nous n'avons pas trouvé de solution pour transmettre correctement les informations à une fonction qui se trouverait dans ce constructeur. Cela nous donne donc :

```
150 <script>
151     Polymer({"is": 'my-conversation'...});
210
211     function changeMainPicture(urli){
212         $('#messages').iviewer('loadImage',  urli.style.background.split('(')[1].split(')')[0]);
213         $('#messages').iviewer('fit');
214     }
215
216 </script>
```

Il suffit alors de rajouter ceci dans le template html pour que tout fonctionne :

```
126                 <div class="imageContainer" >
127                     <div id="{{ index }}"
128                         class$="{{buildClass(item.image_url)}}"
129                         onmousedown="changeMainPicture(this)"
130                         ontouchstart="changeMainPicture(this)">
131                     <!-- Houston, we do not have a problem anymore  ... -->
132                     </div>
133                 </div>
```

Maintenant, on aimerait savoir quelle image est sélectionnée, pour cela nous allons l'entourer d'un bord rouge.

Nous allons donc ajouter une fonction appelée lors de la sélection d'image :

```
211     function changeMainPicture(urli){
212         encadrerImageSlider(urli.id);
213         $('#messages').iviewer('loadImage',  urli.style.background.split('(')[1].split(')')[0]);
214         $('#messages').iviewer('fit');
215     }
216
217     var currentPicture = 0;
218     function encadrerImageSlider(number){
219         document.querySelectorAll(".imageContainer")[currentPicture].style.border = "none";
220         document.querySelectorAll(".imageContainer")[number].style.border = "2px solid red";
221         currentPicture = number;
222     }
```

Il ne manque plus qu'à sélectionner la première image lors de l'initialisation, pour cela on ajoute une ligne dans le *setTimeout* de la fonction *'attached'* du constructeur Polymer :

```
174             var self = this;
175             setTimeout(function () {
176                 if(self.messages.length > 0) {
177                     $('#messages').iviewer({src: self.messages[0].image_url, ui_disabled: true});
178                     encadrerImageSlider(0);
179                 }
180                 self._updateWidthSlider();
181             }, 100);
```

Nous allons maintenant passer à la dernière fonctionnalité de notre application : le zoom / dézoom lorsqu'on double tap sur l'image.

On télécharge la librairie nécessaire :

```
bower install hammerjs --save
```

et on n'oublie pas d'inclure le fichier dans notre *index.html* :

```
16      <script src="./bower_components/hammerjs/hammer.min.js"></script>
```

Nous allons tout d'abord ajouter une propriété à notre composant dans le constructeur Polymer :

```
154         properties: {
155             messages: {
156                 type: Array,
157                 value: []
158             },
159             userName: {
160                 type:String,
161                 value: 'Contact Name'
162             },
163             tapState: {
164                 type: Boolean,
165                 value: false
166             }
167         },
```

et toujours dans le *setTimeout* de la fonction *attached*, nous allons ajouter :

```
184                 var hammer = new Hammer(
185                         document.querySelector('#messages'),
186                         {transform_always_block: true, taps:2}
187                 );
188                 hammer.on('doubletap', function () {
189                     if (self.tapState == false) {
190                         $('#messages').iviewer('zoom_by', 1.5);
191                         self.set('tapState', true);
192                     } else {
193                         $('#messages').iviewer('fit');
194                         self.set('tapState', false);
195                     }
196                 });
```

Avec quel outil de développement, de tests ? Comment teste-t-on les capacités d’adaptations ?

Nous avons utilisé Webstorm 10 pour le développement.
Nous avons testé sur Google Chrome, un Wiko Highway (Android 4.4), un Sony M2 (Android 5.1).
Nous avons également testé (grâce à la console Google) sur la plupart des résolutions / orientations d'écran. Chrome permet également de servir de débugger pour une application en cours d'exécution sur un téléphone branché à l’ordinateur. Cette fonctionnalité permet de débugger l’application du téléphone comme s’il s’agissait d’une page web classique.

Nous n'avons pas testé sur d'autres navigateurs car la compatibilité avec Polymer est compromise.

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

```
git clone https://github.com/delmotte/WhatSnap.git

cd WhatSnap/app/

cordova platform add android

cordova plugin add cordova-plugin-whitelist

cordova plugin add cordova-plugin-device-motion

cd www/

bower install

cd ..

cordova run android
```