



PAGES WEB POUR TERMINAUX MOBILES

BACHOUR Peter

28 Mai 2020

Table des matières

1	Introduction	3
2	Exercice 1 - Etiquettes Formulaires	4
3	Exercice 2 - Media Queries	6
4	Exercice 3 - Détection De Fonctionnalités	9
5	Exercice 4 - Stockage Local	11
6	Exercice 5 - Géolocalisation	13
7	Conclusion	15

Table des figures

1	CSS pour l'étiquette formulaire	4
2	CSS pour les écrans inférieures à 480px	5
3	Capture d'écran d'un écran inférieur à 480px	5
4	Capture d'écran d'un écran inférieur à 800px	6
5	Capture d'écran d'un écran supérieur à 800px	6
6	CSS pour les écrans inférieurs à 800px	7
7	Ajout des fichiers dans le .html	9
8	Contenu du fichier detection_de_fonctionnalites.js	9
9	Contenu du fichier detection_de_fonctionnalites.css	10
10	Affichage du détecteur de fonctionnalité	10
11	Explication du code JavaScript du Stockage Local	11
12	Affichage des données	12
13	Explication du code JavaScript de la Géolocalisation	13
14	Affichage de la Géolocalisation	14
15	Code qui permet d'afficher la distance à l'ESIREM	14

1 Introduction

Les terminaux mobiles disposent de plus petits écrans que les ordinateurs standards et sont tactiles. Donc, les sites web doivent être **responsive** à la taille de l'écran du terminal mobile. En effet, récemment on a pu voir une évolution des tailles d'écrans donc de plus en plus de tailles différentes.

2 Exercice 1 - Etiquettes Formulaires

Le CSS est un langage informatique utilisé sur l'internet pour mettre en forme les fichiers HTML. Dans notre cas on ajoute le code suivant dans un fichier intitulé **etiquette_formulaire.css** : En effet, ce code peut être divisé en deux

```
h1 {  
    font-size: 18pt;  
}  
  
label {  
    width: 100px;  
    text-align: right;  
    display: inline-block;  
    vertical-align: baseline  
}
```

FIGURE 1: CSS pour l'étiquette formulaire

blocs de code. Le premier étant **h1**, ce code s'applique sur toutes les balises **h1** (<h1> .. </h1>) et change la taille du texte à 18pt. Deuxièmement se trouve **label**, ce dernier s'applique sur les balises de type **label** (<label> .. </label>) et applique les changements suivants :

1. **width : 100px** : fixe la largeur du label à 100px (pixel).
2. **text-align : right** : fixe le text à la droite.
3. **display : inline-block** : permet de mettre les éléments sur une seule ligne.
4. **vertical-align : baseline** : fixe le label à être aligné avec la ligne de base du parent.

Pour ajouter le fichier CSS **etiquette_formulaire.css** au fichier HTML **etiquette_formulaire.html**, on ajoute la commande suivante dans le <head> .. </head> du fichier HTML :

```
<link rel="stylesheet" href="etiquette_formulaire.css">
```

Maintenant, faut prendre compte que les terminaux mobiles sont plus petits que les écrans d'ordinateurs. Pour cela, il faut changer le CSS pour le rendre dynamique avec les écrans qui sont inférieurs à 480px. L'ajout de ce block de code au fichier CSS nous permet de résoudre notre problème :

```
@media(max-width: 480px){  
  label {  
    width: auto;  
    display: flex;  
  }  
}
```

FIGURE 2: CSS pour les écrans inférieures à 480px

Grâce à **@media(max-width : 480px)**, on pourra appliquer le code ci-dessus sur les écrans qui sont inférieurs à 480px et surcharger la balise **label** donc changer le style de **width** à "auto" qui signifie de prendre la largeur du text et **display : flex** qui remplit l'espace libre disponible, d'où l'affichage devient comme ça :

Etiquettes de formulaire flexibles avec CSS

Nom d'utilisateur:

Mot de passe:

FIGURE 3: Capture d'écran d'un écran inférieur à 480px

3 Exercice 2 - Media Queries

On s'applique dans cet exercice au changement de CSS en fonction de la taille de l'écran, si c'est un écran supérieur à 800px ou inférieur à 800px.

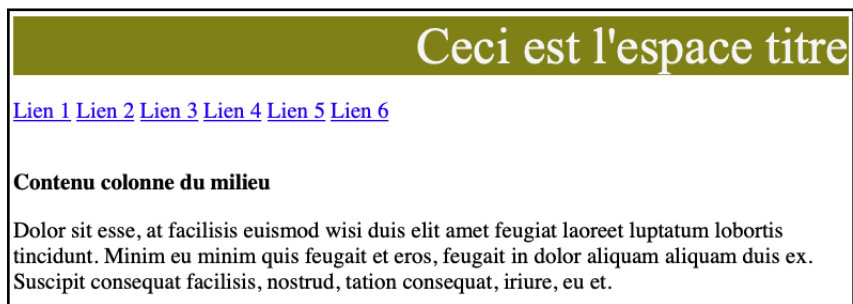


FIGURE 4: Capture d'écran d'un écran inférieur à 800px

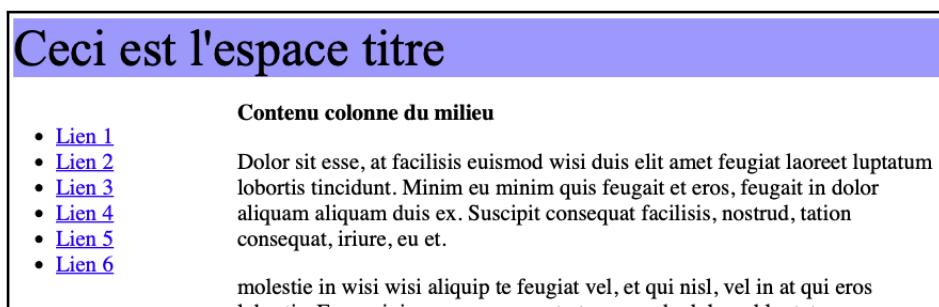


FIGURE 5: Capture d'écran d'un écran supérieur à 800px

Afin de créer le changement nécessaire, on a recours à l'ajout de ce block de code dans le fichier .css :

```
@media screen and (max-width: 800px) {  
  #titrePage {  
    font-size: 36pt;  
    font-family: "Times New Roman", Times, serif;  
    background-color: olive;  
    color: whitesmoke;  
    text-align: right;  
  }  
  #container {  
    font-size: 16pt;  
    position: relative;  
    width: 100%;  
  }  
  #colonneG {  
    width: 200px;  
    height: 100%;  
  }  
  #contenuPage {  
    margin-left: 0px;  
  }  
  #footer {  
    border: 2px gray solid;  
    padding: 5pt;  
    margin-top: 5pt;  
  }  
  ul {  
    list-style: none;  
    padding-inline-start: 0px;  
    white-space: nowrap;  
  }  
  li {  
    display: inline-flex;  
  }  
}
```

FIGURE 6: CSS pour les écrans inférieurs à 800px

En effet, les changements qui ont eu lieu sont :

1. **titrePage :**

- (a) **background-color : olive**
- (b) **color : whitesmoke**
- (c) **text-align : right** : mettre le texte à la droite

2. **ul :**

- (a) **list-style : none** : enlève le style de base de la liste.
- (b) **padding-inline-start : 0px** : fixe le padding de base à 0.
- (c) **white-space : nowrap** : enlève le style de base de la liste.

3. **li :**

- (a) **display : inline-flex** : affichage de la liste en une seule ligne.

4 Exercice 3 - Détection De Fonctionnalités

Après avoir créé les fichiers .js et .css nécessaire, on les ajoute aux fichiers .html dans le <head> .. </head> sous la façon suivante :

```
<script type="text/javascript" src="modernizr.js"></script>
<script type="text/javascript" src="detection_de_fonctionnalites.js"></script>
<script type="text/stylesheet" src="detection_de_fonctionnalites.css"></script>
```

FIGURE 7: Ajout des fichiers dans le .html

Voici le code qu'on a ajouté dans le fichier **detection_de_fonctionnalites.js** :

```
function testFonctionnalites() {
    document.querySelector("#geoloc").innerHTML = Modernizr.geolocation ? "pris en charge" : "non pris en charge";
    document.querySelector("#touch").innerHTML = Modernizr.touch ? "pris en charge" : "non pris en charge";
    document.querySelector("#svg").innerHTML = Modernizr.svg ? "pris en charge" : "non pris en charge";
    document.querySelector("#canvas").innerHTML = Modernizr.canvas ? "pris en charge" : "non pris en charge";
}
window.onload = testFonctionnalites;
```

FIGURE 8: Contenu du fichier detection_de_fonctionnalites.js

Le code ci-dessus permet de tester si la fonctionnalité qui est à l'intérieure du "querySelector" est active sur le navigateur ou non. En effet, les 4 fonctionnalités sont :

1. Géolocalisation
2. Touch Tactile
3. SVG
4. Canvas

De plus, voici le code qu'on ajoute dans le fichier `detection_de_fonctionnalites.css` :

```
.animtest, .noanimtest {  
  display: none;  
}  
  
.cssanimations .animtest {  
  display: block;  
}  
  
.no-cssanimations .noanimtest {  
  display: block;  
}
```

FIGURE 9: Contenu du fichier `detection_de_fonctionnalites.css`

Le code ci-dessus permet d'afficher les classes citées en **block**.
On aura à la fin, l'affichage suivant :

Utiliser la détection de fonctionnalités côté client

Cet exemple illustre comment utiliser Modernizr pour détecter la prise en charge de fonctionnalités dans un navigateur donné. L'ensemble du code est exécuté côté client, et utilise à la fois du JavaScript et du CSS pour détecter la prise en charge de fonctionnalités individuelles, au lieu d'employer du code côté serveur.

Fonctionnalités JavaScript

Geolocalisation: pris en charge
Evenements tactiles: non pris en charge

Fonctionnalités HTML5

SVG: pris en charge
Canvas: pris en charge

Fonctionnalités CSS3

animations CSS prises en charge
animations CSS non prises en charge

FIGURE 10: Affichage du détecteur de fonctionnalité

5 Exercice 4 - Stockage Local

Le but de cet exercice est de stocker localement des données. En effet, les fonctions JavaScript qu'il faut définir sont :

1. `storeLocalContent()`
2. `clearLocalContent()`
3. `initialize()`

Voici ci-dessous l'explication du code :

```
function initialize() {
    //Vérifier si le navigateur que l'utilisateur utilise supporte l'API Web Storage du W3C.
    var bSupportsLocal = (('localStorage' in window) && window['localStorage'] !== null);

    //Si le navigateur ne supporte pas le stockage local, un message sera afficher et un "return"
    //sera mise en exécution.
    if (!bSupportsLocal) {
        document.getElementById('infoform').innerHTML = "<p>Désolé, ce navigateur ne supporte " +
            "pas l'API Web Storage du W3C.</p>";
        return;
    }

    //Au lancement de la page, affichage du contenu du localStorage dans les elements associé
    if (window.localStorage.length !== 0) {
        document.getElementById('firstName').value = window.localStorage.getItem('firstName');
        document.getElementById('lastName').value = window.localStorage.getItem('lastName');
        document.getElementById('postCode').value = window.localStorage.getItem('postCode');
    }
}

//Sauvegarder les valeurs avec un id associé dans le local storage.
//Cette fonction est appelée lorsque l'utilisateur décide de sauvegarder des données.
function storeLocalContent(fName, lName, pCode) {
    window.localStorage.setItem('firstName', fName);
    window.localStorage.setItem('lastName', lName);
    window.localStorage.setItem('postCode', pCode);
}

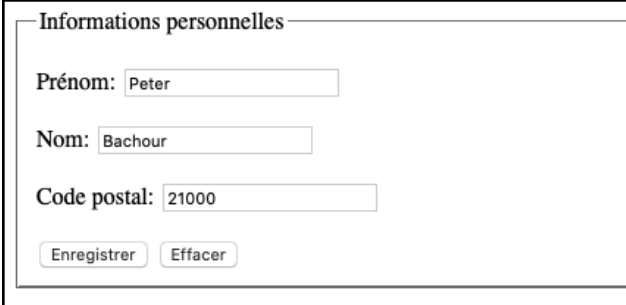
//Nettoie le stockage locale ainsi que vider les elements des valeurs effacer.
function clearLocalContent() {
    window.localStorage.setItem('firstName', "");
    window.localStorage.setItem('lastName', "");
    window.localStorage.setItem('postCode', "");

    document.getElementById('firstName').value = "";
    document.getElementById('lastName').value = "";
    document.getElementById('postCode').value = "";
}

window.onload = initialize;
```

FIGURE 11: Explication du code JavaScript du Stockage Local

Après la sauvegarde des fichiers .html et .js, il est temps de tester si mon navigateur supporte l'API Web Storage du W3C. Donc on essaie de remplir le formulaire et d'enregistrer les données. Après avoir actualisé la page, on peut remarquer que les données sont toujours sauvegardées.



The image shows a web form titled "Informations personnelles" enclosed in a black border. Inside the form, there are three text input fields, each preceded by a label. The first field is labeled "Prénom:" and contains the text "Peter". The second field is labeled "Nom:" and contains the text "Bachour". The third field is labeled "Code postal:" and contains the text "21000". Below these fields, there are two buttons: "Enregistrer" on the left and "Effacer" on the right. Both buttons have a light gray background and a thin black border.

FIGURE 12: Affichage des données

6 Exercice 5 - Géolocalisation

Le but de cet exercice est d'afficher la géolocalisation du terminal mobile avec sa distance à l'ESIREM et d'autres données. Après avoir ajouté le code nécessaire dans le fichier .js, on obtient le code suivant qui est expliqué :

```
//Fonction appelée lorsque la fenêtre se charge "window.onload = getLocation;"
function getLocation() {
    //Vérification de si le "Modernizr" a accès à la geolocation
    if (Modernizr.geolocation) {
        //appelle les fonctions geoSuccess s'il la geolocation est disponible
        // ou geoError si un problème est apparu.
        navigator.geolocation.getCurrentPosition(geoSuccess, geoError);
    }
}

//Affichage de toute les coordonnées dans les elements associés
function geoSuccess(positionInfo) {
    document.getElementById("longitude").innerHTML = positionInfo.coords.longitude;
    document.getElementById("latitude").innerHTML = positionInfo.coords.latitude;
    document.getElementById("precision").innerHTML = positionInfo.coords.accuracy;
    document.getElementById("altitude").innerHTML = positionInfo.coords.altitude;
    document.getElementById("precisionAltitude").innerHTML = positionInfo.coords.altitudeAccuracy;
    document.getElementById("cap").innerHTML = positionInfo.coords.heading;
    document.getElementById("vitesse").innerHTML = positionInfo.coords.speed;
}

//Affichage de l'erreur qui est apparu
function geoError(positionError) {
    if (errorInfo.code == 1)
        alert("L'utilisateur ne souhaite pas partager sa position");
    else if (errorInfo.code == 2)
        alert("Impossible de déterminer une position");
    else if (errorInfo.code == 3)
        alert("Délai de recherche de position trop long");
}

window.onload = getLocation;
```

FIGURE 13: Explication du code JavaScript de la Géolocalisation

Après avoir testé les résultats apparaissent sous la forme suivante (pour des raisons inconnues, des données n'ont pas été affichées) : Enfin, pour ajouter

Exemple géopositionnement

Cet exemple illustre comment utiliser la fonction de géopositionnement du terminal mobile.

Données de position:

Longitude: 5.060509616649555

Latitude: 47.33540534224718

Précision: 65

Altitude:

Précision altitude:

Cap:

Vitesse:

Distance de l'ESIREM:

FIGURE 14: Affichage de la Géolocalisation

notre distance à l'ESIREM, des modifications sont à faire dans la fonction **geo-Succes()**. En ajoutant ce block de code, la distance en kilomètre à l'ESIREM apparaît :

```
var ESIREM = { //Creation de l'objet ESIREM qui a les parametre latitude et longitude
  latitude: 47.3121519,
  longitude: 5.0039326
};
document.getElementById("distance").innerHTML = calculDistance(positionInfo.coords, ESIREM) + "km";
```

FIGURE 15: Code qui permet d'afficher la distance à l'ESIREM

7 Conclusion

En conclusion, à travers ces exercices on a appris à ajuster le style de la page par rapport à la taille de l'écran, ainsi que sauvegarder des données localement et finalement de calculer la géolocalisation de notre terminal mobile.