Pages Web pour terminaux mobiles

ITC315

Waldner Céline
06/06/2020

Table des matières

Exercice 1

On crée un fichier css que l'on lie au fichier html avec la ligne suivante :

```
<link rel="stylesheet" href="styleExo1.css"/>
```

On ajoute ensuite le code nécessaire, expliqué ci-dessous.

La ligne *h1 {font-size: 18pt;}* permet de mettre la taille de la police à 18 pour la balise h1.

L'élément *label {..}* permet de modifier le style des éléments de type label.

La ligne *width: 100px* définit la largeur du label à 100 pixels;

La ligne *text-align: right*; permet de mettre le texte de l'élément label au centre.

La ligne *display: inline-block;* permet d'afficher un bloc sur la ligne du label.

La ligne *vertical-align: baseline* permet d'afficher verticalement les deux labels avec leur blocs

Nous devons maintenant modifier le fichier css pour que les labels soient affichés au-dessus des champs du formulaire pour une largeur d'écran inférieure à 480px.

```
@media screen and (max-width: 480px) {
    label {
    width: 100px;
    text-align: right;
    display: block;
    }
}
@media screen and (min-width: 480px) {
    label {
    width: 100px;
    text-align: right;
    display: inline-block;
    vertical-align: baseline
    }
}
```

Figure 1 Modifications dans le fichier .css

La ligne *@media screen and (max-width: 480 px)* permet de spécifier une taille d'écran pour laquelle sera valide les conditions suivantes. Ici pour une largeur inférieure à 480 pixels, on affiche le label au-dessus de son bloc.

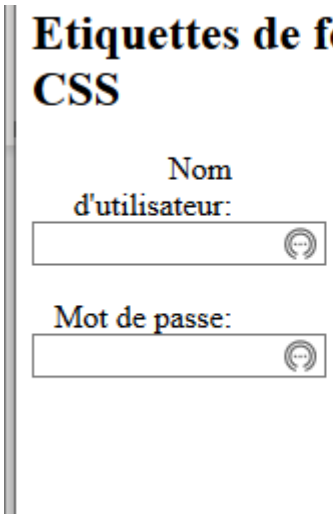


Figure 2 affichage pour une largeur d'écran inférieure à 480 px

On a ensuite un deuxième boc de code signifiant que pour une largeur supérieure à 480 pixels, on affiche les labels et leurs blocs en ligne.

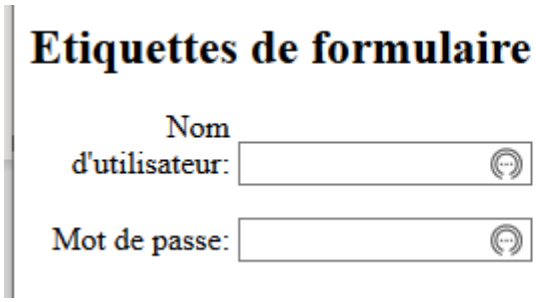


Figure 3 affichage pour une largeur d'écran supérieure à 480 px

Exercice 2

Lorsque l'on redimensionne la fenêtre, le contenu ne s'adapte pas à l'affichage.

Comme demandé, il suffit d'ajouter la ligne au code déjà existant.

```css
@media screen and (min-width: 801px) {
    #titrePage {
        font-size:36pt;
        font-family:"Times New Roman", Times, serif;
        background-color: #9999FF;
    }

    #container {
        font-size: 16pt;
        position: relative;
        width: 100%;
    }

    #colonneG {
        width: 200px;
        height: 100%;
        float:left;
    }

    #contenuPage {
        margin-left: 210px;
    }

    #footer {
        border: 2px gray solid;
        padding: 5pt;
        margin-top: 5pt;
    }
}
```

Figure 4 : fichier css pour une largeur d'écran supérieure à 801 px

Figure 5 Affichage pour une largeur d'écran supérieure à 801 px

Il faut maintenant ajouter le code de l'affichage pour une fenêtre de moins de 801 pixels.

```css
@media screen and (max-width: 800px) {
    #titrePage {
        font-size:36pt;
        font-family:"Times New Roman", Times, serif;
        background-color:#556B2F;
        color: white;
        text-align: right;
    }

    #colonneG {
        width: 300px;
        height: 100%;
    }

    li{
        display : inline;
    }
    #container {
        font-size: 16pt;
        position: relative;
        width: 100%;
    }



    #footer {
        border: 2px gray solid;
        padding: 5pt;
        margin-top: 5pt;
    }
}
```

Figure 6: fichier css pour l'affichage d'un écran d'une largeur inférieure à 801 px

On modifie la couleur d'arrière-plan, la couleur de police et l'alignement du titre de la page. On enlève le positionnement à gauche pour les liens. On ajoute un affichage en ligne pour la balise li. Enfin on enlève la marge du texte de la page.

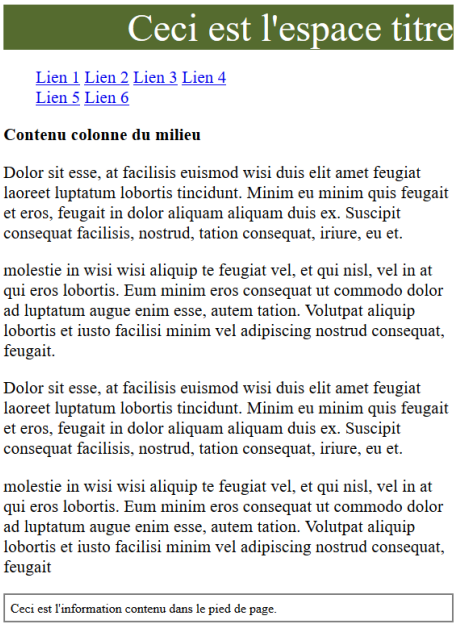


Figure 7 Affichage pour une largeur d'écran inférieure à 801 px

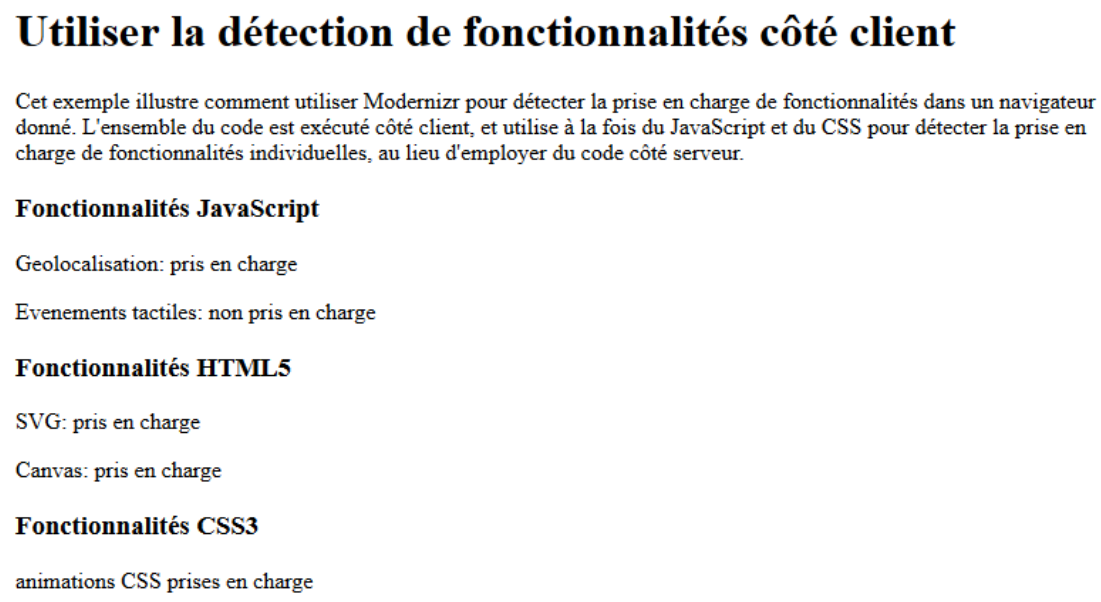
Exercice 3

On ajoute les différents fichiers dans le fichier html.

```
    <script src="modernizr.js">
    </script>
    <script src="exo3.js">
    </script>
     <link rel="stylesheet" href="styleExo3.css"/>
```

Le code dans le fichier .js ajouté permet de tester si les fonctionnalités de Géolocalisation, d'évènement tactiles, de SVG et de Canvas sont prises en charge par le côté client.

Le code dans le fichier .css ajouté permet de tester si les fonctionnalités d'animations CSS sont prises en charge par le côté client.



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

Figure 8 Affichage après l'ajout des différents fichiers

Exercice 4

Le but du fichier stockage_local.html est de stocker des informations côté client. Les fonctions JavaScript *storeLocalContent* et *clearLocalContent* doivent être définies.

La ligne *var bSupportsLocal = (('localStorage' in window) && window['localStorage'] !== null);* permet de définir le lieu de stockage.

Les lignes de code :

```
  if (!bSupportsLocal) {
        document.getElementById('infoform').innerHTML = "<p>Désolé, ce navigat
eur ne supporte pas l'API Web Storage du W3C.</p>";
        return;
    }
```

permettent d'ajouter une condition pour vérifier si le navigateur peut définir un lieu de stockage et dons si il est compatible avec l'API Web Storage.

```
    if (window.localStorage.length != 0) {
        document.getElementById('firstName').value = window.localStorage.getIt
em('firstName');
        document.getElementById('lastName').value = window.localStorage.getIte
m('lastName');
        document.getElementById('postCode').value = window.localStorage.getIte
m('postCode');
        }
```

Ces lignes permettent de récupérer les informations firstName, lastName et postCode du formulaire et de les stocker.

```
function storeLocalContent(fName, lName, pCode) {
    window.localStorage.setItem('firstName', fName);
    window.localStorage.setItem('lastName', lName);
    window.localStorage.setItem('postCode', pCode);
    }
```

Cette fonction est appelée au lancement du fichier html, elle remet la dernière valeur rentrée dans le formulaire.

Voici la fonction permettant de nettoyer le stockage local. On met à vide les champs du formulaire, ainsi que les valeurs stockées.

```
function clearLocalContent() {
    document.getElementById('firstName').value = '';
    document.getElementById('lastName').value = '';
    document.getElementById('postCode').value = '';
    window.localStorage.setItem('firstName', '');
    window.localStorage.setItem('lastName', '');
    window.localStorage.setItem('postCode', '');
}
```

Après avoir sauvegardé les données, si on change de page lorsque l'on revient les données sont toujours dans les champs du formulaire.

Exercice 5

Ce fichier contient une fonction de géopositionnement.

La fonction getLocation() sera appelé au lancement du fichier html.

```
 if (Modernizr.geolocation) {
        navigator.geolocation.getCurrentPosition(geoSuccess, geoError);
    }
```

Ces lignes permettent de tester si on peut récupérer la géolocalisation du navigateur, si oui, on la récupère sinon il y a un message d'erreur.

```
function geoSuccess(positionInfo) {
    document.getElementById("longitude").innerHTML = positionInfo.coords.longi
tude;
    document.getElementById("latitude").innerHTML = positionInfo.coords.latitu
de;
    document.getElementById("precision").innerHTML = positionInfo.coords.accur
acy;
    document.getElementById("altitude").innerHTML = positionInfo.coords.altitu
de;
    document.getElementById("precisionAltitude").innerHTML = positionInfo.coor
ds.altitudeAccuracy;
    document.getElementById("cap").innerHTML = positionInfo.coords.heading;
    document.getElementById("vitesse").innerHTML = positionInfo.coords.speed;
}

function geoError(positionError) {
    if (positionError.code == 1)
    alert("L'utilisateur ne souhaite pas partager sa position");
    else if (positionError.code == 2)
    alert("Impossible de déterminer une position");
    else if (positionError.code == 3)
    alert("Délai de recherche de position trop long");
}
```

Le but de la fonction *geoSuccess* est de récupérer la géolocalisation du navigateur et le but de la fonction *geoError* permet de savoir pour la géolocalisation a échouée.

Après avoir testé le fichier, on obtient une longitude et une latitude avec une précision. Les autres champs n'affichent rien.

Afin d'afficher la distance séparant la position du navigateur et Esirem il suffit de rajouter ces lignes de code

```
const esirem = {latitude: 47.3121519, longitude: 5.0039326};
    document.getElementById("distance").innerHTML = calculDistance(positionInf
o.coords, esirem);
```

On définit une variable esirem avec ses coordonnées puis on appelle la fonction *calculDistance* que l'on met dans la div correspondante.