

Gestion des fichiers

Ceci est une documentation préliminaire, traduite par [Christine Dubois \(Agilcom\)](#) et sujette à changement

Dans le chapitre précédent, vous avez appris à stocker des données dans une base de données. Une autre méthode consiste à travailler avec des fichiers texte. En effet, les fichiers texte représentent un moyen simple de stockage de données pour un site web (ce type de fichier est parfois appelé un fichier plat). Des fichiers texte peuvent être dans différents formats, tels que .txt, .xml, ou .csv (valeurs séparées par des virgules).

A la fin de ce chapitre, vous saurez :

- Comment créer un fichier texte et y écrire des données.
- Comment ajouter des données à un fichier existant.
- Comment lire un fichier et afficher son contenu.
- Comment supprimer des fichiers d'un site web.
- Comment permettre aux utilisateurs de charger (faire un upload) un ou plusieurs fichiers.

Les fonctionnalités de programmation ASP.NET suivantes vous seront présentées dans ce chapitre:

- L'objet File, qui permet de gérer des fichiers.
- L'assistant FileUpload.
- L'objet Path, qui fournit des méthodes pour manipuler les chemins et les noms de fichier.

Remarque: Si vous voulez télécharger des images et les manipuler (par exemple les couper ou les redimensionner), référez-vous au [chapitre 7: Travailler avec des images](#).

Créer un fichier texte et y écrire des données

Si vous souhaitez stocker des données dans un fichier texte, vous pouvez utiliser la méthode *File.WriteAllText* pour spécifier le fichier à créer et les données à écrire. Dans la procédure qui suit, vous allez créer une page qui contient un formulaire simple constitué de trois zones de texte (nom, prénom et adresse e-mail) et un bouton d'envoi. Lorsque l'utilisateur soumet le formulaire, ses entrées sont stockées dans un fichier texte.

1. Créez un nouveau fichier nommé *UserData.cshtml*.
2. Remplacez son contenu par ce qui suit:

```
@{
    var result = "";
    if (IsPost)
    {
        var firstName = Request["FirstName"];
        var lastName = Request["LastName"];
        var email = Request["Email"];
        var userData = firstName + "," + lastName +
            "," + email + Environment.NewLine;
    }
}
```

```

        var dataFile = Server.MapPath("~/App_Data/data.txt");
        File.WriteAllText(@dataFile, userData);
        result = "Information saved.";
    }
}
<!DOCTYPE html>
<html>
<head>
    <title>Write Data to a File</title>
</head>
<body>
    <form id="form1" method="post">
        <div>
            <table>
                <tr>
                    <td>First Name:</td>
                    <td><input id="FirstName" name="FirstName" type="text"
/></td>

                </tr>
                <tr>
                    <td>Last Name:</td>
                    <td><input id="LastName" name="LastName" type="text" /></td>
                </tr>
                <tr>
                    <td>Email:</td>
                    <td><input id="Email" name="Email" type="text" /></td>
                </tr>
                <tr>
                    <td></td>
                    <td><input type="submit" value="Submit"/></td>
                </tr>
            </table>
        </div>
        <div>
            @if(result != ""){
                <p>Result: @result</p>
            }
        </div>
    </form>
</body>
</html>

```

Les balises HTML créent le formulaire avec les trois zones de texte. Le code utilise la propriété *IsPost* pour déterminer s'il s'agit bien d'un postback de la page, avant de commencer le traitement.

La première chose à faire est de récupérer les entrées de l'utilisateur et de les attribuer à des variables. Puis on concatène les valeurs des différentes variables en une seule chaîne de caractères délimitées par des virgules, et on la stocke dans une variable différente. Notez que le séparateur virgule est une chaîne contenue entre guillemets (","), parce que vous encapsulez littéralement une virgule dans la chaîne globale. À la suite des données concaténées, on ajoute *Environment.NewLine*. Cela ajoute un saut de ligne (un caractère de nouvelle ligne). Le résultat de la concaténation ressemble à ceci :

David, Jones, davidj@contoso.com
(Avec un saut de ligne invisible à la fin.)

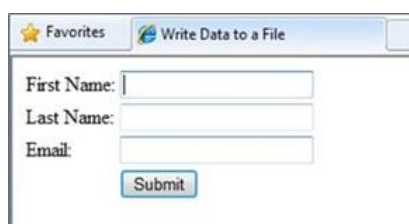
Le code créé ensuite une variable qui contient l'emplacement et le nom du fichier dans lequel on veut stocker les données. La définition de la localisation du fichier nécessite une manipulation spéciale. Dans les sites Web, l'utilisation de chemins absolus tels que C:\dossier\fichier.txt pour identifier des fichiers sur le serveur Web n'est pas conseillée. En effet, si le site est déplacé, un chemin absolu devient incorrect. En outre, pour un site hébergé (par opposition à un site sur votre ordinateur), la plupart du temps vous ne savez même pas à l'avance quel est le chemin correct à utiliser dans le code.

Mais parfois (comme maintenant, pour écrire dans un fichier), vous avez besoin d'un chemin d'accès complet. La solution est d'utiliser la méthode *MapPath* de l'objet *Server*. A l'exécution, elle retourne le chemin d'accès de votre site Web. Pour obtenir le chemin de la racine du site, utilisez le caractère "~" (suivi si besoin du nom du sous-dossier, par exemple ~/App_Data/, pour obtenir le chemin d'accès au sous-dossier *App_Data*.) Il suffit ensuite de concaténer toute autre information supplémentaire sur le résultat renvoyé par la méthode *MapPath* pour créer le chemin d'accès complet. Dans cet exemple, on y ajoute le nom du fichier. (Pour en savoir plus sur la façon de travailler avec des chemins de fichiers et de dossiers, référez-vous au [chapitre 2 - Introduction à la programmation Web ASP.NET en utilisant la syntaxe Razor](#))

Le fichier de l'exemple sera enregistré dans le dossier *App_Data*. Ce dossier est un dossier spécial d'ASP.NET qui est utilisé pour stocker des fichiers de données, comme nous l'avons vu au [chapitre 5 - Travailler avec les données](#).

La méthode *WriteAllText* de l'objet *File* écrit les données dans le fichier. Cette méthode attend deux paramètres, le nom (avec son chemin) du fichier et les données que l'on souhaite y écrire. Notez que le nom du premier paramètre est préfixé par le caractère @. Cela indique à ASP.NET que vous lui fournissez une chaîne littérale, et que les caractères tels que "/" ne doivent pas être interprétés d'une manière particulière. (Pour plus d'informations, voir le [chapitre 2](#).)

1. Exécutez la page dans un navigateur.



The image shows a web browser window with a single tab titled "Write Data to a File". The browser's address bar is empty. The page content consists of a form with three text input fields labeled "First Name:", "Last Name:", and "Email:". Below these fields is a "Submit" button. The browser's interface includes a "Favorites" button on the left and a search bar on the right.

2. Saisissez des valeurs dans les champs puis cliquez **Submit**.
3. Fermez le navigateur.
4. Revenez au projet et rafraîchissez l'affichage.
5. Ouvrez le fichier data.txt. Les données que vous avez soumises dans le formulaire sont inscrites dans le fichier.



6. Fermez le fichier data.txt.

Ajouter des données à un fichier existant

Dans l'exemple précédent, vous avez utilisé *WriteAllText* pour créer un fichier texte et y écrire une ligne de données. Si vous appelez la méthode à nouveau en lui passant le même nom de fichier, celui-ci sera complètement écrasé. Or une fois que vous avez créé un fichier, il est utile de pouvoir y ajouter de nouvelles données à la suite de son contenu initial. Pour cela, on utilise la méthode *AppendAllText* de l'objet *File*.

1. Dans le site, faites une copie du fichier *UserData.cshtml* et nommez celle-ci *UserDataMultiple.cshtml*.
2. Remplacez le bloc de code situé avant la balise d'ouverture `<html>`, avec le bloc de code suivant:

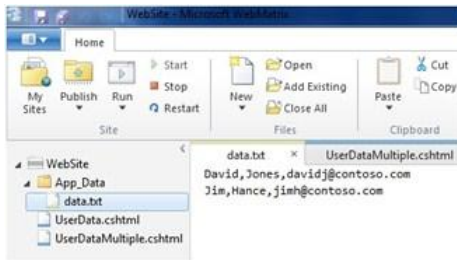
```
@{
    var result = "";
    if (IsPost)
    {
        var firstName = Request["FirstName"];
        var lastName = Request["LastName"];
        var email = Request["Email"];

        var userData = firstName + "," + lastName +
            "," + email + Environment.NewLine;

        var dataFile = Server.MapPath("~/App_Data/data.txt");
        File.AppendAllText (@dataFile, userData);
        result = "Information saved.";
    }
}
```

La seule différence avec le code précédent est qu'on utilise ici la méthode *AppendAllText*, au lieu de la méthode *WriteAllText*. Les méthodes sont semblables, à ceci près que *AppendAllText* ajoute les données à la fin du fichier. Comme pour la méthode *WriteAllText*, *AppendAllText* crée le fichier s'il n'existe pas déjà.

3. Exécutez la page dans un navigateur.
4. Saisissez des valeurs dans les champs puis cliquez **Submit**.
5. [Optionnel] Saisissez d'autres valeurs et soumettez à nouveau le formulaire.
6. Rebasculez dans votre projet, faites un clic droit sur le dossier du projet, puis cliquez **Refresh**.
7. Ouvrez le fichier data.txt. Il contient les nouvelles données que vous venez d'entrer.



Lire et afficher des données à partir d'un fichier

Même si vous ne rencontrez pas le besoin d'écrire des données dans un fichier texte, vous aurez probablement besoin de pouvoir lire des données de l'un d'eux. Pour ce faire, vous pouvez de nouveau utiliser l'objet *File*. L'objet *File* peut lire chaque ligne du fichier individuellement (séparée par des sauts de ligne) ou lire des éléments individuels, quelle que soit la façon dont ils sont séparés.

La procédure suivante montre comment lire et afficher les données que vous avez créées dans l'exemple précédent.

1. Créez un nouveau fichier nommé *DisplayData.cshtml*.
2. Remplacez son contenu comme suit :

```
@{
    var result = "";
    Array userData = null;
    char[] delimiterChar = {','};

    var dataFile = Server.MapPath("~/App_Data/data.txt");

    if (File.Exists(dataFile)) {
        userData = File.ReadAllLines(dataFile);
        if (userData == null) {
            // Empty file.
            result = "The file is empty.";
        }
    }
    else {
        // File does not exist.
        result = "The file does not exist.";
    }
}
<!DOCTYPE html>

<html>
<head>
    <title>Reading Data from a File</title>
</head>
<body>
    <div>
        <h1>Reading Data from a File</h1>
        @result
        @if (result == "") {
            <ol>
                @foreach (string dataLine in userData) {
                    <li>
                        User
                    </li>
                }
            </ol>
        }
    }
}
```

```

        @foreach (string dataItem in dataLine.Split(delimiterChar))
    {
        <li>@dataItem</li >
    }
    </ul>
</li>
}
</ol>
}
</div>
</body>
</html>

```

Le code commence par lire le fichier que vous avez créé dans l'exemple précédent et place le résultat dans une variable nommée *userData*, via l'appel de méthode *ReadAllLines*:

```
File.ReadAllLines (dataFile)
```

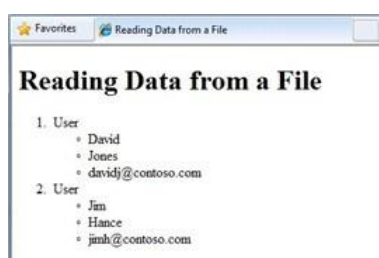
La lecture du fichier est soumise à condition dans une instruction *if*. En effet, lorsque vous voulez lire un fichier, une bonne pratique consiste à utiliser la méthode *File.Exists* de façon à vérifier au préalable que le fichier est disponible. Le code vérifie également si le fichier est vide.

Le corps de la page contient deux boucles *foreach*, imbriquées l'une dans l'autre. La boucle *foreach* extérieur récupère ligne à ligne le contenu du fichier de données. Dans cet exemple, les lignes sont définies par des sauts de ligne dans le fichier (donc chaque élément de données est sur sa propre ligne). La boucle crée un nouvel élément ** à l'intérieur d'une liste ordonnée **. Elle affiche enfin un compteur d'exécution (en utilisant la variable *userCount*). Notez que le compteur démarre à zéro.

La boucle *foreach* intérieure décompose chaque ligne de données en éléments (champs) en utilisant la virgule comme séparateur (Si on reprend l'exemple précédent, chaque ligne du fichier contient trois champs séparés par une virgule : le nom, le prénom et l'adresse e-mail). Le code de cette boucle imbriquée crée également une liste ** et affiche un élément de liste pour chaque champ dans la ligne de données.

Le code illustre comment utiliser deux types de données, un *array* (tableau) et un *char* (caractère). Le type *array* est utilisé au niveau de l'appel de la méthode *File.ReadAllLines* qui retourne les données sous la forme d'un tableau. Le type *char* est quant à lui nécessaire parce que la méthode *Split* retourne un tableau dans lequel chaque élément est de type *char*.

3. Exécutez la page dans un navigateur. Les données que vous avez inscrites dans les exemples précédents s'affichent.



Afficher des données d'un fichier Microsoft Excel délimités par des virgules

Vous pouvez utiliser Microsoft Excel pour enregistrer les données contenues dans une feuille de calcul dans un fichier délimité par des virgules (.csv). Le fichier est alors enregistré au format texte brut, et non au format Excel. Chaque ligne de la feuille de calcul est séparée par un saut de ligne dans le fichier texte, et chaque élément de données est séparé par une virgule. Pour lire ce fichier, vous pouvez réutiliser le code de l'exemple précédent, en changeant simplement le nom du fichier de données.

Supprimer des fichiers

Pour supprimer des fichiers de votre site, vous pouvez utiliser la méthode *File.Delete*. La procédure suivante montre comment permettre aux utilisateurs de supprimer une image (fichier.jpg) d'un dossier d'images, à partir du nom du fichier.

1. Dans le site, créer un sous-dossier nommé **images**.



2. Copiez un ou plusieurs fichiers .jpg dans ce dossier images.
3. A la racine du site, créez un nouveau fichier nommé *FileDelete.cshtml*.
4. Remplacez son contenu par ce qui suit :

```
@{
    bool deleteSuccess = false;
    var photoName = "";
    if (IsPost) {
        photoName = Request["photoFileName"] + ".jpg";
        var fullPath = Server.MapPath("~/images/" + photoName);

        if (File.Exists(fullPath))
        {
            File.Delete(fullPath);
            deleteSuccess = true;
        }
    }
}
<!DOCTYPE html>
<html>
    <head>
        <title>Delete a Photo</title>
    </head>
    <body>
        <h1>Delete a Photo from the Site</h1>
        <form name="deletePhoto" action="" method="post">
```

```

    <p>File name of image to delete (without .jpg extension):
    <input name="photoFileName" type="text" value="" />
    </p>
    <p><input type="submit" value="Submit" /> </p>
</form>

@if(deleteSuccess) {
    <p>
    @photoName deleted!
    </p>
}
</body>
</html>

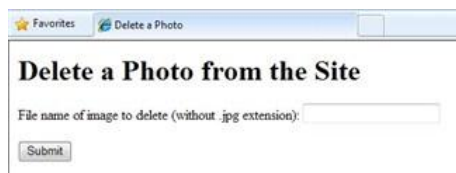
```

Cette page contient un formulaire à partir duquel les utilisateurs peuvent saisir le nom d'un fichier image. Le nom seul du fichier leur est demandé, sans l'extension du fichier ; cette précaution évite qu'un utilisateur supprime de manière arbitraire des fichiers sur votre site.

Le code récupère le nom du fichier que l'utilisateur a saisi, puis construit un chemin complet. Pour créer le chemin d'accès, le code utilise le chemin de votre site web (renvoyé par la méthode *Server.MapPath*), le nom du dossier images, le nom que l'utilisateur a fourni, et l'extension «.jpg» en tant que chaîne littérale.

Pour supprimer le fichier, le code invoque ensuite la méthode *File.Delete*, en lui passant le chemin d'accès complet que vous avez construit. Au bas de la page, est affiché un message de confirmation de la suppression du fichier.

1. Exécutez la page dans un navigateur.



2. Saisissez le nom du fichier à supprimer, puis cliquez **Submit**. Si le fichier est supprimé avec succès, le message de confirmation est affiché au bas de la page.

Permettre aux utilisateurs de charger un fichier

L'assistant *FileUpload* permet aux utilisateurs de charger (upload) des fichiers sur votre site web. La procédure ci-dessous illustre le procédé pour construire une page à partir de laquelle l'utilisateur peut charger un fichier sur le site.

1. Dans le dossier *App_Data*, créez un nouveau dossier et nommez-le **Uploadedfiles**.
2. A la racine, créez un nouveau fichier nommé *FileUpload.cshtml*.
3. Remplacez son contenu avec le texte suivant:

```

@{
    var fileName = "";

```



```

    if (IsPost) {
        var fileSavePath = "";
        var uploadedFile = Request.Files[0];
        fileName = Path.GetFileName(uploadedFile.FileName);
        fileSavePath = Server.MapPath("~/App_Data/UploadedFiles/" +
            fileName);
        uploadedFile.SaveAs(fileSavePath);
    }
}
<!DOCTYPE html>
<html>
    <head>
        <title>FileUpload - Single-File Example</title>
    </head>
    <body>
        <h1>FileUpload - Single-File Example</h1>
        @FileUpload.GetHtml(
            initialNumberOfFiles:1,
            allowMoreFilesToBeAdded:false,
            includeFormTag:true,
            uploadText:"Upload")
        @if (IsPost) {
            <span>File uploaded!</span><br/>
        }
    </body>
</html>

```

Le corps de la page utilise l'assistant *FileUpload* pour créer la zone de chargement et les boutons utiles (que vous reconnaissez probablement) :

FileUpload



Les propriétés à définir pour l'assistant *FileUpload* précisent que vous souhaitez une zone de chargement unique et que vous souhaitez nommer le bouton *Upload* (Charger). (Vous ajouterez plusieurs zones de chargement pour charger plus d'un fichier plus loin dans ce chapitre.)

Lorsque l'utilisateur clique sur *Submit*, le code en haut de la page récupère le fichier et l'enregistre. L'objet *Request* que vous utilisez habituellement pour obtenir des valeurs de champs de formulaire comprend également un tableau (array) de fichiers qui contient le fichier (ou les fichiers) qui ont été chargé(s). Vous pouvez ainsi récupérer chaque fichier de manière individuelle à partir de ce tableau. Par exemple, pour obtenir le premier fichier qui a été chargé, cela donne *Request.Files[0]*, pour obtenir le second fichier, codez *Request.Files[1]*, et ainsi de suite. (Rappelez-vous que le compteur de position commence habituellement à zéro.)

Après avoir récupéré un fichier particulier, on l'attribue en général à une variable (ici, *UploadedFile*) de façon à pouvoir le manipuler. Pour déterminer le nom du fichier chargé, utilisez sa propriété *FileName*. Toutefois, lorsqu'un fichier est chargé ainsi, *FileName* contient le nom d'origine du fichier de l'utilisateur, c'est-à-dire incluant l'intégralité du chemin d'accès. Il pourrait ressembler à ceci:

C:\Users\Public\Sample.txt

Ces informations de chemin, correspondant au chemin sur l'ordinateur de l'utilisateur, et non à celui de votre serveur, ne sont pas particulièrement intéressantes. Le nom du fichier seul (sample.txt) est utile. Pour extraire d'un chemin complet uniquement le nom du fichier, utilisez la méthode *Path.GetFileName*, comme ceci:

```
Path.GetFileName (uploadedFile.FileName)
```

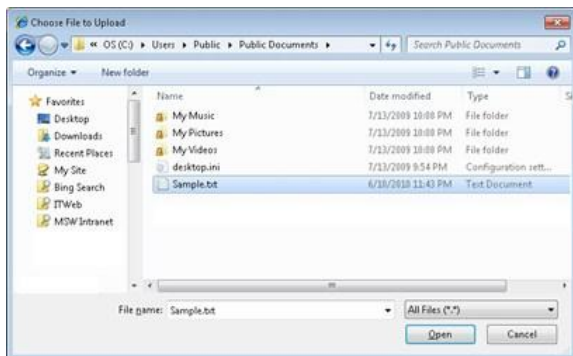
L'objet *Path* est un véritable utilitaire comprenant de nombreuses méthodes de ce type, que vous pouvez utiliser pour supprimer des chemins, combiner des chemins, etc.

Une fois que l'on sait récupérer le nom du fichier chargé, il suffit de construire le chemin dans lequel on souhaite charger le fichier sur le site. Pour ce faire, on combine *Server.MapPath*, les noms de dossiers adéquats (dans l'exemple *App_Data/Uploadedfiles*), et le nom du fichier. Avec la méthode *SaveAs*, on enregistre le fichier ainsi dans l'emplacement voulu sur le site.

4. Exécutez la page dans un navigateur.



5. Cliquez **Browse** (Parcourir), puis sélectionnez le fichier à charger.



La zone de texte à côté du bouton **Browse** se remplit avec le chemin complet du fichier.

6. Cliquez **Upload**.
7. Dans le site, faites un clic-droit sur le dossier du projet et sélectionnez **Refresh**.
8. Ouvrez le dossier **Uploadedfiles**. Le fichier que vous avez chargé se trouve dans le dossier.



Permettre aux utilisateurs de charger plusieurs fichiers

Dans l'exemple précédent, un utilisateur ne peut charger qu'un seul fichier à la fois. Mais l'assistant *FileUpload* permet également le chargement simultané de plusieurs fichiers. C'est très utile pour des scénarios tels que le chargement de photos, où le chargement des fichiers un à un est très fastidieux. L'exemple qui suit montre comment permettre aux utilisateurs le chargement de deux fichiers à la fois, mais la même technique peut être utilisée pour télécharger plus de fichiers que cela.

1. Créez une nouvelle page nommée *FileUploadMultiple.cshtml*.
2. Remplacez son contenu par le suivant :

```
@{
    var message = "";
    if (IsPost) {
        var fileName = "";
        var fileSavePath = "";
        int numFiles = Request.Files.Count;
        message = "File upload complete. Total files uploaded: " +
            numFiles.ToString();
        for(int i =0; i < numFiles; i++) {
            var uploadedFile = Request.Files[i];
            fileName = Path.GetFileName(uploadedFile.FileName);
            fileSavePath = Server.MapPath("~/App_Data/UploadedFiles/" +
                fileName);
            uploadedFile.SaveAs(fileSavePath);
        }
    }
}
<!DOCTYPE html>
<html>
    <head><title>FileUpload - Multiple File Example</title></head>
<body>
    <form id="myForm" method="post"
        enctype="multipart/form-data"
        action="">
        <div>
            <h1>File Upload - Multiple-File Example</h1>
            @if (!IsPost) {
                @FileUpload.GetHtml(
                    initialNumberOfFiles:2,
                    allowMoreFilesToBeAdded:true,
                    includeFormTag:true,
                    addText:"Add another file",
                    uploadText:"Upload")
            }
            <span>@message</span>
        </div>
    </form>
</body>
</html>
```

Dans cet exemple, l'assistant *FileUpload* utilisé dans le corps de page est configuré pour permettre aux utilisateurs de charger deux fichiers par défaut. Si *allowMoreFilesToBeAdded* est positionné à *true*, l'assistant affiche un lien qui permet à l'utilisateur d'ajouter davantage de boîtes de chargement si besoin.



Pour traiter les fichiers soumis par l'utilisateur, le code utilise la même technique de base que vous avez utilisée dans l'exemple précédent, à savoir récupérer les fichiers à partir de *Request.Files*, puis les enregistrer (cela inclut également tout le travail nécessaire pour obtenir le nom du fichier et le chemin correct). La nouveauté ici est que vous ne savez pas à l'avance combien de fichiers l'utilisateur va vouloir charger. Pour déterminer combien de fichiers il a décidé de charger, utilisez *Request.Files.Count*.

Connaissant le nombre de fichiers chargés, il suffit alors de parcourir le tableau *Request.Files*, de récupérer chaque fichier l'un après l'autre et de l'enregistrer. Pour boucler un nombre déterminé de fois à travers une collection, on utilise une boucle *for*, comme ceci:

```
for(int i =0; i < numFiles; i++) {  
    var uploadedFile = Request.Files[i];  
    fileName = Path.GetFileName(uploadedFile.FileName);  
    // etc.  
}
```

La variable *i* est juste un compteur temporaire qui commence à zéro et s'incrémente jusqu'à la limite supérieure de votre choix. Dans l'exemple, la limite supérieure est le nombre total de fichiers à charger. Du fait que le compteur démarre à zéro, ce qui est classique dans les scénarios utilisant un compteur avec ASP.NET, la limite supérieure est en fait le nombre total de fichiers - 1. (Si trois fichiers sont téléchargés, le compteur ira de zéro à 2.)

3. Exécutez la page dans un navigateur. Le navigateur affiche la page et deux zones de chargement.
4. Sélectionnez les deux fichiers à charger.
5. Cliquez **Add another file** (ajouter un autre fichier). La page affiche une nouvelle zone de chargement.



6. Cliquez sur **Upload**.
7. Dans le site, faites un clic-droit sur le dossier du projet, puis cliquez sur **Refresh**.
8. Ouvrez le dossier **Uploadedfiles** pour vérifier que les fichiers ont été chargés avec succès.

Ressources supplémentaires

[Exporting to a CSV File](#)