Prise en main

Page principale

Pour lancer le serveur et afficher la page principale :

Clic droit sur Index.aspx > View in Browser

Tags disponibles

- B{} = qui permet de mettre en gras le texte entre accolade.
- #{}# = pour que le texte entre accolade ne soit pas interprété.

Exemple : « Un B{texte en gars} est possible grâce au tag #{B{ }}# »

Arbre syntaxique

En appuyant sur le bouton « Render », la représentation de l'arbre syntaxique s'affiche où l'on peut distinguer l'ensemble des fils à partir de la racine de l'arbre.

La représentation est identique à celle d'un parcours classique d'arborescence de dossier.

ATTENTION : si le texte source contient des retours à la ligne, la représentation graphique de l'arbre est quelque peu altérée.

Exercices

Ajouter un tag

Le but de cet exercice est d'ajouter un tag qui sera supporté par l'analyseur syntaxique.

Vous pouvez ajouter le tag que vous souhaitez, par exemple, mettre en italique, en majuscule/minuscule, souligner le texte etc.

Pour cela, il faut créer le nouveau tag dans le fichier Render.cs et de l'ajouter à la liste des tags de la classe SyntaxeTree.

Vous pouvez maintenant tester votre nouveau tag.

Ajouter des boutons de mise en forme

NB : Dans cet exercice, les technologies AJAX et JQuery sont utilisées.

Le but de cet exercice est d'ajouter un bouton sur l'interface utilisateur pour permettre de mettre automatiquement les balises du tag autour du texte sélectionné.

Voici les étapes à effectuer :

- Ajouter le bouton sur la page d'index
- Créer un fichier JS dans le dossier Script et l'ajouter à la page d'index.
- Dans ce fichier sera écrit l'appel AJAX lié au bouton créé :
  - L'url est sous la forme
    - "Index.aspx/${nom de la méthode invoquée dans Index.aspx}"
  - Le nom de la méthode invoquée doit être dans le fichier Index.aspx.cs, en « public static » annotée « `[System.Web.Services.WebMethod]` »

- La méthode doit prendre en paramètre une variable de type string dont le nom est identique à celui de la clé JSON dans les data envoyées.
- La méthode retourne un type string qui sera écrit dans « response.d »
- ATTENTION : Le comportement d'un bouton au click soumet par défaut le formulaire dans lequel il est. N'oubliez pas d'annuler cet évènement avec e.preventDefault() ;

➢ Corps de l'appel AJAX :

```
$.ajax({
            type: "POST",
            url : ,
            data:                                '{text                                :
"'+$('#SourceCode_TextBox').textrange('get','text')+'"}',
            contentType: "application/json; charset=utf-8",
            dataType: "json",
            success: function(response) {
                $('#SourceCode_TextBox').textrange('replace', response.d);
            },
            error : function(response) {
                console.log(response);
            },
            failure: function(response) {
                console.log(response);
            }
        });
```

Rendering

L'objectif est de pouvoir générer du code à partir de l'arbre syntaxique représentant la saisie de l'utilisateur.

Par exemple, il est possible de produire le code HTML de l'arbre pour avoir un rendu visuel des tags.

- ➢ Créez la classe HtmlRenderer dans un nouveau fichier du projet TagManagerLib
- ➢ La classe implémente l'interface Renderer
- ➢ Pour représenter les tags HTML résultats, on va créer la classe abstraite AbstractHtmlTag :
  - Elle prend en constructeur un AbstractTag qui sera le tag auquel il sera lié.
  - Un getter sur ce tag : {get ;}
  - Deux méthodes abstraites :
    - OpenHtmlTag qui retournera la balise html d'ouverture
    - CloseHtmlTag qui retournera la balise html de fermeture
    - Ainsi que les deux méthodes `GetHashCode et Equals` basées sur OpenHtmlTag et CloseHtmlTag. Elles seront de préférence finale.
- ➢ Ensuite, pour chaque tag connue, il faut créer sa classe de tag équivalente html qui étendra AbstractHtmlTag
- ➢ Une fois ces classes déclarées, il faut renseigner dans le constructeur de la classe HtmlRenderer une hashmap (Dictionary en C#) qui prendra en clé un AbstractTag et en valeur son AbstractHtmlTag équivalent.
- ➢ Voici le corps de la méthode Render

```
        public string Render(SyntaxTree tree)
        {
            string result = "";
            foreach (Node child in tree.Root.Children)
            {
                result += renderNode(child);
            }
            return result;
        }

        private string renderNode(Node node)
        {
            if (node is InnerTextNode)
            {
                return HttpUtility.HtmlEncode(((InnerTextNode)node).Text);
            }

            string result = "";

            AbstractHtmlTag tagHtml = null;
            if (node.Tag != null && _htmlTags.ContainsKey(node.Tag))
            {
                tagHtml = _htmlTags[node.Tag];
            }

            if (tagHtml != null)
            {
                result = tagHtml.OpenHtmlTag;
            }

            if (new TagNoProcess().Equals(node.Tag))
            {
                result += renderNoPrecessNode(node);
            }
            else
            {
                foreach (Node child in node.Children)
                {
                    result += renderNode(child);
                }
            }

            if (tagHtml != null)
            {
                result += tagHtml.CloseHtmlTag;
            }

            return result;

        }

        private string renderNoPrecessNode(Node node)
        {
            string result = "";
            foreach (Node child in node.Children)
            {
                if (child.Tag != null)
                {
                    result += child.Tag.OpenTag + renderNoPrecessNode(child)
+ child.Tag.CloseTag;
                }
```

```
                else
                {
                    result += child.ToString() + renderNoPrecessNode(child);
                }
            }
            return HttpUtility.HtmlEncode(result);
        }
    }
```

Export de fichier

HTML

L'objectif est de pouvoir exporter le texte source dans un fichier sous format HTML.

Pour ce faire, il faut ajouter un bouton d'export sur la page principale.

Ensuite, créer la méthode d'export qui sera exécutée au click du bouton dans le fichier Index.aspx.cs.

Cette méthode sera de type « public void » et prendra en paramètre `(object sender, EventArgs e)` pour qu'elle soit acceptée en tant qu'évenement, voici le corps de cette méthode :

```
            string path = Path.GetTempPath() + "/render.html";
            TextWriter tw = new StreamWriter(path);
            // write a line of text to the file
            tw.WriteLine(#######);
            // close the stream
            tw.Close();

            string name = Path.GetFileName(path);
            Response.AppendHeader("content-disposition","attachment;
filename=" + name);
            Response.ContentType = "text/html";
            Response.WriteFile(path);
            Response.End();
```

Cette méthode sera utilisée dans la méthode `Page_Load` en écrivant : `(id du bouton dans l'aspx).Click += (nom de la méthode) ;`

Pour aller plus loin

JSON

Créer un renderer JSON qui transformera l'arbre syntaxique en objet JSON.

Faire un Export/Import de fichier JSON.

Match des tags en Regex

Les tags sont analysés dans le texte source en utilisant la méthode IndexOf de la classe String.

Il serait intéressant de récupérer les tags à l'aide de regex pour ainsi pouvoir faire un tag « color » par exemple.

Ce tag serait sous la forme « C[]{} » avec entre crochet la couleur et entre accolade le texte à colorer.

Une regex entre les crochets définirait les couleurs (par ex. « [a-z] ») qui serait reprit dans la balise html « <font color=(couleur issue de la regex) ».

Ex : Le tag C[green]{Texte vert} donnerait en HTML Text vert