

Introduction à Handlebars avec NodeJS

Joseph AZAR – Nov 2024

Introduction

Dans ce tutoriel, nous allons voir comment utiliser le moteur de template Handlebars avec Node.js et Express. Je vais couvrir ce que sont les moteurs de template (*Template Engines*) et comment Handlebars peut être utilisés pour créer des applications Web Server Side Rendered (SSR).

Nous verrons également comment configurer Handlebars avec le framework Express.js et comment utiliser les assistants intégrés pour créer des pages dynamiques. Enfin, nous verrons comment développer un assistant personnalisé (*custom helper*) en cas de besoin.

Qu'est-ce qu'un moteur de template ?

Dans les années 90, lorsque Internet a été introduit dans le monde, il était principalement utilisé à des fins scientifiques telles que la publication d'articles de recherche et comme canal de communication entre les universités et les scientifiques. La plupart des pages Web à l'époque étaient statiques. Une page Web statique est la même pour chaque utilisateur et ne change pas d'un utilisateur à l'autre. Si quelque chose devait être changé sur une page, cela aurait été fait manuellement.

Dans le monde moderne, les choses sont beaucoup plus interactives et adaptées à chaque utilisateur. Aujourd'hui, presque tout le monde a accès à Internet. La plupart des applications Web d'aujourd'hui sont dynamiques. Par exemple, sur Facebook, vous et moi verrons des flux d'actualités très différents une fois connectés. Pour chaque personne, la page suivra le même modèle (c'est-à-dire des publications séquentielles avec des noms d'utilisateur au-dessus), mais le contenu sera différent.

C'est le travail d'un moteur de template - le modèle (template) pour le fil d'actualités est défini puis, en fonction de l'utilisateur actuel et de la requête à la base de données, le modèle (template) est rempli avec le contenu reçu.

Nous pouvons utiliser des moteurs de template à la fois dans le backend et le front-end. Si nous utilisons un moteur de template dans le backend pour générer le code HTML, nous appelons cela le Server Side Rendered (SSR).

Handlebars

Handlebars est populaire pour les modèles (tempaltes) back-end et front-end. Par exemple, le framework front-end populaire **Ember** utilise Handlebars comme moteur de template.

Handlebars est une extension du langage de modèle (template) **Mustache**, qui se concentre principalement sur la simplicité et la création de modèles minimale.

Handlebars avec Node.js

Pour commencer, créez un dossier vide, ouvrez le terminal dans ce dossier (à l'aide de webstorm ou visual studio), puis exécutez `npm init` pour créer un projet Node.js vide avec les paramètres par défaut.

```
$ npm init
```

Avant de commencer, nous devons installer les modules Node.js requis. Vous pouvez installer les modules `express` et `express-handlebars` en exécutant :

```
$ npm install express express-handlebars
```

Ensuite, créons la structure de répertoires Handlebars par défaut. Le dossier des vues (`views`) contient tous les modèles Handlebars :



Le dossier "**layouts**" à l'intérieur du dossier des vues contiendra les mises en page ou les enveloppes (*wrappers*) de modèles. Ces mises en page contiendront la structure HTML, les *style sheets* et les scripts partagés entre les templates.

Le fichier "**main.hbs**" est la mise en page principale. Le fichier "**home.hbs**" est un exemple de modèle Handlebars sur lequel nous allons nous baser.

Nous ajouterons d'autres modèles et dossiers au fur et à mesure.

Dans notre exemple, nous utiliserons un seul fichier script pour garder cela simple. Importons les bibliothèques requises dans notre fichier **index.js** et créons une application Express :

```
const express = require('express');
const hbsengine = require('express-handlebars');
const port = 3000;
const app = express();

app.listen(port, () => {
  console.log(`Le serveur écoute sur le port ${port}`);
});
```

Maintenant, nous pouvons configurer express-handlebars comme moteur de vue :

```
const express = require('express');
const hbsengine = require('express-handlebars');
const port = 3000;
const app = express();

app.engine('hbs', hbsengine.engine({ config: {
  defaultLayout: 'main',
  extname: '.hbs'
}}));

app.set('view engine', 'hbs');

app.listen(port, () => {
  console.log(`Le serveur écoute sur le port ${port}`);
});
```



modèle principal qui sera utilisé dans d'autres modèles

Extension de fichiers

Par défaut, l'extension des modèles Handlebars est **.handlebars**. Mais dans les paramètres ici, nous l'avons changé en **.hbs** via le drapeau (flag) **extname** car il est plus court.

Incluons les scripts et les styles Bootstrap dans la mise en page **main.hbs**. Copiez ce qui suit et collez-le dans **layouts/main.hbs** :

```

<html lang="en">
<head>
  <!-- <meta> tags -->
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">
  <title>Handlebars Demo</title>
</head>

<body>
<div class="container">
  {{{body}}}
</div>

<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"></scrip
t>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"></scr
ipt>
</body>
</html>

```

Et maintenant, changeons notre **home.hbs** pour inclure un message :

```
<h1>Bonjour Handlebars</h1>
```

Pour pouvoir accéder à cette page, nous devons configurer un gestionnaire de requêtes. Définissons-le au chemin racine :

```

const express = require('express');
const hbengine = require('express-handlebars');
const port = 3000;
const app = express();

app.engine('hbs', hbengine.engine( config: {
  defaultLayout: 'main',
  extname: '.hbs'
})));

app.set('view engine', 'hbs');

app.get('/', (req, res) => {
  res.render('home');
});

app.listen(port, () => {
  console.log(`Le serveur écoute sur le port ${port}`);
});

```

Nous pouvons exécuter l'application avec **node index.js** dans la console, mais nous pouvons également choisir d'utiliser un outil comme **nodemon**. Avec **nodemon**, nous n'avons pas besoin de redémarrer le serveur à chaque fois que nous apportons une modification - lorsque nous modifions le code, nodemon actualisera le serveur.

Installons-le :

```
$ npm i --save-dev nodemon
```

Et l'exécution de l'application avec nodemon se fait via :

```
$ nodemon index.js
```

Visitons notre application via le navigateur :



Caractéristiques du Handlebars

Afin de présenter certaines des fonctionnalités de Handlebars, nous allons créer un flux de médias sociaux. Le flux extraira les données d'un tableau simple, simulant une base de données.

Le flux contiendra des publications avec des images et des commentaires. S'il n'y a pas de commentaires sur une image, un message "Soyez le premier à commenter ce message" apparaîtra.

Mettons à jour notre **home.hbs** pour commencer :

```

<nav class="navbar navbar-dark bg-dark">
  <a class="navbar-brand" href="#">Handlebars Demo</a>
</nav>

<div class="posts">
  <div class="row justify-content-center">
    <div class="col-lg-7" style="margin-top: 50px;">
      <div class="card">
        
        <div class="card-body">
          <h5 class="card-title">Posté par Elon Musk</h5>

          <ul class="list-group">
            <li class="list-group-item">C'est censé être un commentaire</li>
            <li class="list-group-item">C'est censé être un commentaire</li>
          </ul>
        </div>
      </div>
    </div>
  </div>
</div>

```

```

<nav class="navbar navbar-dark bg-dark">
  <a class="navbar-brand" href="#">Handlebars Demo</a>
</nav>

```

```

<div class="posts">
  <div class="row justify-content-center">
    <div class="col-lg-7" style="margin-top: 50px;">
      <div class="card">

```

```

        
        <div class="card-body">
          <h5 class="card-title">Posté par Elon Musk</h5>

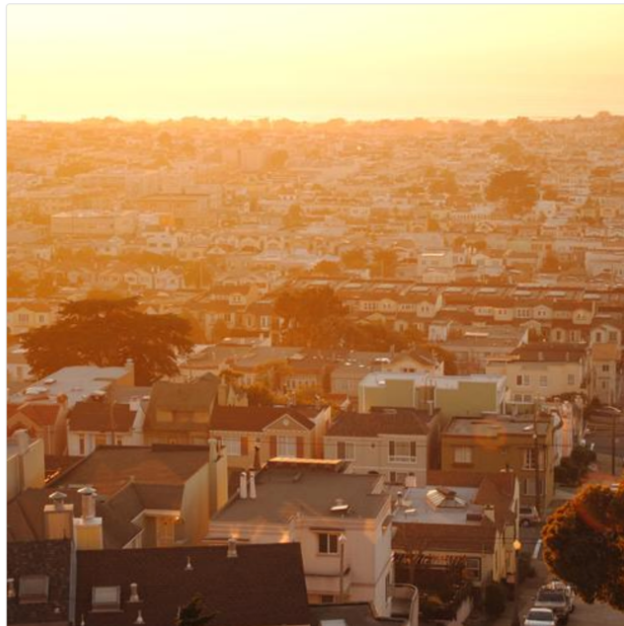
```

```

          <ul class="list-group">
            <li class="list-group-item">C'est censé être un
commentaire</li>
            <li class="list-group-item">C'est censé être un
commentaire</li>
          </ul>
        </div>
      </div>
    </div>
  </div>
</div>

```

Notre page ressemble maintenant à ceci :



Posté par Elon Musk

C'est censé être un commentaire

C'est censé être un commentaire

Passer des paramètres aux modèles/templates

Maintenant, supprimons ces valeurs codées en dur de la page elle-même et transmettons-les du script `index.js` à la page. Celles-ci seront remplacées ultérieurement par des valeurs de commentaire dans le tableau :

```

index.js
6 app.engine('hbs', hbs.engine({ config: {
7   defaultLayout: 'main',
8   extname: '.hbs'
9 } }));
10
11 app.set('view engine', 'hbs');
12
13 app.get('/', function (req, res) {
14   res.render('home', {
15     post: {
16       author: 'Elon Musk',
17       image: 'https://picsum.photos/500/500',
18       comments: []
19     }
20   });
21 });
22
23 app.listen(port, () => {
24   console.log(`Le serveur écoute sur le port ${port}`);
25 });

```

L'objet **"post"** contient des champs tels que **author**, **image**, et **comments**. Nous pouvons référencer la publication dans notre modèle Handlebars par **{{post}}** :

```

index.js  home.hbs
1 <nav class="navbar navbar-dark bg-dark">
2   <a class="navbar-brand" href="#">Handlebars Demo</a>
3 </nav>
4
5 <div class="posts">
6   <div class="row justify-content-center">
7     <div class="col-lg-7" style="...">
8       <div class="card">
9
10        
12        <div class="card-body">
13          <h5 class="card-title">Posté par {{post.author}}</h5>
14
15          <ul class="list-group">
16            <li class="list-group-item">C'est censé être un commentaire</li>
17            <li class="list-group-item">C'est censé être un commentaire</li>
18          </ul>
19        </div>
20      </div>
21    </div>
22  </div>
23 </div>

```

En référençant ces valeurs avec le gestionnaire qui affiche la page, elles sont insérées côté serveur et l'utilisateur reçoit un code HTML apparemment statique avec ces valeurs déjà présentes.

Utiliser les conditions

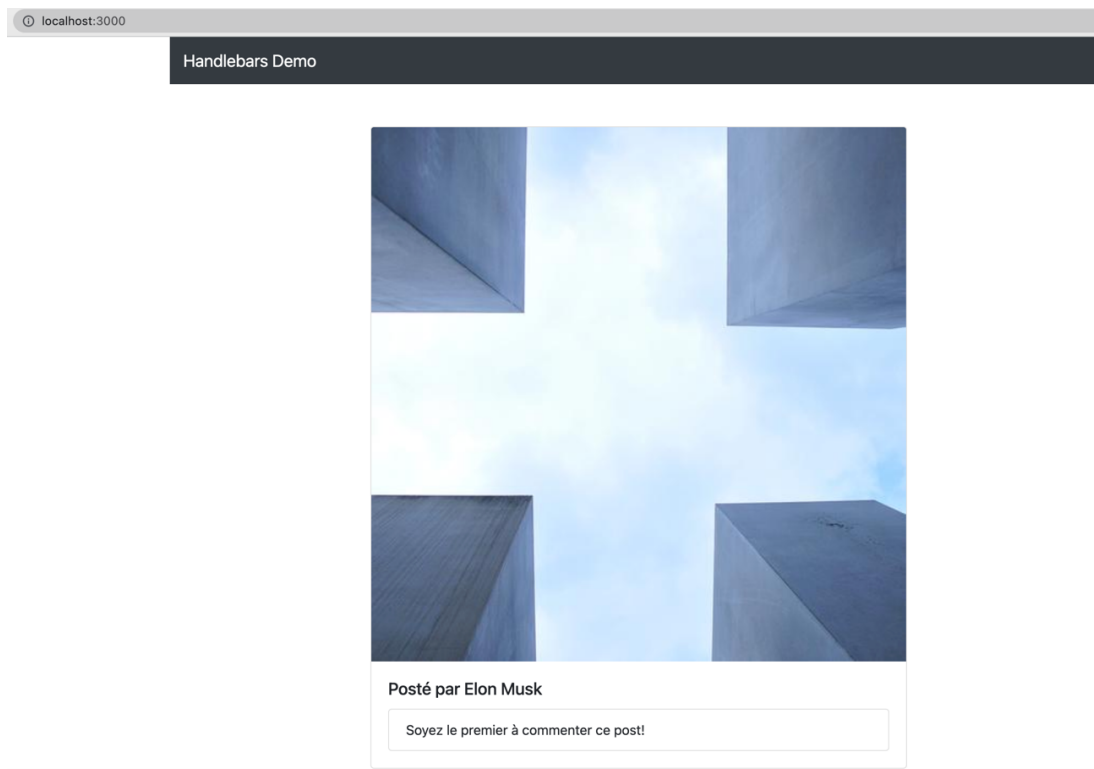
Puisque nous avons une logique conditionnelle, c'est-à-dire afficher les commentaires s'ils sont présents et un message s'ils ne le sont pas, voyons comment nous pouvons utiliser les conditions dans les modèles Handlebars :

```
1 <nav class="navbar navbar-dark bg-dark">
2   <a class="navbar-brand" href="#">Handlebars Demo</a>
3 </nav>
4 <div class="posts">
5   <div class="row justify-content-center">
6     <div class="col-lg-7" style="...">
7       <div class="card">
8
9         
11       <div class="card-body">
12         <h5 class="card-title">Posté par {{post.author}}</h5>
13         {{#if post.comments}}
14           <ul class="list-group">
15             <!-- Afficher la logique des commentaires -->
16
17           </ul>
18         {{else}}
19           <ul class="list-group">
20             <li class="list-group-item">Soyez le premier à commenter ce post!</li>
21           </ul>
22         {{/if}}
23       </div>
24     </div>
25   </div>
26 </div>
27 </div>
```

```
{{#if post.comments}}
  <ul class="list-group">
    <!-- Afficher la logique des commentaires -->
```

```
  </ul>
{{else}}
  <ul class="list-group">
    <li class="list-group-item">Soyez le premier à commenter ce
post!</li>
  </ul>
{{/if}}
```

Maintenant, vous ne devriez voir que la section "Soyez le premier à commenter ce post!" affichée sur votre page puisque le tableau de commentaires est vide :



Le **#if** est une aide intégrée (**built-in helper**) dans Handlebars. Si l'instruction **if** renvoie **true**, le bloc à l'intérieur du bloc **#if** sera rendu. Si **false**, **undefined**, **null**, **""**, **0** ou **[]** sont renvoyés, le bloc ne sera pas rendu.

Notre tableau est vide (**[]**) donc le bloc n'est pas rendu.

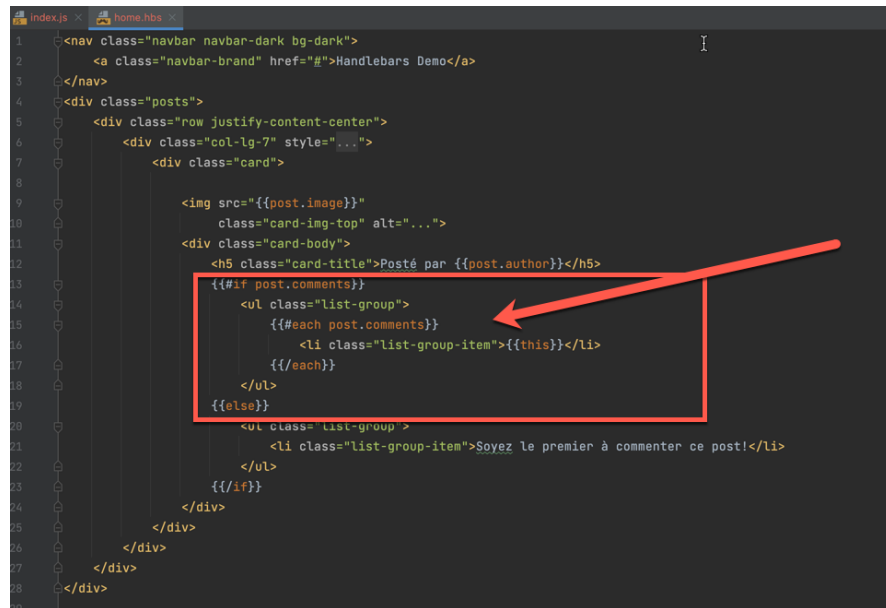
#if n'accepte qu'une seule condition et vous ne pouvez pas utiliser la syntaxe de comparaison JavaScript (**===**). Si vous devez utiliser plusieurs conditions ou une syntaxe supplémentaire, vous pouvez créer une variable dans le code et la transmettre au modèle. De plus, vous pouvez définir votre propre assistant, ce que nous ferons dans la dernière section.

Utiliser des boucles

Puisqu'un post peut contenir plusieurs commentaires, nous aurons besoin d'une boucle pour les parcourir tous et les afficher. Commençons par remplir notre tableau avec quelques commentaires :

```
app.get('/', function (req, res) {  
  res.render('home', {  
    post: {  
      author: 'Elon Musk',  
      image: 'https://picsum.photos/500/500',  
      comments: ["C'est mon premier commentaire",  
                 "Ceci est mon deuxième commentaire",  
                 "Ceci est mon troisième commentaire"]  
    }  
  });  
});
```

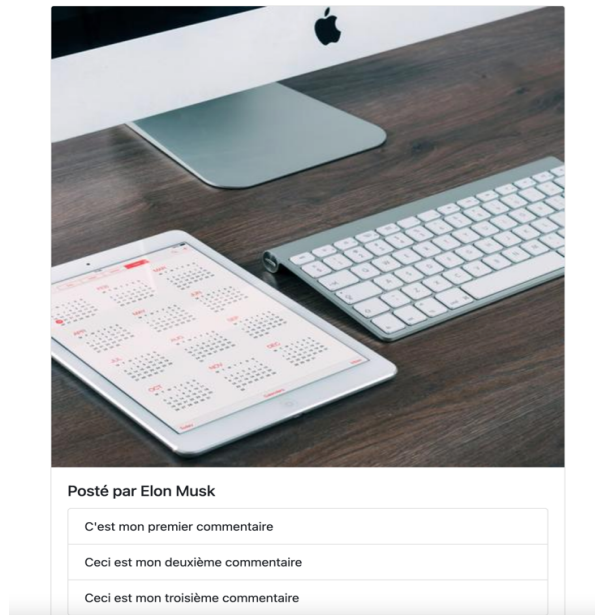
Et maintenant, dans notre modèle, nous allons utiliser la boucle `#each` pour les parcourir toutes :



```
1 <nav class="navbar navbar-dark bg-dark">
2   <a class="navbar-brand" href="#">Handlebars Demo</a>
3 </nav>
4 <div class="posts">
5   <div class="row justify-content-center">
6     <div class="col-lg-7" style="border: 1px solid #ccc; padding: 10px; margin: 10px auto;">
7       <div class="card">
8         
10        <div class="card-body">
11          <h5 class="card-title">Posté par {{post.author}}</h5>
12          {{#if post.comments}}
13            <ul class="list-group">
14              {{#each post.comments}}
15                <li class="list-group-item">{{this}}</li>
16              {{/each}}
17            </ul>
18          {{else}}
19            <ul class="list-group">
20              <li class="list-group-item">Soyez le premier à commenter ce post!</li>
21            </ul>
22          {{/if}}
23        </div>
24      </div>
25    </div>
26  </div>
27 </div>
28 </div>
```

```
{{#if post.comments}}
  <ul class="list-group">
    {{#each post.comments}}
      <li class="list-group-item">{{this}}</li>
    {{/each}}
  </ul>
{{else}}
  <ul class="list-group">
    <li class="list-group-item">Soyez le premier à commenter ce post!</li>
  </ul>
{{/if}}
```

À l'intérieur de la boucle `#each`, vous pouvez utiliser **this** pour référencer l'élément qui se trouve dans l'itération actuelle. Dans notre cas, il fait référence à une chaîne qui est ensuite rendue :



Si vous avez un tableau d'objets, vous pouvez également accéder à n'importe quel attribut de cet objet. Par exemple, s'il existe un tableau de personnes, vous pouvez simplement utiliser **this.name** pour accéder au champ de nom.

Maintenant, modifions nos paramètres de modèle pour qu'ils contiennent plusieurs posts :

```
app.get('/', function (req, res) {
  res.render('home', {
    posts: [
      {
        author: 'Elon Musk',
        image: 'https://picsum.photos/500/500',
        comments: ["C'est mon premier commentaire",
          "Ceci est mon deuxième commentaire",
          "Ceci est mon troisième commentaire"]
      },
      {
        author: 'Bill Gates',
        image: 'https://picsum.photos/500/500?2',
        comments: [
        ]
      }
    ]
  });
});
```

Maintenant, nous pouvons également mettre un **#each** pour parcourir les posts. Remplacez le contenu de **home.hbs** par ce qui suit :

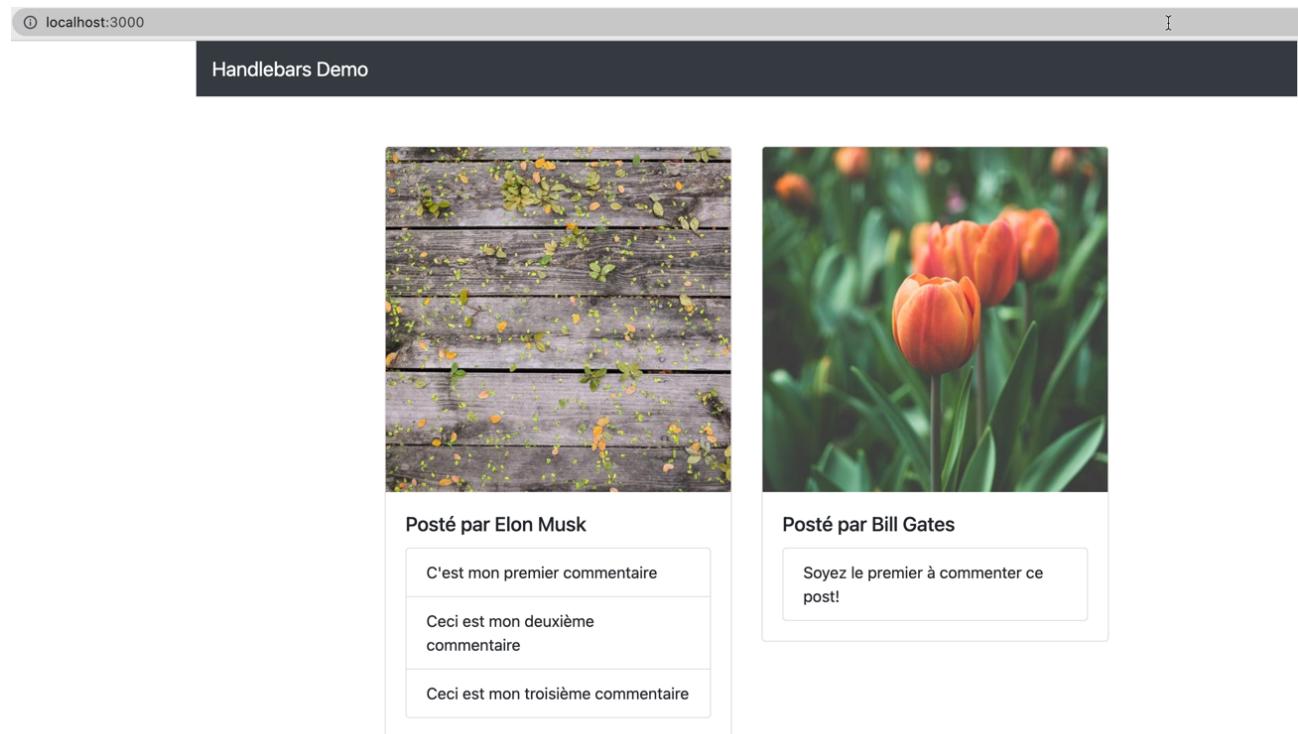
```
<nav class="navbar navbar-dark bg-dark">
  <a class="navbar-brand" href="#">Handlebars Demo</a>
</nav>
<div class="posts">
  <div class="row justify-content-center">
    {{#each posts}}
      <div class="col-lg-4" style="margin-top: 50px;">
        <div class="card">

          
```

```

<div class="card-body">
  <h5 class="card-title">Posté par {{this.author}}</h5>
  {{#if this.comments}}
    <ul class="list-group">
      {{#each this.comments}}
        <li class="list-group-item">{{this}}</li>
      {{/each}}
    </ul>
  {{else}}
    <ul class="list-group">
      <li class="list-group-item">Soyez le premier à
commenter ce post!</li>
    </ul>
  {{/if}}
</div>
</div>
</div>
{{/each}}
</div>
</div>

```

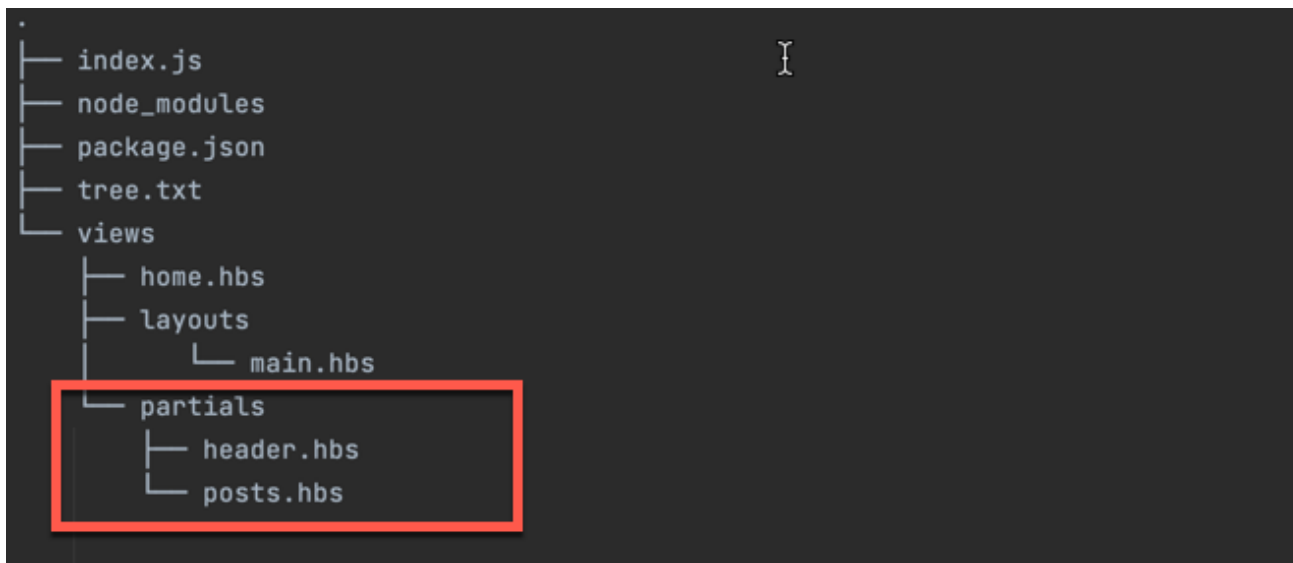


Utilisation de Partial

Pratiquement toutes les pages Web contiennent des sections différentes. À la base, il s'agit des sections En-tête, Corps et Pied de page. Étant donné que l'en-tête et le pied de page sont généralement partagés entre de nombreuses pages, l'avoir dans toutes les pages Web deviendra bientôt extrêmement ennuyeux et tout simplement redondant.

Heureusement, nous pouvons utiliser Handlebars pour diviser ces sections en modèles et simplement inclure ces modèles en tant que "**partials**" dans les pages elles-mêmes.

Dans notre cas, puisque nous n'avons pas de pied de page, créons un fichier **header.hbs** et un fichier **posts.hbs** dans un répertoire **partials** :



Ensuite, nous déplacerons le code d'en-tête dans le fichier **header.hbs** :

```
<nav class="navbar navbar-dark bg-dark">
  <a class="navbar-brand" href="#">Handlebars Demo</a>
</nav>
```

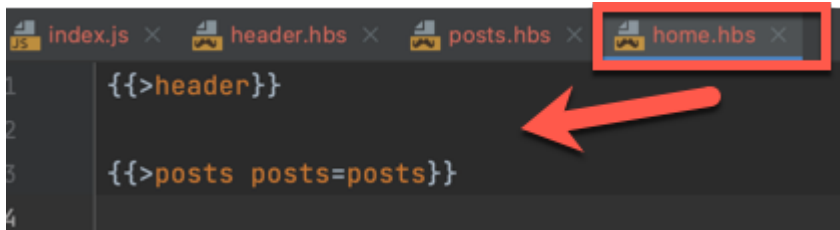
Et le code du flux dans le fichier **posts.hbs** :

```
<div class="posts">
  <div class="row justify-content-center">
    {{#each posts}}
      <div class="col-lg-4" style="margin-top: 50px;">
        <div class="card">
          
          <div class="card-body">
            <h5 class="card-title">Posté par {{this.author}}</h5>
            {{#if this.comments}}
              <ul class="list-group">
                {{#each this.comments}}
                  <li class="list-group-item">{{this}}</li>
                {{/each}}
              </ul>
            {{else}}
              <ul class="list-group">
                <li class="list-group-item">Soyez le premier à
commenter ce post!</li>
              </ul>
            {{/if}}
          </div>
        </div>
      </div>
    {{/each}}
  </div>
</div>
```

Et maintenant, nous pouvons les inclure dans le fichier **home.hbs** :

```
{{>header}}
```

```
{{>posts posts=posts}}
```



L'utilisateur ne verra pas de différence, mais notre fichier **home.hbs** est beaucoup plus propre maintenant. Cela devient super utile lorsque vous avez des pages Web complexes.

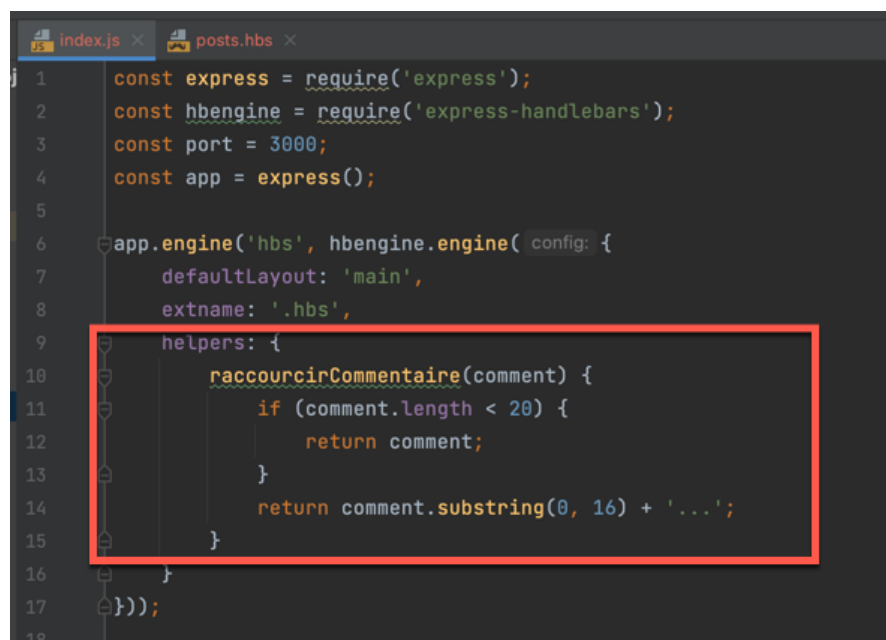
Ici, nous avons simplement inclus le fichier **header.hbs** et passé un paramètre **posts** au champ **posts** du fichier **posts.hbs**.

Ce que cela fait, c'est qu'il transmet les posts de notre gestionnaire au paramètre **posts** dans le fichier de page **posts.hbs**.

Création d'un Custom Helper

Comme vous pouvez le voir sur la page, nous pouvons avoir un seul commentaire qui consomme plus d'une ligne. Créons un assistant personnalisé (custom helper) pour résumer ce texte.

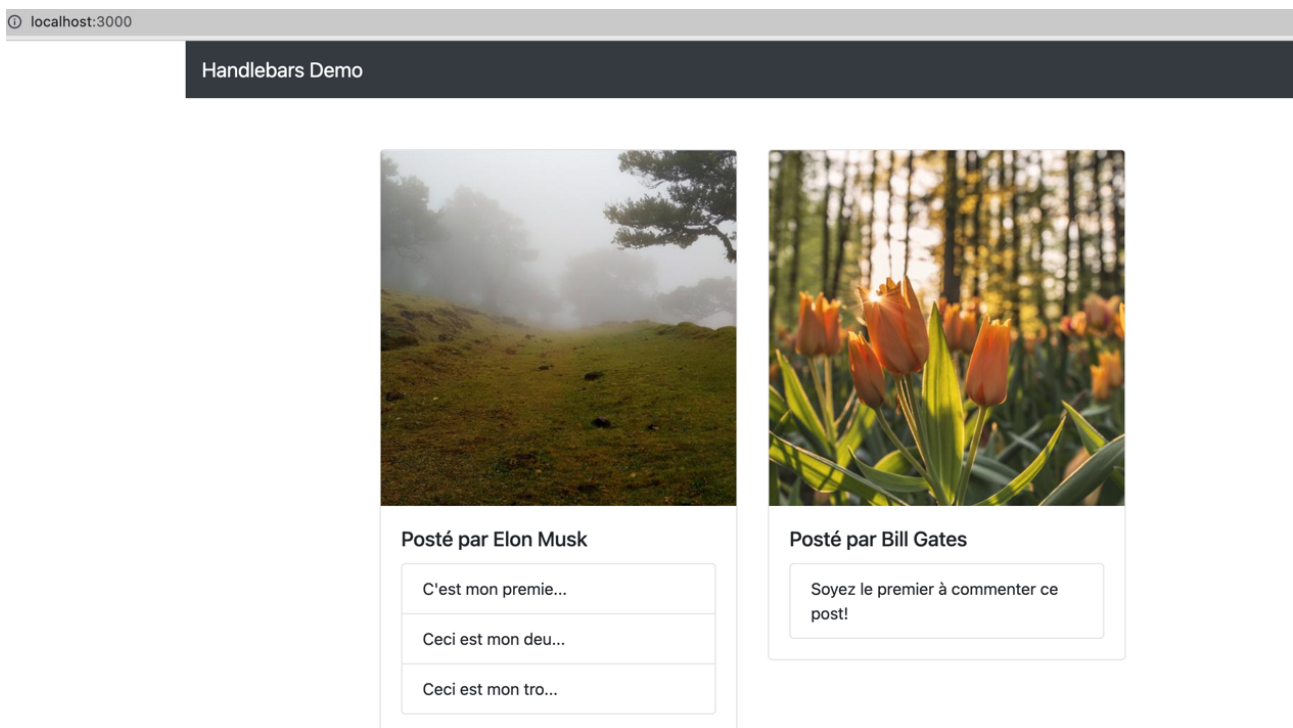
Pour ce faire, dans la configuration du Handlebars, nous pouvons définir nos fonctions d'assistance. Dans notre cas, nous limiterons les commentaires à 12 caractères :



Utilisons maintenant cet assistant dans notre modèle **posts.hbs** pour raccourcir les commentaires :

```
1 <div class="posts">
2   <div class="row justify-content-center">
3     {{#each posts}}
4       <div class="col-lg-4" style="...">
5         <div class="card">
6
7           
9
10          <div class="card-body">
11            <h5 class="card-title">Posté par {{this.author}}</h5>
12            {{#if this.comments}}
13              <ul class="list-group">
14                {{#each this.comments}}
15                  <li class="list-group-item">{{raccourcirCommentaire this}}</li>
16                {{/each}}
17              </ul>
18            {{else}}
19              <ul class="list-group">
20                <li class="list-group-item">Soyez le premier à commenter ce post!</li>
21              </ul>
22            {{/if}}
23          </div>
24        </div>
25      {{/each}}
26    </div>
27  </div>
```

Les commentaires sont raccourcis sur notre page maintenant :



Conclusion

Dans ce tutoriel, nous avons couvert les bases de Handlebars - un moteur de template pour Node.js.. À l'aide de Handlebars, nous pouvons créer des pages Web dynamiques qui s'affichent côté serveur ou côté client. En utilisant les conditions, les boucles, les partiels et les fonctions d'assistance personnalisées de Handlebars, nos pages Web deviennent plus que du HTML statique.

