Express

Express
Partie 1

LIMITES DE NODE.JS / HTTP

- Bibliothèque http de Node.js peu pratique
  - Routes décrites dans une même méthode.
  - Envoi de données HTML par de multiples opérations.
  - Récupération des données du formulaire fastidieuse.

- Solution : Express
  - Framework pour construire des applications Web avec Node.js ou des API REST
  - Standard de fait (très utilisé)
  - Créé en 2010, Open Source licence MIT
  - Développé par Douglas Christopher Wilson
  - Choix d'être volontairement minimaliste

PROJET AVEC EXPRESS

- Objectifs :
  - Traiter des requêtes simples avec Express
  - Servir des fichiers statiques (html, css ou images)
  - Mettre en place des routes
  - Récupérer les données d'un formulaire
  - Utilisation des templates avec handlebars

PRÉPARATION DU PROJET

- Création du projet :
  - Créer un dossier `node-express` et se placer dedans
  - Exécuter `npm init -y` (pour valider toutes les options par défaut)
  - Ajouter les dépendances : body-parser, express et path

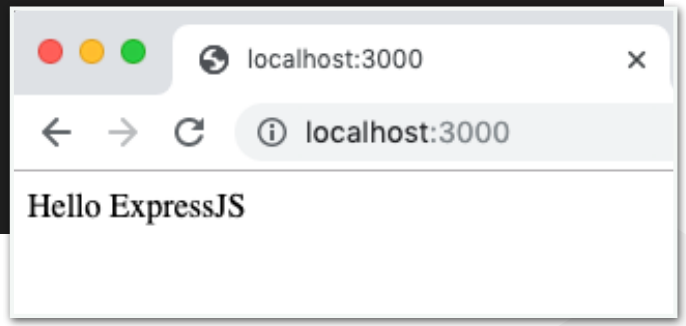
    ```
    npm install body-parser express path
    ```

CODE DE BASE

- Créer le fichier `index.js` et taper le code suivant :

index.js

```
1   const express = require('express');
2
3   const app = express();
4
5   app.get('/',(req,res)=>{
6       res.send("Hello ExpressJS");
7   });
8
9   app.listen(3000,() => {
10      console.log("Listening 3000");
11  })
```



- Exécuter en tapant : `node index.js`
- Dans la console le message "Listening 3000" doit apparaître.
- Dans Chrome, consulter `http://localhost:3000`, le message "Hello ExpressJS" doit apparaître.
- Toute demande sur une autre route "`/test`" par exemple donnera "Cannot GET /test"

CODE DE BASE : EXPLICATIONS

index.js

```
1   const express = require('express');
2
3   const app = express();
4
5   app.get('/',(req,res)=>{
6       res.send("Hello ExpressJS");
7   });
8
9   app.listen(3000,() => {
10      console.log("Listening 3000");
11  })
```

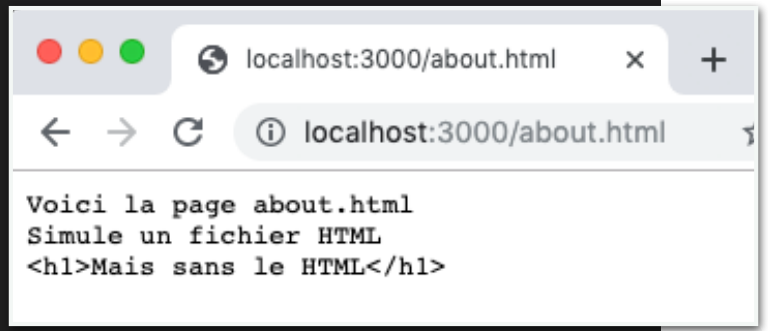
- Ligne 3 : création de l'application.
- Ligne 5 : création de la route "/"
  - `req` (request) : représente les données de la requête
  - `res` (response) : représente les données de la réponse
- Ligne 6 : envoi d'un message en réponse pour la route "/"
- Ligne 9 : mise en route du server sur le port 3000

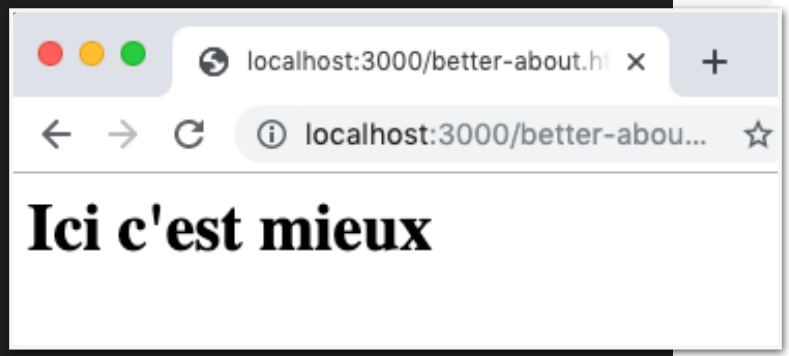
HTML

- Format par défaut : texte brut

index.js

```
1   const express = require('express');
2
3   const app = express();
4
5   app.get("/", (req,res) => {
6       res.send("Hello ExpressJS");
7   })
8
9   app.get("/about.html", (req,res)=>{
10      res.write("Voici la page about.html\n");
11      res.write("Simule un fichier HTML\n");
12      res.write("<h1>Mais sans le HTML</h1>")
13      res.end();
14  })
15
16  app.get('/better-about.html',(req,res) => {
17      res.header('Content-type','text/html');
18      res.write("<h1>Ici c'est mieux</h1>")
19      res.end();
20  })
21
22  app.listen(3000,() => {
23      console.log("Listening 3000");
24  })
```





- `.write()` peut être utilisé de multiple fois, pas `.send()`

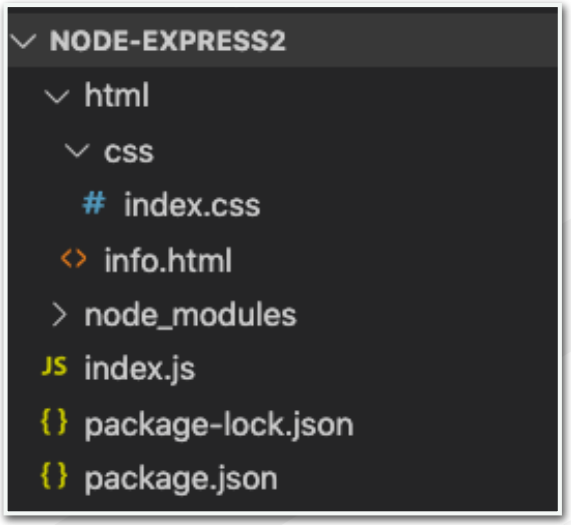
Express
Partie 2

LES FICHIERS STATIQUES

- Fichiers statiques = fichiers qui ne varient jamais (pas d'interprétation)
  - Fichier HTML
  - Fichier CSS
  - Images

- Attention : il faut inclure la bibliothèque `path`.

- Préparer la structure de fichiers suivante :



LES FICHIERS STATIQUES

- Coder le fichier `/html/info.html` et le fichier `/html/css/index.css`

/html/info.html

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <meta http-equiv="X-UA-Compatible" content="ie=edge">
7       <title>Express</title>
8       <link rel="stylesheet" href="./css/index.css">
9   </head>
10  <body>
11      <h1>Express - Info</h1>
12  </body>
13  </html>
```

/html/css/index.css

```
1   html {
2       font-family: Arial;
3   }
```

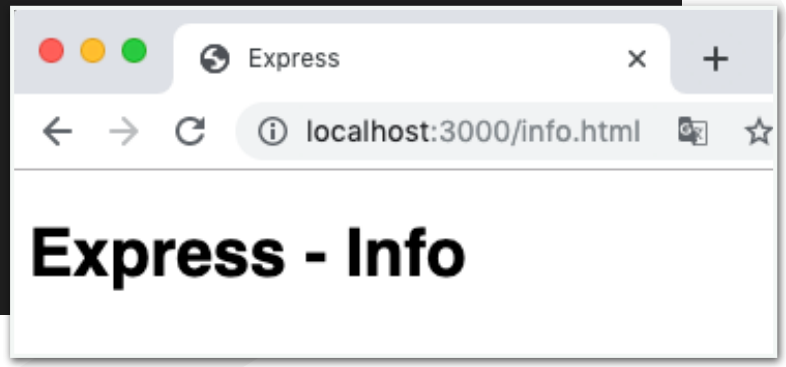
LES FICHIERS STATIQUES

- Coder le fichier `/index.js`

/index.js [extrait]

```
1  const express = require('express');
2  const path = require('path');
3
4  const app = express();
5
6  app.use(express.static(path.join(__dirname,'html')));
7
8  app.get("/", (req,res) => {
9      res.send("Hello ExpressJS");
10 })
11
12 app.get("/about.html", (req,res)=>{
13     res.write("Voici la page about.html\n");
14     res.write("Simule un fichier HTML\n");
15     res.write("<h1>Mais sans le HTML</h1>")
16     res.end();
17 })
18
19 app.get('/better-about.html',(req,res) => {
```

Désigne le dossier "html" pour la source des fichiers statiques /



- La route `/info.html` doit donner une page avec la police Arial.
- `__dirname` est le dossier courant.

Express
Partie 3

DONNÉES DE FORMULAIRE

- `body-parser` permet d'interpréter facilement les données de formulaire.
- Ajouter au début de `/index.js`

/index.js [extrait]

```
1   const express = require('express');
2   const path = require('path');
3   const bodyParser = require('body-parser');
4
5   const app = express();
6
7   app.use(bodyParser.json());
8   app.use(bodyParser.urlencoded({
9       extended:true
10  }));
11
12  app.use(express.static(path.join(__dirname,'html')));
13
14  app.get("/", (req,res) => {
15      res.send("Hello ExpressJS");
16  })
```

DONNÉES DE FORMULAIRE

- Modifier le fichier `/html/info.html`

/html/info.html [extrait]

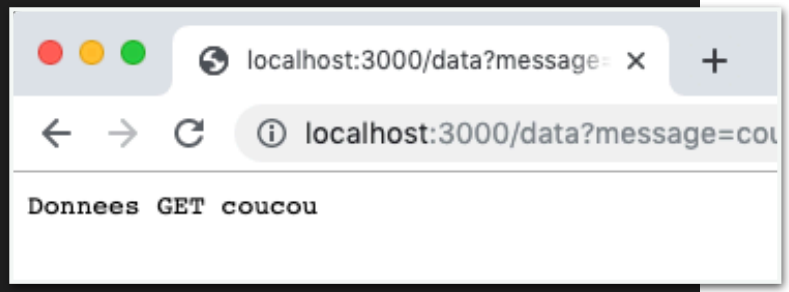
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Express</title>
    <link rel="stylesheet" href="./css/index.css">
</head>
<body>
    <h1>Express - Info</h1>
    <h2>Form GET</h2>
    <form action="/data" method="GET">
        <label for="message">Message : </label>
        <input type="text" name="message" id="message" placeholder="Entrez votre
            message">
        <button type="submit">Envoyer</button>
    </form>
    <h2>Form POST</h2>
    <form action="/data" method="POST">
        <label for="message">Message : </label>
        <input type="text" name="message" id="message" placeholder="Entrez votre
            message">
        <button type="submit">Envoyer</button>
    </form>
</body>
</html>
```

DONNÉES DE FORMULAIRE

- Ajouter les traitements /data en GET et en POST

/index.js [extrait]

```
25  app.get('/better-about.html',(req,res) => {
26      res.header('Content-type','text/html');
27      res.write("<h1>Ici c'est mieux</h1>")
28      res.end();
29  })
30
31  app.get('/data',(req,res) => {
32      res.write('Donnees GET ');
33      res.write(req.query.message);
34      res.end();
35  })
36
37  app.post('/data',(req,res) => {
38      res.write('Donnees POST ');
39      res.write(req.body.message);
40      res.end();
41  })
42
43  app.listen(3000,() => {
44      console.log("Listening 3000");
45  })
```



- Tester les deux formulaires et leurs réponses respectives

RETOURNER DU JSON

- Définir le tableau `messages` pour stocker les messages.
- Modifier les routes `app.get('/data')` et `app.post('/data')` :

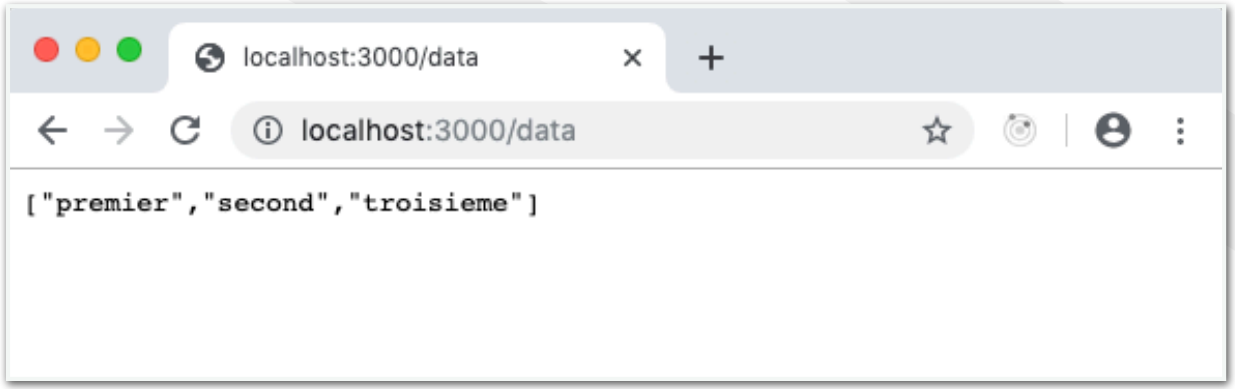
/index.js [extrait]

```
25  app.get('/better-about.html',(req,res) => {
26      res.header('Content-type','text/html');
27      res.write("<h1>Ici c'est mieux</h1>")
28      res.end();
29  })
30
31  const messages = [];
32
33  app.get('/data',(req,res) => {
34      res.json(messages);
35  })
36
37  app.post('/data',(req,res) => {
38      messages.push(req.body.message);
39      res.redirect('/info.html');
40  })
41
42  app.listen(3000,() => {
43      console.log("Listening 3000");
44  })
```

Redirige le navigateur

RETOURNER DU JSON

- Entrer "premier", "second", "troisième" dans le formulaire POST
- Constater qu'à chaque validation on revient sur le formulaire.
- Consulter `/data` en GET et obtenir les données en JSON.



PARAMÈTRE DE ROUTE

- Ne pas confondre paramètre de route et paramètre d'URL
- Ici il s'agit de gérer `/data/0`, `/data/1` ... représentée par `/data/:id`

/index.js [extrait]

```
31  const messages = [];
32
33  app.get('/data',(req,res) => {
34      res.json(messages);
35  })
36
37  app.get('/data/:id',(req,res) => {
38      if(messages[req.params.id]) {
39          res.json(messages[req.params.id]);
40      } else {
41          res.send("Indice non valide");
42      }
43  })
44
45  app.post('/data',(req,res) => {
46      messages.push(req.body.message);
47      res.redirect('/info.html');
48  })
```

- Paramètres de la route stockés dans `req.params`.
- Permet de n'afficher que le message demandé (en entrer au préalable).

Express
Partie 4

PROBLÉMATIQUE

- Comment intégrer des données dans du code HTML ?
- Solutions du pauvre :
  - Faire des concaténations dans une grosses chaîne de caractères, puis faire un `res.send()` du résultat final.
  - Éclater le code HTML en plein de `res.write()`
- Aucune de ces solutions n'est satisfaisante.

SOLUTION : MOTEUR DE TEMPLATE

- Un moteur de template est une "moulinette" qui prend en entrée :
  - Un fichier template
  - Des données

- Et qui produit un fichier HTML correctement formé.
- Moteur de template le plus connu : PHP (mais aussi Twig)

- Pour Node.js :
  - Handlebars
  - Pug
  - ...

- Intérêt : détacher l'affichage des traitements

HANDLEBARS

- Moteur de template JavaScript
  - Créé par Yehuda Katz en 2010
  - Open Source licence MIT
  - Basé sur un autre moteur de template : Mustache

- Utilisations possibles :
  - Dans un navigateur (client-side)
  - Avec Node.js (server-side)

- Node.js non nécessaire pour utiliser Handlebars

HANDLEBARS : PRÉPARATION DU PROJET

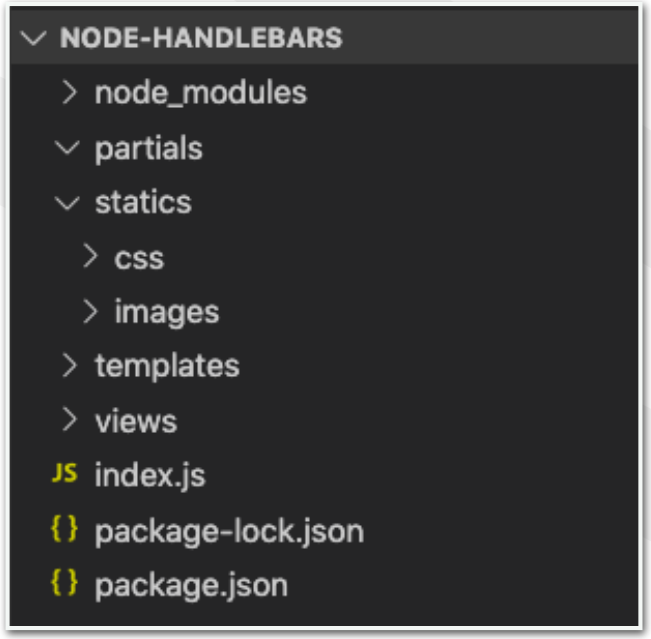
- Créer un dossier `node-handlebars`
- Se placer dans le dossier
- Exécuter `npm init -y`
- Installer les packages : `express`, `body-parser`, `express-handlebars`, `path`
- Installer le package de développement `nodemon` (`--save-dev`)
- Dans packages.json ajouter en tant que script :

  `"start": "nodemon index.js"`
- Créer le fichier `index.js`

- Note : une dépendance marquée "développement" ne sera pas conservée lors de la génération (build) pour publication

HANDLEBARS : PRÉPARATION DU PROJET

- Reproduire la structure de dossiers suivante :



HANDLEBARS : PRÉPARATION DU PROJET

- Coder le fichier `index.js` :

/index.js

```
1   const express = require('express');
2   const path = require('path');
3   const bodyParser = require('body-parser');
4   const handlebars = require('express-handlebars');
5
6   const app = express();
7
8   app.get('/', (req, res) => {
9     res.send('GET request to the homepage')
10  })
11
12  app.listen(3000,()=>{
13      console.log("listening to 3000")
14  })
```

- Exécuter `npm start`

HANDLEBARS : MISE EN ROUTE

- Configurer Handlebars dans Express

/index.js [extrait]

```
1   const express = require('express');
2   const path = require('path');
3   const bodyParser = require('body-parser');
4   const handlebars = require('express-handlebars');
5
6   const app = express();
7
8   app.engine('hbs',handlebars({
9       extname: '.hbs',
10      defaultLayout: 'default',
11      layoutsDir: __dirname+'/templates',
12      partialsDir: __dirname+'/partials'
13  }));
14  app.set('view engine','hbs');
15
16  app.get('/', (req, res) => {
17    res.send('GET request to the homepage')
18  })
```

Associe l'extension hbs avec Handlebars

Définit le moteur par défaut

HANDLEBARS : MISE EN ROUTE

- Coder le fichier `/templates/default.hbs`

/templates/defaults.hbs

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <meta http-equiv="X-UA-Compatible" content="ie=edge">
7       <title>Handlebars</title>
8   </head>
9   <body>
10      <h1>Template : default.hbs</h1>
11      {{{body}}}
12  </body>
13  </html>
```

Zone d'insertion de la view

- Fonctionnement en layout + view :
  - Layout : squelette HTML commun à plusieurs pages
  - View : partie spécifique à chacune des pages

HANDLEBARS : MISE EN ROUTE

- Coder le fichier `/views/hello.hbs`

/views/hello.hbs

```
1   <h2>Hello {{name}}</h2>
```

- Modifier le fichier `/index.js` :

/views/hello.hbs

```
16  app.get('/', (req, res) => {
17    res.send('GET request to the homepage')
18  })
19
20  app.get('/hello/:name?', (req, res) => {
21      let name = 'les amis';
22      if(req.params.name) {
23          name = req.params.name;
24      }
25      res.render('hello',{
26          name: name
27      });
28  })
29
30  app.listen(3000,()=>{
31      console.log("listening to 3000")
32  })
```

:name est facultatit (effet du ?)

Déclenchement du rendu de la vue "hello" avec les données en paramètres

- Tester avec `http://localhost:3000/hello/les i3`

Express
Partie 5

HANDLEBARS : {{ }} ET {{{ }}}

- Deux notations propres à Handlebars sont apparues dans les fichiers :
  - `{{name}}` dans `/views/hello.hbs`
  - `{{{body}}}` dans `/templates/default.hbs`

- Handlebars est conçu pour échapper toutes les données (neutraliser le code HTML qu'elles pourraient contenir).
- Principe :
  - Sécurité par défaut
  - Responsabilisation du développeur quand il a besoin d'ouvrir

- `{{ }}` : envoi des données avec échappement
- `{{{ }}}` : envoi des données sans échappement (interprétation HTML possible)

- Essayer de mettre `{{ }}` dans le template.

HANDLEBARS : CONTEXTE

- Handlebars fonctionne avec la notion de contexte
- `{{name}}` cherche dans le contexte actuel la propriété `name`
- Dans notre cas, correspond à la propriété name de l'objet transmis lors de l'appel de `render()`

```
20  app.get('/hello/:name?', (req, res) => {
21      let name = 'les amis';
22      if(req.params.name) {
23          name = req.params.name;
24      }
25      res.render('hello',{
26          name: name
27      });
28  })
```

- `this` correspond au contexte courant
- Handlebars dispose des opérateurs suivantes :
  - `.` (point) : accès à une propriété du contexte
  - `.[ ]` (point crochets) : accès à un élément d'un tableau

HANDLEBARS : {{#EACH}} ... {{/EACH}}

- Handlebars utilise des helpers
- `{{#each}} … {{/each}}` est un helper d'itération (complète) sur un ensemble (tableau)
- L'itération change le contexte à l'intérieur de la boucle.

- Créer le fichier `/data.js`

/data.js [1/2]

```
1   exports.languagesList = {
2       title: "Les langages du Web",
3       languages: [
4           {
5               name: 'HTML',
6               comments: [
7                   "Quelques balises, pas trop compliqué",
8                   "La base, indispensable"
9               ]
10          },
11          {
12              name: 'CSS',
13              comments : [
14                  "Carrément Supide, Sodide"
15              ]
16          },
```

HANDLEBARS : {{#EACH}} ... {{/EACH}}

- Suite de `/data.js`

/data.js [2/2]

```
17          {
18              name: 'PHP',
19              comments: [
20                  "De quoi faire quelques bricoles sérieuses",
21                  "Bien sauf le mélange avec le HTML",
22                  "Faut passer le premier stade"
23              ]
24          },
25          {
26              name: 'JavaScript',
27              comments: []
28          },
29      ]
30  }
```

- Intégrer `/data.js` dans `/index.js`

/index.js

```
1  const express = require('express');
2  const path = require('path');
3  const bodyParser = require('body-parser');
4  const handlebars = require('express-handlebars');
5  const data = require('./data')
6
7  const app = express();
```

HANDLEBARS : {{#EACH}} ... {{/EACH}}

- Ajouter la vue `/views/languages.hbs`

/views/languages.hbs

```
<h2>{{title}}</h2>
<ul>
{{#each languages}}
    <li>
        <p>Langage : {{name}}</p>
        <p>Commentaires : </p>
        <ul>
            {{#each comments}}
            <li>{{this}}</li>
            {{/each}}
        </ul>
    </li>
{{/each}}
</ul>

```

```
exports.languagesList = {
    title: "Les langages du Web",
    languages: [
        {
            name: 'HTML',
            comments: [
                "Quelques balises, pas trop compliqué",
                "La base, indispensable"
            ]
        },
        {
            name: 'CSS',
            comments : [
                "Carrément Supide, Sodide"
            ]
        },
```

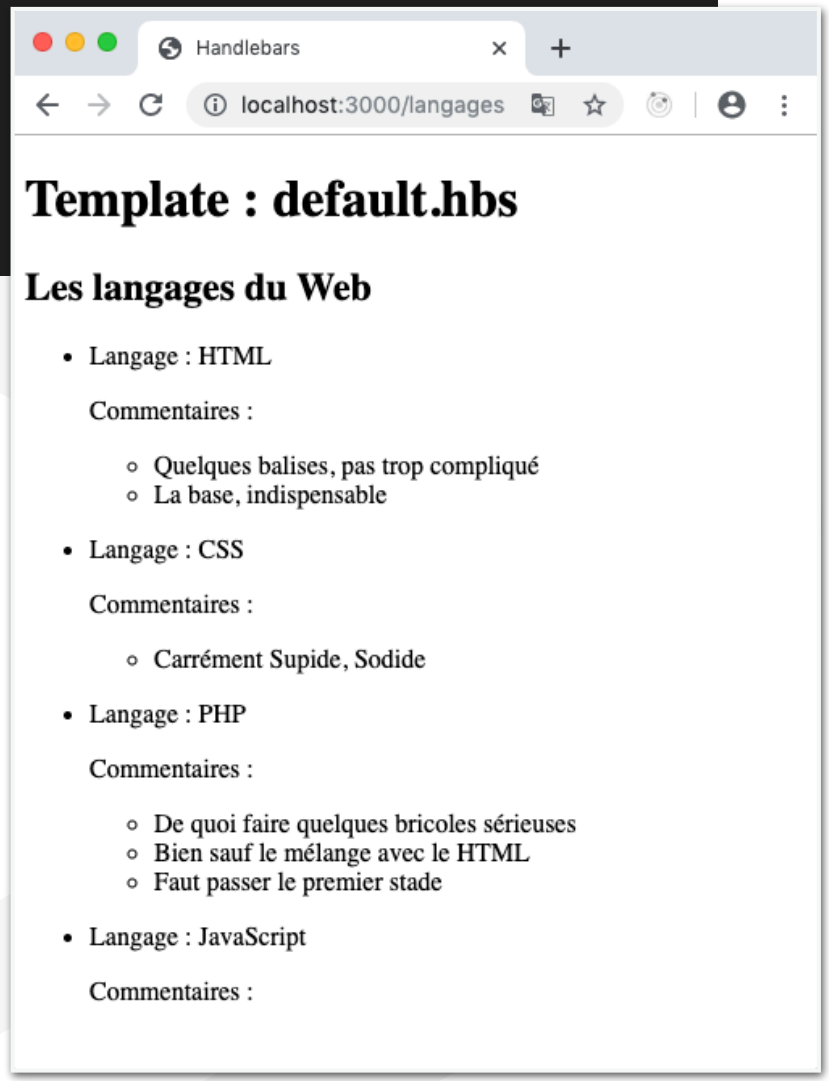
- Faire les correspondances entre les données et le template
- Observer le changement de contexte

HANDLEBARS : {{#EACH}} ... {{/EACH}}

- Ajouter la route `/langages` dans `/index.js`

/index.js [extrait]

```
31  app.get('/langages', (req, res) => {
32      res.render('languages',data.languagesList);
33  })
34
35  app.listen(3000,()=>{
36      console.log("listening to 3000")
37  })
```



Express
Partie 6

HANDLEBARS : {{#IF}} ... {{/IF}}

- `{{#if}}` ... `{{/if}}` permet de tester l'existence d'une propriété
  - N'accepte pas de condition plus précise
  - Possibilité de détecter la non existence avec `{{#unless}}`...`{{/unless}}`

- Objectif : ne pas afficher de zone commentaire vide des langages.

/views/languages.hbs

```
1  <h2>{{title}}</h2>
2  <ul>
3  {{#each languages}}
4      <li>
5          <p>Langage : {{name}}</p>
6          {{#if comments}}
7          <p>Commentaires : </p>
8          <ul>
9              {{#each comments}}
10             <li>{{this}}</li>
11             {{/each}}
12         </ul>
13         {{/if}}
14     </li>
15 {{/each}}
16 </ul>
```

HANDLEBARS : PARTIALS

- Partial = morceau de d'apparence indépendant
  - Considéré comme un sous-template
  - Peut être utilisé dans n'importe quel autre template

- Objectif : passer en partial la partie commentaire
- Créer le fichier `/partials/comments-block.hbs`

/partials/comments-block.hbs

```
1  {{#if this}}
2  <p>Commentaires : </p>
3  <ul>
4      {{#each this}}
5      <li>{{this}}</li>
6      {{/each}}
7  </ul>
8  {{/if}}
```

- Noter le changement de contexte (`comments` devenu `this`)

HANDLEBARS : PARTIALS

- Corriger `/views/languages.hbs`

/views/languages.hbs

```
1  <h2>{{title}}</h2>
2  <ul>
3  {{#each languages}}
4      <li>
5          <p>Langage : {{name}}</p>
6          {{>comments-block comments}}
7      </li>
8  {{/each}}
9  </ul>
```

- Noter le passage de `comments` en paramètre qui devient `this` dans le partial.
- Rien ne change pour `/index.js`

HANDLEBARS : AUTRE LAYOUT

- Utilisation de `/templates/default.hbs` par défaut
- Objectif : utiliser un autre layout
- Créer le fichier `/templates/centered.hbs`

/templates/centered.hbs

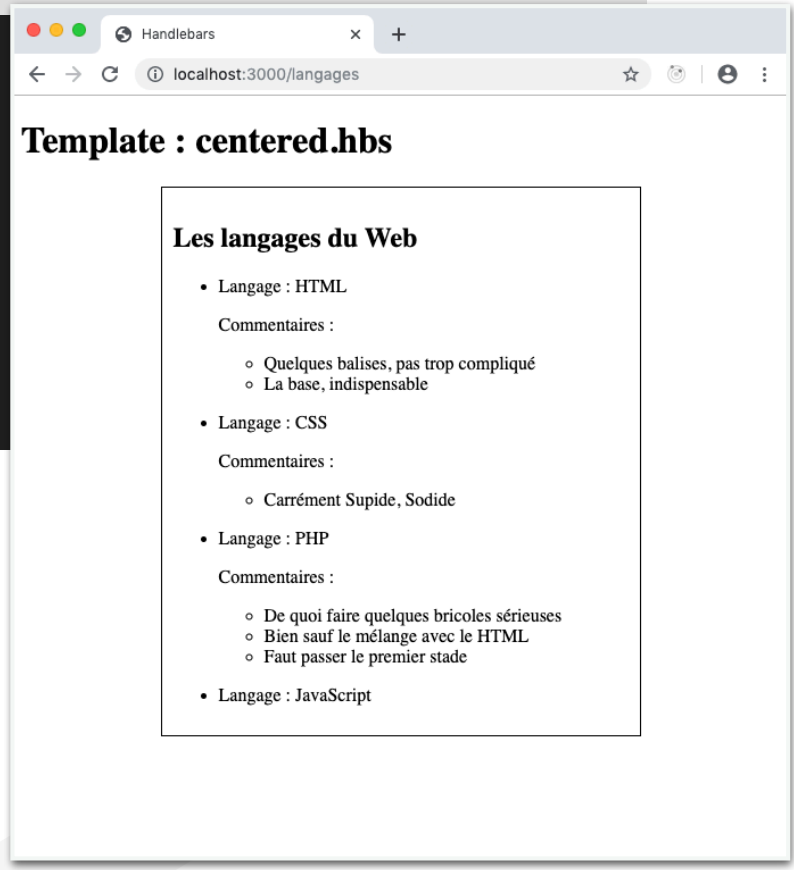
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Handlebars</title>
    <style type="text/css">
    .centered {
        width:400px;
        margin-left:auto;
        margin-right: auto;
        border:1px solid black;
        padding:10px;
    }
    </style>
</head>
<body>
    <h1>Template : centered.hbs</h1>
    <div class="centered">
        {{{body}}}
    </div>
</body>
</html>
```

HANDLEBARS : AUTRE LAYOUT

- Corriger la route `/langages` dans `/index.js`

/index.js

```
31  app.get('/langages', (req, res) => {
32      res.render('languages',{
33          ...data.languagesList,
34          layout:'centered'
35      });
36  })
37
38  app.listen(3000,()=>{
39      console.log("listening to 3000")
40  })
```



- Noter l'utilisation de la déstructuration (ligne 33) pour ajouter la propriété `layout`.

Express
Partie 7

HANDLEBARS : CUSTOM HELPER

- Possibilité de faire des custom helpers : fonctions JavaScript transmises à Handlebars
- Objectif : indiquer le nombre de commentaires après chaque nom de langage

- Modifier le fichier `/index.js`

/index.js [extrait]

```
31  app.get('/langages', (req, res) => {
32      res.render('languages',{
33          ...data.languagesList,
34          layout:'centered',
35          helpers: {
36              commentsCount:(language) => {
37                  if(language.comments){
38                      return `<small>(${language.comments.length})</small>`;
39                  }
40              }
41          }
42      });
43  })
```

- Création d'un helper `commentsCount`

HANDLEBARS : CUSTOM HELPER

- Modifier `/views/languages.hbs`

/views/languages.hbs

```
1  <h2>{{title}}</h2>
2  <ul>
3  {{#each languages}}
4      <li>
5          <p>Langage : {{name}} {{#commentsCount this}}{{/commentsCount}}</p>
6          {{>comments-block comments}}
7      </li>
8  {{/each}}
9  </ul>
```