Travaux pratiques

TP1 : installer bootstrap

- Créez un nouveau répertoire test-npm-bootstrap et déplacez-vous dans celui-ci.
- Initiez npm - vous pouvez laisser les valeurs par défaut.
- Ouvrez votre fichier “package.json” et observez son contenu.
- Installez la librairie bootstrap - vous pouvez aller voir le site “https://getbootstrap.com/” si vous êtes coincés.
- Ouvrez à nouveau votre fichier “package.json”, qu’est ce qui a changé ?

A cette étape, votre librairie bootstrap est maintenant installée dans votre projet. Il vous reste à l’utilisez.

- Créez un nouveau fichier index.html et mettez-y la structure de base d’une page html.
- Ajoutez une balise link pour charger dans votre page le fichier “bootstrap.css”. Ce fichier est quelque part dans le répertoire “bootstrap” de votre node_modules, à vous de trouver où.
- Pour vérifier que tout fonctionne bien, ajoutez le code suivant dans votre page html :

```
<h3 align="center">Bootstrap</h3>
```

Si votre titre n’est pas centré, c’est que vous n’avez pas chargé correctement bootstrap !

TP 2 : créer un serveur web avec express

Mon premier serveur web

L’objectif de ce tp est de créer votre propre serveur web avec node.js.

Pour rappel, node.js est un environnement d’exécution capable de faire tourner du Javascript - plutôt que de charger du Javascript dans un navigateur comme nous avons déjà l’habitude de le faire.

- Créez un nouveau répertoire mon-premier-serveur-web et déplacez-vous dans celui-ci.
- Initiez npm - vous pouvez laisser les valeurs par défaut.
- Installez la librairie “express” avec la commande suivante :

```
npm install express
```

Cette librairie sert à créer facilement un serveur web.

- Créez un fichier “app.js”

Dans ce fichier, la première chose à faire est de charger votre librairie express. Pour cela, ajoutez sur la première ligne :

```
const express = require('express');
```

Sur node.js, la fonction ‘require’ permet de charger une librairie installée. Si la chaîne de caractères fournie est un chemin absolue, alors ‘require’ va automatiquement chercher la librairie demandée dans le répertoire ‘node_modules’.

La deuxième étape est de créer notre serveur web :

```
const app = express()
```

Ensuite, nous allons configurer sur quel port notre serveur écoute :

```
const port = 3000

app.listen(port, () => {
  console.log(`My first web server listening on port ${port}`)
})
```

Enfin, pour configurer un endpoint et une réponse :

```
app.get('/', (req, res) => {
  res.send('Hello World!')
})
```

Avec le code précédent, nous demandons à express de configurer le endpoint racine “/” (autrement dit localhost:3000/ - localhost:3000 fonctionne aussi) et nous y associons une fonction. A chaque fois que le endpoint sera interrogé (une requête http avec le verbe GET sur localhost:3000), la fonction sera déclenchée. Celle-ci reçoit en paramètre deux choses : ‘req’ un objet représentant la requête http, ‘res’ un objet représentant la réponse à envoyer.

Enfi, res.send nous permet de répondre à la requête reçue avec une chaîne de caractères.

- Mettez à jour votre fichier package.json : définissez un script “start” qui lance la commande “node app.js”.
- Lancez votre script start

- Ouvrez un navigateur, rendez-vous sur l'url localhost:3000, que constatez vous ?

Félicitations, vous venez de créer un votre premier serveur web node.js !

TP 3 : Renvoyez du json

Jusqu'à maintenant, vous n'avez qu'un seul endpoint et celui-ci renvoie une chaîne de caractères. Dans la réalité la majorité des serveurs nous renvoie des données au format json.

Avec express, vous pouvez utiliser la fonction 'res.json' pour renvoyer du json.

- Ajoutez un nouvel endpoint :
  - verbe: GET
  - path : '/json'
  - renvoie l'objet { firstName: 'Riri', lastName: 'Duck' } au format json.

Pour vérifier si vous avez réussi, ouvrez un navigateur et aller sur l'url 'localhost:3000/json'.

N'oubliez pas de redémarrer votre serveur web ! Par défaut, node ne relance pas votre serveur à chaque modification enregistrée.

TP 4 : communication front-back

Votre serveur fonctionne, mais pour l'instant vous l'interrogez manuellement en renseignant des urls dans votre navigateur. Dans la réalité, ce sont d'autres applications qui interrogent nos serveurs pour récupérer des données.

- Créez un nouveau répertoire ‘mon-front’ et déplacez-vous dans celui-ci.
- Créez un fichier index.html avec la structure de base d’une page html.
- Créez un fichier app.js et reliez-le à votre page html
- Utilisez votre extension go-live pour ouvrir votre page.
- Vérifiez que votre fichier js est bien relié avec un console.log
- Déclenchez une requête http auprès de localhost:3000/json, quel problème rencontrez-vous ?

CORS signifie Cross Origin Resource Sharing.

C’est un mécanisme qui permet de protéger l’accès aux endpoints de nos serveurs. Par défaut, seul une requête provenant de la même origine peut interroger un endpoint. Cela signifie concrètement que si vous souhaitez interroger localhost:3000/json, alors votre requête doit avoir comme origine localhost:3000.

Quand vous utilisez go-live, vous lancez un serveur web. Celui-ci se lance par défaut sur localhost:5500. Votre requête http part de votre page affichée grâce à ce serveur. Ce qui signifie l’origine de votre requête est “localhost:5500”.

localhost:5500 n’est pas égal à localhost:3000. Donc votre requête est bloquée par votre serveur node (celui qui tourne sur localhost:3000).

- Retournez dans le projet de votre serveur node
- Installez la dépendance ‘cors’
- Allez voir la documentation de la librairie pour savoir comment l’utilisez avec express.

Si vous avez utilisé correctement la librairie cors, alors maintenant votre requête vers localhost:3000/json fonctionne.

Dans votre projet front :

- Convertissez les données reçues en objet javascript et affichez les dans votre console.
- Dans votre fichier html, ajoutez une div avec l’id “data-container”
- En utilisant du js, placez les données que vous recevez de votre requête au sein de votre div.

Félicitations, vous êtes maintenant capable de récupérez des données depuis un serveur et de les afficher à l’écran - quelque chose fait tous les jours dans de véritables applications web !