


TP n° 7


Express, routes et moteur de *template*

préambule : Dans ce TP, nous allons nous familiariser avec le framework *Express*, qui se définit lui-même comme une infrastructure Web pour Node.js. Comme pour le TP n° 6, tous les exercices de cette séance peuvent être réalisés sur le serveur `licinfo2.univ-jfc.fr` ou bien sur votre machine personnelle, et, quel que soit le cas, de préférence dans le conteneur configuré dans le TP n° 6.

Exercice 1 – Site web avec Express

La première fonctionnalité d'Express est de servir des fichiers statiques. Dans cet exercice, on configure une route permettant de servir un répertoire contenant un site web statique.

 Après avoir initialisé un nouveau projet Node.js avec *npm*, installer *Express* avec la commande `npm install express`.

 Créer un nouveau fichier `server.js` et y copier/coller le code ci-dessous. Il s'agit d'un code *boilerplate* pour créer un serveur Web intégrant le *middleware* Express.

```
1 const express = require('express')
2 const http = require('http')
3
4 var app = express();
5 var server = http.createServer(app);
6
7 server.listen(3000, function() {
8   console.log("Serveur démarré");
9 });
```

 Créer un répertoire `website` à la racine de votre projet. Copier/coller les fichiers de l'archive `grumpy.zip` téléchargeable sur Moodle.

 Ajouter une route statique en copiant la ligne suivante

```
6 app.use('/', express.static('website'));
```

L'instruction `app.use` permet d'ajouter une route, en l'occurrence ici sur l'URL `/` (racine). `express.static` permet d'indiquer à Express qu'il doit servir tous les fichiers de ce répertoire de manière statique.

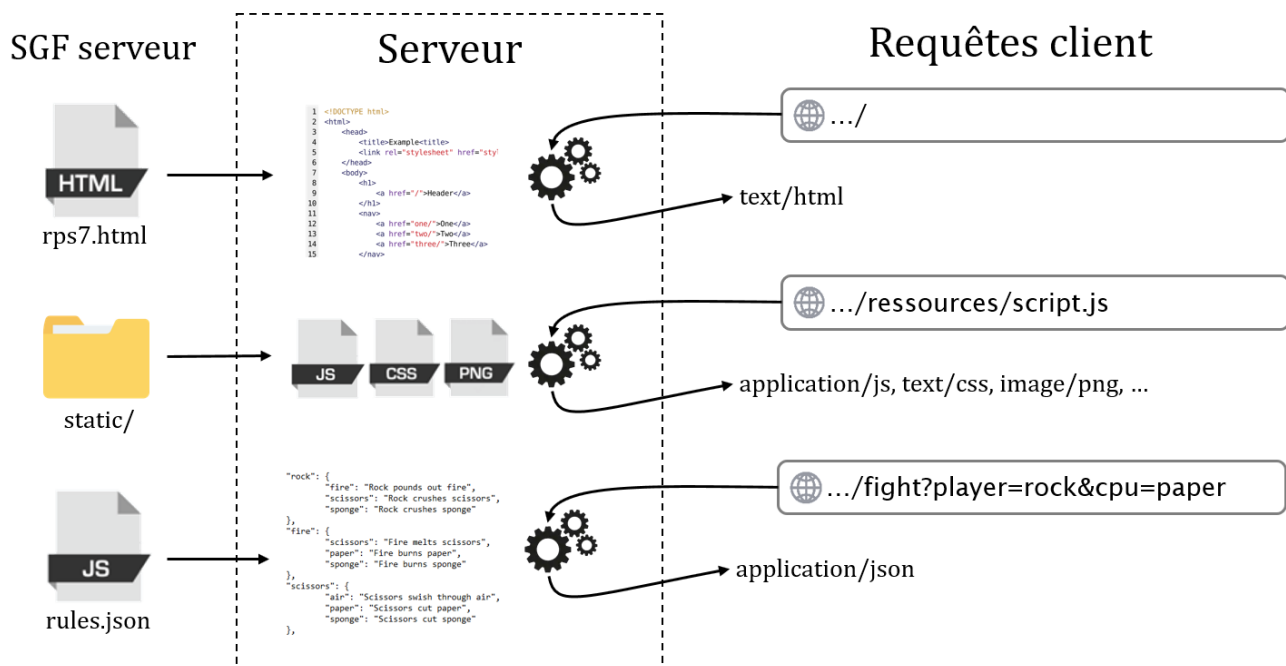
✎ Monter le conteneur docker ou bien lancer le serveur manuellement avec la commande `npm start`. Vérifier que le site s'affiche et fonctionne bien.

✎ Modifier le script `like.js` pour incrémenter le compteur de respects à chaque nouveau clic sur l'image.

Exercice 2 – Routage avancé

Dans ce nouvel exercice, nous explorons des fonctionnalités de routage plus avancées de Node.js. Nous mettons en œuvre une version améliorée¹ du jeu de Pierre-Feuille-Ciseaux où un joueur joue contre l'ordinateur.

L'architecture générale du serveur est donnée dans la figure suivante :




On y voit 3 routes au fonctionnement différent, chacune liée à des requêtes différentes. Nous allons les mettre en œuvre dans les questions suivantes. Tous les fichiers nécessaires à la résolution de l'exercice sont disponibles sur Moodle.

✎ À partir d'un nouveau serveur Express, ajouter une route prenant en charge les requêtes de type GET sur l'URL `http://serveur/`. Le serveur répond en envoyant le contenu du fichier `rps7.html` situé à la racine du serveur. On utilisera les instructions `app.get` pour créer une nouvelle route, et `res.sendFile` pour renvoyer le contenu d'un fichier.

✎ Ajouter une deuxième route prenant en charge les requêtes du client sur l'URL `http://serveur/ressources/*`. Le serveur sert de manière statique tout fichier demandé par le client et présent dans le répertoire `static` à la racine du serveur.

1. Plus d'infos : <https://www.umop.com/rps7.htm>


 Enfin, ajouter une troisième route permettant de traiter les requêtes de type GET sur l'URL `http://serveur/fight`. Le serveur répond en JSON en fonction des paramètres passés dans la requête. Le script à écrire se déroule de cette manière :

- À la réception d'une requête, le serveur récupère les valeurs des 2 paramètres `player` et `cpu`. 3 cas sont possibles :
 - Si les valeurs de `player` et `cpu` sont égales, alors il y a égalité.
 - Si `rules[player][cpu]` est défini, alors `player` a gagné.
 - Si `rules[player][cpu]` est non défini, alors `cpu` a gagné (et logiquement `rules[cpu][player]` est défini).
- Après avoir déterminé le gagnant, s'il y en a un, le serveur retourne un objet JSON de la forme `{outcome: o, message: m}`, où :
 - `o` est égale à 0 en cas d'égalité, 1 de victoire de `player` et -1 sinon.
 - `m` est la chaîne `rules[player][cpu]` si `player` a gagné, ou `rules[cpu][player]` si `cpu` a gagné. En cas d'égalité, il n'y a pas de message.

La structure de données `rules` doit être chargée à partir du fichier `rules.json` situé à la racine du serveur. On pourra utiliser l'instruction `fs.readFileSync` du module `fs` pour le chargement.


Exercice 3 – PUG


Un serveur Web moderne n'écrit pas de code HTML mais génère des documents à la volée à partir de gabarits. Le principe d'un moteur de *template* est de combiner un modèle de données à des *templates* (i.e. des gabarits) pour produire des documents. Dans cet exercice, nous concevons un site Web simple à partir de données structurées et de gabarits. Les ressources nécessaires pour cet exercice sont disponibles sur Moodle.

 Dans un nouveau serveur Express, indiquer que les pages seront rendues par le moteur intégré PUG, en insérant les lignes 5 et 6 ci-dessous après déclaration de l'app Express :

```
4 var app = express();
5 app.set('view engine', 'pug');
6 app.set('views', 'views/');
```

La ligne 6 indique les gabarits seront cherchés dans le répertoire `views` à la racine du serveur.

 Télécharger le gabarit `index.pug` à placer dans le répertoire des gabarits.

 Dans le serveur, ajouter une nouvelle route `http://serveur/userin` en copiant le code suivant :

```
1 app.get('/userin', (req, res) => {
2   var locals = {
3     username: req.query.username,
4     firstTime: true
5   };
6   res.render("index", locals);
7 });
```

Lancer le serveur et tester l'URL `http://serveur/userin?username=David%20Panzoli`

✎ Modifier le code pour qu'un utilisateur soit reconnu à partir de son deuxième accès à la page.

Exercice 4 – Dinopedia

Dans cet exercice, on souhaite fabriquer un site qui retourne des fiches descriptives des dinosaures décrits dans le fichier `dinos.json`. Le résultat est illustré ci-dessous pour la fiche du *Diplodocus*, dont le code HTML est accessible dans le fichier `diplodocus.html` à télécharger sur Moodle.



Diplodocus

Quadrupède herbivore à long cou

On pense que le dinosaure *Diplodocus* avait un long cou utilisé pour atteindre à la fois la végétation basse et haute. Les scientifiques débattent encore de la façon dont le reptile a réussi à tenir un cou aussi long. Les paléontologues pensent maintenant que les ligaments qui allaient de l'arrière du cou à la hanche auraient permis au reptile de tenir son cou suffisamment bien sans utiliser de muscles. La colonne vertébrale est fendue en son milieu, et cet espace était utilisé pour maintenir des ligaments comme celui-ci. Les scientifiques pensent également que le reptile avait des épines étroites et pointues sur le dos.

période	de 155 à 145 millions d'années
poids	20 tonnes
longueur	26 m

✎ Récupérer le fichier `dinos.json` et l'archive `dinos.zip` contenant les images des dinosaures.

✎ Mettre en place un serveur Express capable de répondre à des requêtes sur l'URL `http://serveur/specimen=x` (où `x` est un nom de dinosaure) en retournant la fiche correspondante, ou une erreur 404 si le dinosaure n'existe pas.

On s'inspirera de la page `diplodocus.html` pour fabriquer² le gabarit de la fiche qu'on nommera `fiche.pug`.

✎ Mettre en place un site permettant de naviguer dans les différentes fiches à partir d'un menu. On pourra fabriquer un gabarit spécifique pour le menu et utiliser l'instruction `include` de PUG pour lier la fiche au menu.

2. Documentation officielle de PUG : <https://pugjs.org/api/reference.html>