



## Review

## Slides Vidéo

### Express & EJS

#### Architecture web client/serveur

Le client sera matérialisé par la partie front end, c'est à dire par le navigateur web :

- IE, Chrome, Firefox, Safari...

Le serveur, lui, constitue la partie back end.

#### Interactions clients/serveur

1. Le client (navigateur) par l'intermédiaire d'une URL ou d'un lien pose une QUESTION au serveur. On appelle cela une REQUÊTE.
2. Le serveur, pour sa part, reçoit la question posée. Par exemple, « puis-je commander un voyage en TGV, aller-retour, Lyon-Paris, le 1er Janvier 2019, pour 1 personne, en 1ère classe ? ». Un traitement est prévu par le serveur pour recevoir une question de ce type, et c'est pourquoi il va pouvoir traiter la demande. Il va ainsi vérifier la disponibilité des places dans le train concerné. Et seulement lorsqu'il aura procédé au traitement, il enverra une REPONSE au client.
3. Le client pourra désormais afficher la page contenant les éléments de réponse appropriés.

Ces interactions rendent des pages web statiques ainsi dynamiques. Elles sont uniques pour chaque client (en fonction des questions et des réponses).

Dans le web dynamique, c'est le serveur qui se charge de modifier un modèle de page.

#### Construire un serveur web Express

Grâce à NodeJS, nous allons pouvoir

## Création du backend

⚠ La commande initialise un nouveau répertoire

```
express --view=ejs --git myapp
```

Gestion de fichier HTML

Nom du projet

Paramétrage de git



Grâce à Node.js, nous avons pu utiliser un framework très utile qui se nomme Express.  
Pour installer un générateur de serveur Express sur votre ordinateur, il suffit d'exécuter dans le terminal la commande suivante :

```
npm install express-generator  
-g
```

Cette commande n'est plus utile dès lors qu'Express Generator est déjà installé sur votre machine.

Ensuite, pour effectivement générer un serveur Express et créer concrètement votre backend, il suffit de vous positionner où vous souhaitez sur votre machine avec votre terminal, et d'exécuter la commande suivante :

```
express --view=ejs --git  
myapp
```

Cette commande initialise un nouveau répertoire myapp (vous pouvez le changer par le nom de votre projet lors de l'exécution de l'instruction).

- Se positionner dans le répertoire du backend

```
cd myapp
```

- Installer les modules

```
npm install
```

- Lancer le serveur

```
npm start
```

- Afficher une page  
<http://localhost:3000>  
Ou  
<http://127.0.0.1:3000>

Express, comment ça marche ?

---

Un framework tel qu'Express, c'est tout simplement la désignation de fonctionnalités clef en main, ainsi qu'une organisation de fichiers particulière.

Les routes .js = les questions

```
├─ app.js
├─ bin
│   └─ www
├─ package.json
├─ public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └─ style.css
├─ routes
│   ├── index.js
│   └─ users.js
└─ views
    ├── error.ejs
    └─ index.ejs
```

- Exemple avec le fichier routes/index.js

```
var express =
require('express');
var router =
express.Router();

/* GET home page. */
router.get('/', function(req,
res, next) {
  res.render('index', { title:
'Express' });
});

module.exports = router;
```

Pour faire simple, les routes correspondent aux traitements prévus des QUESTIONS ou REQUÊTES, envoyées par le client, au serveur.

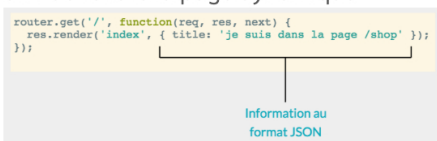


Le traitement à exécuter est une mécanique asynchrone. Pour rappel, il s'agit d'une fonction de callback. Elle ne se déclenchera que si la question /home (la requête par URL <http://localhost:3000/home>) est posée dans ce cas concret.

Si la question posée est / (la requête par URL <http://localhost:3000/>), le serveur va répondre par l'affichage de la page index, c'est à dire la lecture du fichier index.ejs et l'envoi du résultat au navigateur.



Lors de l'envoi de cette réponse, il déversera des informations précises. Dans le même exemple ci-dessous, il déversera, au format JSON, un titre « je ne suis pas dans la page /shop ». La clef title sera accessible dans le fichier index.ejs afin de rendre la page dynamique.



Mais au fait, c'est quoi .ejs ?

Les views .ejs = les réponses (sous forme de templates)

```

├─ app.js
├─ bin
│   └─ www
├─ package.json
├─ public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └─ style.css
├─ routes
│   ├── index.js
│   └─ users.js
└─ views
    ├── error.ejs
    └─ index.ejs
    
```

EJS est un moteur de template pour page web. Il sert à rendre les pages HTML dynamiques. Contrairement au HTML statique, EJS permet de prévoir en amont des parties dans la page qui seront dynamiques et qui dépendent de la réponse renvoyée par le serveur au client. Pour faire simple, le serveur va déverser des informations dans le HTML.

Ci-dessous, dans le fichier index.ejs, on a inséré la variable title, prévue dans le traitement juste avant.

```

index.ejs
<html>
  <head>
  </head>
  <body>
    <h1><%= title %></h1>
  </body>
</html>
    
```

Nom de la clef  
choisi dans le JSON

Pour délimiter le HTML et le JS :

HTML `<% JS %>` HTML

Pour afficher la valeur d'une variable

`<%= variabeJS %>`

## Structures complexes

Ici, la route index.js avait prévue d'envoyer un title, lui-même exploité dynamiquement dans le fichier index.ejs

Nous pourrions imaginer des structures de données plus complexes.

Par exemple, dans le fichier /route/index.js avec l'envoi d'un tableau.

```

router.get('/', function(req, res, next) {
  res.render('index', { products: ['chaussures', 'casquettes', 'pantalons'] });
});
    
```

Tableau

Des lors, la clef products pourrait être accessible dans le fichier /views/index.ejs

```

index.ejs
<html>
    
```

```

<head>
</head>
<body>
  <ul>
    <%
      for(var i=0; i<products.length; i++) {
    %>
      <li><%= products[i] %></li>
    <%
      }
    %>
  </ul>
</body>
</html>

```

Remarquons que par ce qu'il y a plusieurs products (c'est un tableau...), il est nécessaire de générer une boucle dans le fichier view/index.ejs afin d'afficher correctement la totalité des éléments.

Pour aller plus loin, on pourrait également prévoir une condition, selon laquelle on voudrait afficher, s'il n'y avait aucun produit, « Aucun article disponible ».

```

index.ejs
<html>
<head>
</head>
<body>
  <%
    if(products.length == 0) {
  %>
    <h2>Aucun article disponible</h2>
  <%
    }
  %>
</body>
</html>

```


## Ressources externes

Les images, fichiers Javascript externes, ou feuilles de style doivent être tout simplement déposés dans le répertoire public au sein des dossiers correspondants.

```

├─ app.js
├─ bin
│   └─ www
├─ package.json
├─ public
│   ├── images
│   ├── javascripts
│   └── style.css
├─ routes
│   ├── index.js
│   └── users.js
└─ views
    ├── error.ejs
    └── index.ejs

```



Next >>