Prog Web

# Cours 1 : HTML

Tokenisation ?

# Cours 2 : CSS

Poser des propriétés de style sur HTML avec CSS

* Couleur
* Taille
* Positionnement

Lier un fichier CSS à un fichier HTML :  
<link src="style.css" rel="stylesheet" type="text/css"/>

Une règle CSS => un ensemble de propriétés CSS

Un sélecteur (\*) est une requête permettant d’identifier 0, 1, des nœuds du DOM

Le navigateur a déjà construit le DOM

* Aller chercher la/les feuille(s) CSS
* Pour chaque règle, exécuter la requête du sélecteur
* Attacher les règles aux nœuds « matchés »

3 façons de base pour requêter un nœud :

* Les balises
* Les id
* Les classes

En termes de priorités pour le CSS lorsqu’il y a un conflit :  
balise < class < id

# Cours 3 : Le back (HTTP)

Le back fournit les ressources demandées par le front

1. Qu’est-ce qu’une demande de ressource ?
   * Il y a au moins l’URL

HTTP

* + Il y a le VERB associé à cette ressource
    - GET
    - POST
    - DELETE
    - PUT
    - PATCH
  + etc…

<a href=”images.html”> => <a href=”images.html?client=1”> pour identifier le client initial (= URL rewriting)

Le back

* attend les requêtes HTTP

Si je reçois une requête  
Alors je revois le fichier qui correspond à l’URL

Express JS (if, construction de la ressource, l’envoi de la réponse)

Initialisation : const app = express(), // notre SW est initialisé, non démarré

If : app.get(‘/index.html’, (req, res) => {/Todo});

Si VERB = GET

Si path \*URL == index.html

Construction dynamique  
/Todo let ressource = "<HTML> …" ;  
 ressource += "<ul><li> … "} ;

J’envoie { ] res.send(ressource) ;

# Cours 4 : Serveur

HTTP

* Req
  + Verb (méthode)
  + URL
    - Origine (https://google.fr)
    - Path name (index.html ou images/image1.png)
    - Search params (origine/path/… ?prop\_name=value&p=v)
* Res
  + Status (Code)
    - 200 => OK
    - 404 => Not Found
    - 304 => Cache
  + Content (en HTML)

Serveur Web => répondre aux requêtes (envoyer des res)

Si (req == ???) alors res = …

ExpressJS

Pour installer ExpressJS :

const express = require(‘express’) ;

Initialiser Express

const app = express() ;

Ecouter sur le port 8080

app.listen(8080, () => {console.log(‘OK’) ;}) ;

(Règle de base) => Req alors Cannot …

Ajouter une règle en ExpressJS :

app.VERB(‘pathLike’, (req, res) => {//ajouter des conditions -> construire/lire la ressource -> envoyer la ressource};

Exemple :

app.get(‘/’, (req, res) => {  
 res.send(‘OK’);  
});

possibilité de renvoyer un fichier avec res.sendFile(absolute path)

app.get(‘\*’ … signifie que quelque soit la requête le res sera effectué

req.path pour renvoyer le chemin du fichier localement (?)

const fs = require(‘fs’)

# Cours 5 : Génération par template (Back)

Pour construire du HTML dynamique (= qui change en fonction de différents paramètres)

// concaténation de string  
let html = ‘<html><body>’  
html += …  
res.send(html)

Mais mix HTML et JS (= langage serv) => difficile à lire pour le développeur

Template(Exemple) => du HTML avec des zones dynamiques  
<html>  
 <body>  
 <h1> Bonjour NAME </h1>  
   
 Dynamic  
 </body>  
</html>

Mettre les str avant d’envoyer les ressources

Délimitation des zones

en ejs = <% zone %>  
en php = < ? zone ?>

Quelles données a template en ejs le serveur demande la réalisation du template (vers HTML)

res.render(nomTemplate, objetDictionnaire) ;

//Template compteur

***serveur.js***let i = 0;  
app.get(‘/compteur’, (req, res) => {  
 i++ ;  
 res.render(‘compteur’, {nb\_req : i});  
});

***compteur.ejs***<html>  
 <body>  
 Cette page a été vue  
 <%=nb\_req%>  
 </body>  
</html>

***images.ejs***  
<html>  
 <body>  
 <div>  
 <% for (let i = 0 ; i < images.length ; i++) { %>  
 <image src = ‘<% =images[i] %>’>  
 <% } %>

***serveur.js***  
app.get(‘/images’, (req, res) => {  
 ??????????????????????????  
 res.render(‘images’ {images : valeur\_image});

});

Convention :

Si path\_name == /static/\*  
Alors retourner le fichier, s’il se trouve dans le répertoire ‘/public’

* app.use(‘/static’, express.static(‘public’);

# Cours 8 : Bases de données et l’algèbre relationnelle (requêtes)

Modèle  
Conceptuel  
Données

Groupe Etudiant  
Table Table  
Clé Clé  
 Clé étran. (Groupe)

|  |  |
| --- | --- |
| Groupe | |
| ID | Name |
| 1 | Groupe 1 |
| 2 | Groupe 2 |

|  |  |  |
| --- | --- | --- |
| Etudiant | | |
| ID | Name | G\_ID |
| 1 | Toto | 1 |
| 2 | Tutu | 1 |
| 3 | Tata | 2 |

Etudiants du groupe 1  
Déclaratif (semaine dernière) ~~Comment~~  
Quel n-uplet solution ? (1, Toto) => ID-Etu, Nom-Etu  
Quelles contraintes ? ID\_Groupe = 1 AND ID\_Groupe = G\_ID  
Quel domaine de recherche ? Etudiant, Groupe (= Produit scalaire)

Groupe X Etudiant  
1 Groupe1 1 Toto 1  
1 Groupe1 2 Tutu 1  
1 Groupe1 3 Tata 2 ←----- ?????????

Algèbre des opérations sur un domaine

Algèbre relationnelle  
6 opérations de base (+1)  
1 domaine = les relations

Les 6 opérations de base

* La sélection = Tcontrainte (R) → R’ (R’ contient les n-uplets de R qui satisfONT contrainte) (SIGMA PAS T)
* La projection = TTcol (R) → R’ (R’ ne contient que les colonnes col de R) (PI PAS T)
* Le produit scalaire = R1 X R2 → R3 (R1 et R2 doivent avoir des schéma différents)
* Le renommage de colonne = Pcol → col’ (R) → R’ (R’ a un autre nom col’ pour la colonne col)
* L’union = R1 U R2 → R3 (R3 contient les lignes de R1 et R2)
* La différence = R1 – R2 → R3

Exercice :

* Les noms des étudiants de 2021  
  TTName (Tannée=2021 (Etudiant))
* Les étudiants du groupe 1  
  TTName (PGD=G\_ID (PGName=’Groupe1’ (Etudiant X PID->GD AND Name->GName (Groupe):)))
* TG\_ID=ID (Groupe X PID->EID AND Name->EName (Etudiant)) => Jointure
* (Sucre Syntaxique)  
  La jointure = A JOINa=b B T (A X P (B)) ( x → genre de sablier sur le coté)

Pourquoi deux façons de faire ?

Déclaratif VS Algèbre  
→ c’est équivalent en expressivité  
→ en algèbre on sait optimiser