

Bachelier en Informatique de Gestion

Web : principes de base Projet de Développement Web

Enseignement supérieur économique de type court

Code FWB : 7534 29 U32 D1, 7534 30 U32 D3

Code ISFCE : 4IWPB, 4IPW3



Table des matières

Généralités

- 01. Introduction au web
- 03. Outils
- 05. Format XML
- 06. Format JSON

Front-End

- 12. Structure HTML
- 13. Formulaire HTML
- 14. Mise en forme CSS
- 15. Adaptabilité
- 17. Javascript
- 18. Bibliothèque jQuery
- 19. Composant Vue.js

Back-End

- 21. Middleware PHP
- 22. Traitement du formulaire
- 23. Architecture MVC
- 24. Données SQL
- 25. Données NoSQL
- 27. Requête asynchrone

27. Requête Asynchrone & API

Asynchronous JavaScript and XML (AJAX)

Vanilla JavaScript : Fetch



= BUROTIX ()

Principe de la requête asynchrone

Principe

- En un mot : rafraichir dynamiquement une partie de la page web avec une info venant du serveur web
 - En route vers la [Single Page Application](#)
- Côté serveur :
 - Requête http(s) classique, rien à signaler
- Côté client :
 - Une fonction JS envoie une requête au serveur.
 - Le serveur retourne des données.
 - HTML, CSS, XML, JSON, ...
 - Une fonction JS traite ces données retournées et les affiche à un endroit précis de la page.
- Remarques :
 - Le nom **AJAX** "Asynchronous JavaScript and XML" est obsolète car JSON est plus souvent utilisé que XML.
 - La technologie **AJAX** est moins utilisée de nos jours.
 - La technologie **Fetch** (Vanilla JS) est préférée.



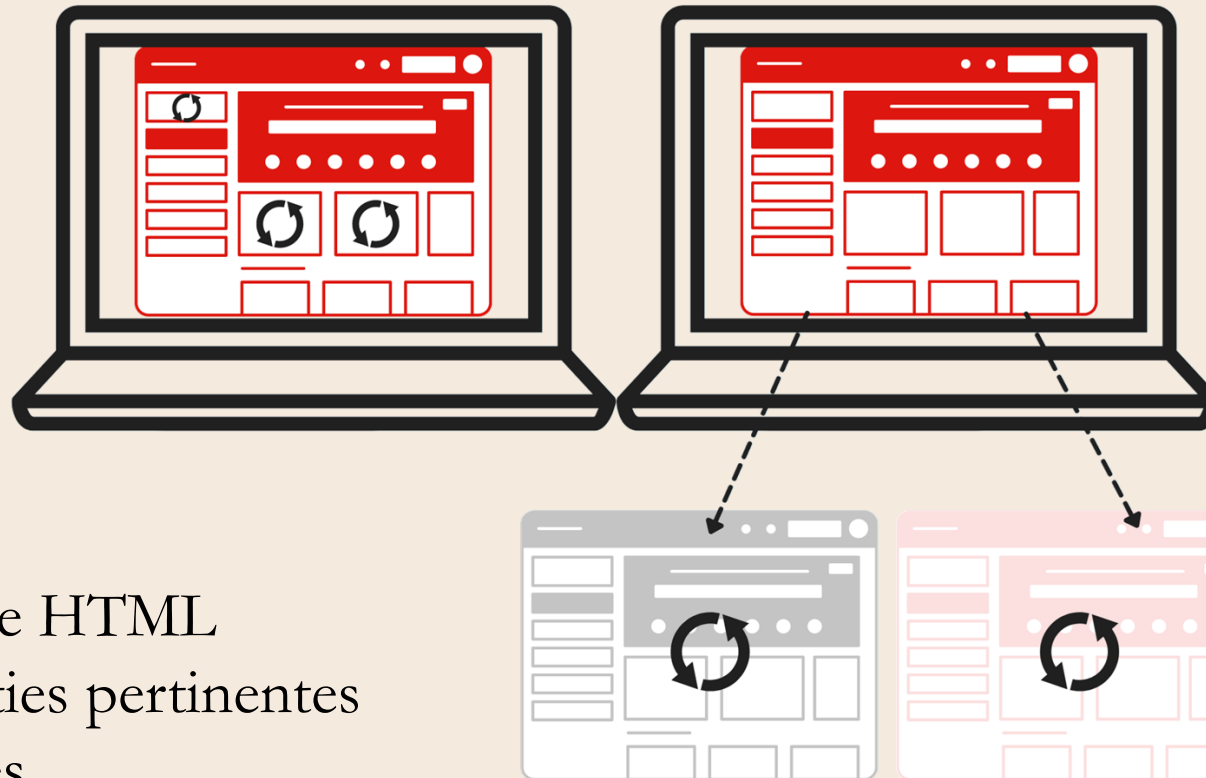
Rappel : Single Page App vs Mult. Page App

SPA vs MPA

~~`$.ajax()`~~
`fetch()`

SPA

- une seule page HTML
- seules les parties pertinentes sont modifiées

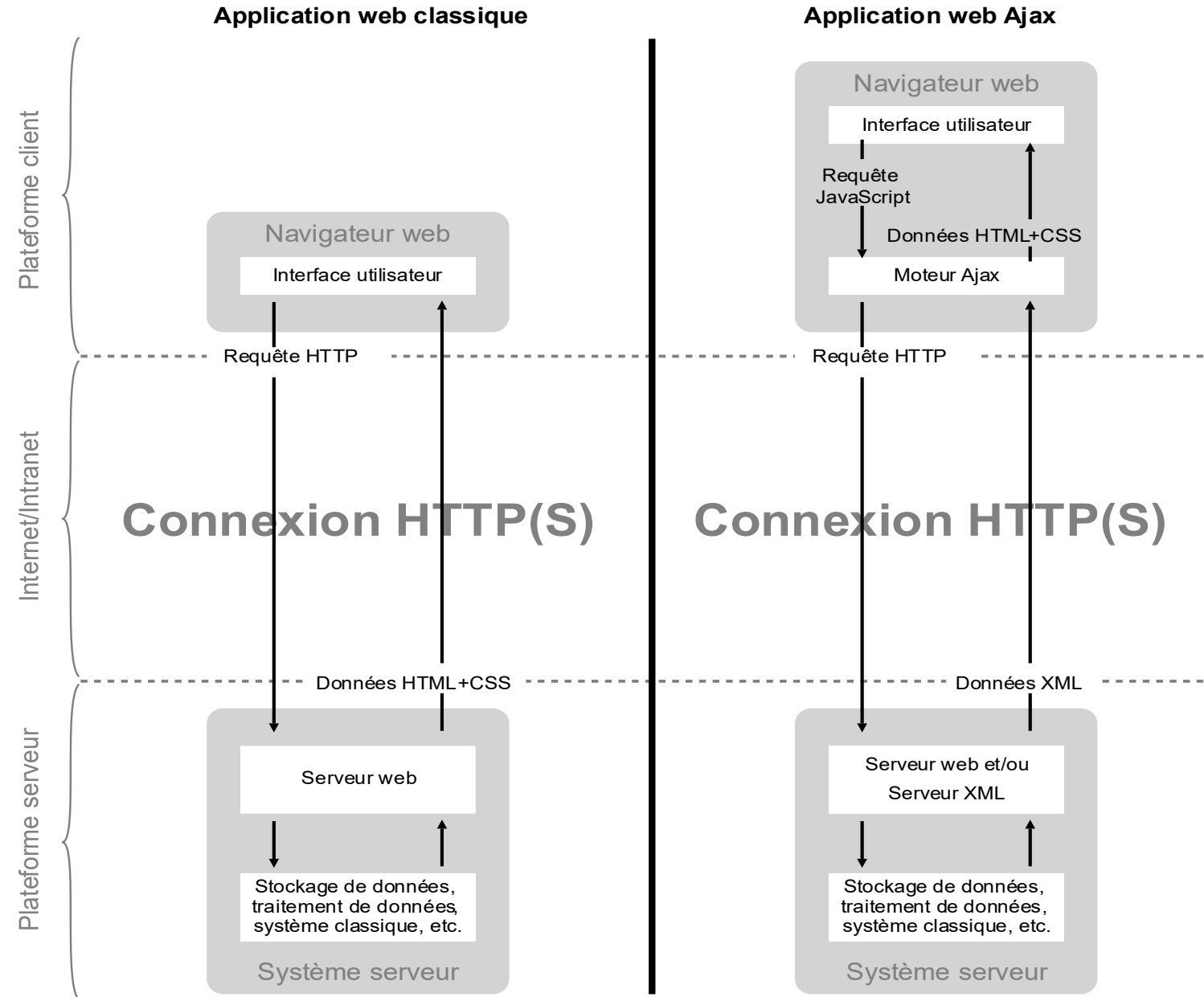


`<form>`

MPA

- plusieurs pages HTML
- page entièrement rechargée

Principe



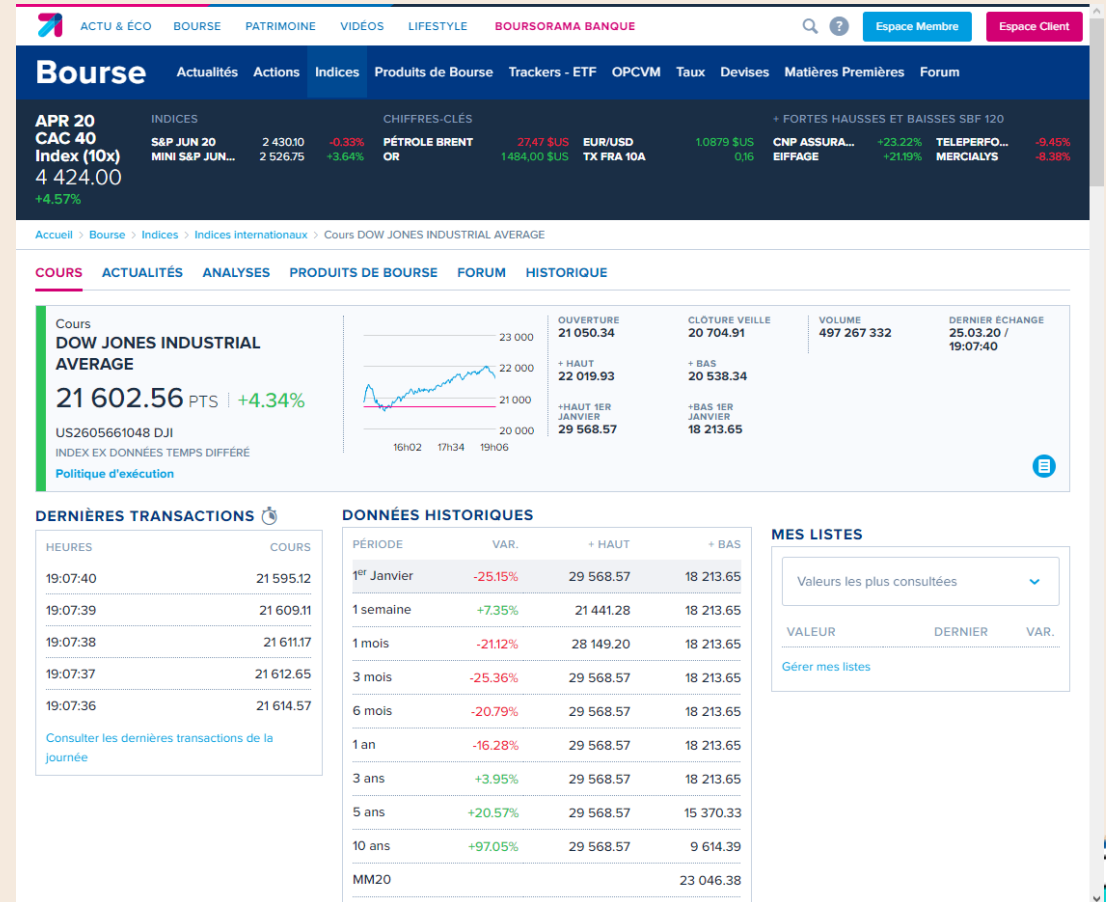
Motivation

- Dynamisation du rafraîchissement de la page
 - Page dynamique
- Amélioration des performances
 - Vitesse de rafraîchissement
 - Lisibilité accrue
 - Économie de données transférées



Exemple d'application

- Cours de la bourse
 - Dow Jones Industrial Average
 - Exemple sur Boursorama
 - Mise à jour chaque minute



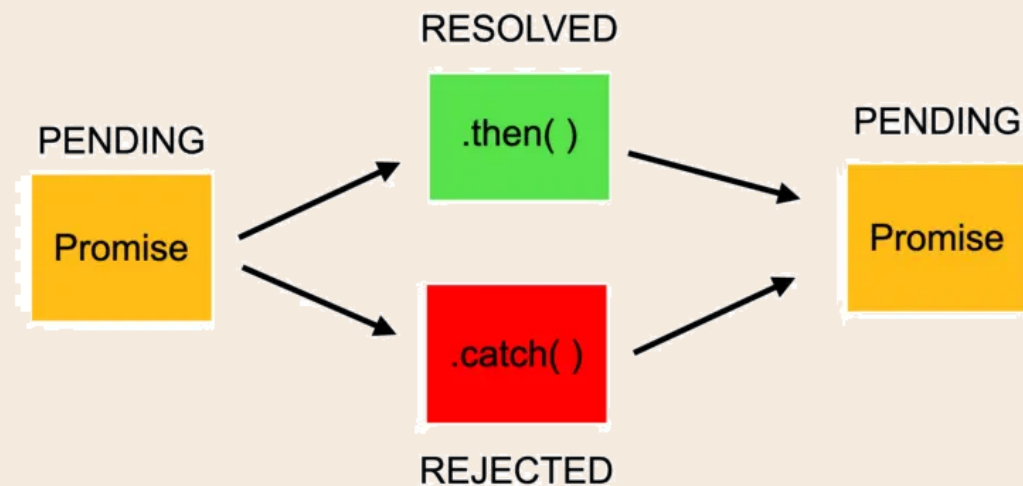
Objet JS XMLHttpRequest

- Rôle
 - effectuer des requêtes HTTP asynchrones depuis le navigateur vers un serveur.
- Gestion de tout type de données
 - JSON, texte, HTML, etc.
- Utilisation
 - base de l'AJAX traditionnel
- Inconvénients
 - Syntaxe verbeuse et peu intuitive
 - Gestion des erreurs moins pratique
- Événements
 - pour gérer les différentes étapes de la requête
 - **.onload**, **.onerror**, **.onprogress**
- Quelques propriétés
 - **.responseText**
 - réponse du serveur (texte)
 - **.status**
 - Code HTTP de la réponse
 - ex : 200, 404, ...
 - **.readyState**
 - état de la requête (0 à 4)

Objet JS Promises (promesses)

- objet JavaScript moderne représentant la valeur future d'une opération asynchrone
 - natif dans les navigateurs modernes
- **contrat** entre
 - le code qui lance l'opération asynchrone et
 - le code qui utilise son résultat.
- analogie :
 - Vous commandez un café dans un bar.
 - Le serveur vous **promet** de vous apporter un café.
 - Pendant que le café est préparé (opération asynchrone), vous continuez à discuter.
 - Une fois prêt, le serveur vous **livre** le café (la promesse est "résolue")
 - ou vous dit qu'il n'y a plus de café (la promesse est "rejetée")

Objet JS Promises



- Une promesse peut être dans un seul des trois états suivants
 - **pending**
 - l'opération asynchrone n'est pas encore terminée
 - **resolved**
 - l'opération a réussi
 - la promesse retourne une valeur
 - **rejected**
 - l'opération a échoué
 - la promesse retourne une erreur.
- Les promesses permettent d'**enchaîner** des opérations asynchrones.
 - lisibilité accrue

Implantation

AJAX (Jquery)

- Basé sur l'objet **XMLHttpRequest**
 - callbacks exécutés en fonction du résultat de la requête
 - success, error, complete
- Vanilla JS : syntaxe verbeuse
- Jquery : syntaxe intuitive
 - **\$.ajax(...)**
 - gestion automatique des erreurs
- Compatibilité avec tous les navigateurs
- Vintage

Fetch (Vanilla JS)

- Basé sur les **Promises**
 - enchaînement clair avec **.then()** et **.catch()**
 - plus flexible pour les requêtes complexes
- Vanilla JS : natif et syntaxe concise
- Nécessité de vérifier les réponses
 - Pas de gestion automatique des erreurs HTTP
- Compatibilité avec tous les navigateurs
- Moderne
 - chaînage de requête



Asynchronous JavaScript and XML (AJAX)

implantation dans JQuery

Architecture

- Pour définir complètement une requête AJAX, on spécifie
 1. Le sélecteur et l'évènement déclenchant l'appel AJAX
 2. L'élément éventuel à mettre à jour
 3. L'URL à charger
 - avec des paramètres éventuels, sous format JSON
 - à l'aide de méthodes jQuery
 - d'autres méthodes (Javascript) existent, non vues dans ce cours.
 4. Le code de retour ou "callback" éventuel
 - traitement des données retournées
- Serveur web indispensable (p.ex. WAMP)



Méthode `.load()` : exo 01

- Fichier : `exo0X_main.html`
- Chargement d'un contenu HTML statique (fichier HTML)
 - Même si tous les fichiers sont HTML, il faut malgré tout un serveur web
 - Méthode `$(selector).load(...)` avec arguments :
 - URL à charger
 - Code de rappel ou *callback* (optionnel)



Méthode .load() : exo 01

```
$('#maj').click(function() {  
    $('#div1').load(  
        'exo.html',  
        function() {  
            alert('zone mise à jour');  
        });  
});
```

1. Sélecteur et évènement déclenchant l'appel AJAX
2. Élément à mettre à jour
3. URL à charger
4. code de rappel (*callback*)



Méthode `.load()` : exo 02

- Fichier : `exo0X_main.html`
- Chargement d'un contenu HTML dynamique (fichier PHP)
 - Fichier PHP appelé (avec params)
 - Méthode `$(selector).load(...)` avec arguments :
 - URL à charger
 - Paramètres de l'URL (optionnel)
 - Callback (optionnel)



Méthode .load() : exo 02

```
$('#maj').click(function() {  
    var param = { number : 1234 };  
    $('#div1').load(  
        'exo.php',  
        param  
    );  
});
```

1. Sélecteur et évènement déclenchant l'appel AJAX
2. Élément à mettre à jour
3. URL à charger avec params JSON



Fonction \$.post() : exo 03

- Fichier : exo0X_main.html
- Chargement et traitement de données par AJAX
 - Fichier PHP appelé (avec params)
 - Il n'y a plus nécessairement d'élément à mettre à jour
 - Puisqu'on veut traiter des données ...
 - Méthode \$.post(...) avec arguments :
 - URL à charger
 - Paramètres de l'URL (optionnel)
 - Callback, ramenant les données en paramètre
 - Type de données retournées

https://www.w3schools.com/jquery/ajax_post.asp

Fonction \$.post() : exo 03

```
$('#maj').click( function() {  
    $.post(  
        'exo.php',  
        { number : 1234 },  
        function(data) {  
            alert(data);  
            $('#troisieme').html(data);  
        },  
        "html"  
    );  
});
```

1. Sélecteur et évènement déclenchant l'appel AJAX
2. URL à charger avec **params JSON**
3. Callback avec **données retournées**
4. **Format des données retournées**



Fonction \$.ajax()

la fonction JQuery la plus élaborée ...

\$.ajax({options});

- **options :**

- **type** : type de la requête, GET ou POST (GET par défaut).
- **url** : adresse à laquelle la requête doit être envoyée.
- **data** : données à envoyer au serveur.
- **dataType** : type des données qui doivent être retournées par le serveur : xml, html, script, json, text.
- **success** : fonction à appeler si la requête aboutit.
- **error** : fonction à appeler si la requête n'aboutit pas (p.ex.timeout dépassé)
- **timeout** : délai maximum (en millisecondes) pour que la requête soit exécutée.



AJAX & SESSION : exo 11

- Deux fichiers de départ, à installer :
 - Client : `exo11_card_client.php`
 - Serveur : `exo11_card_server.php`
- Situation courante :
 - Quand l'utilisateur ajoute un produit au panier (côté client), alors la super globale SESSION enregistre le choix (côté serveur).
- Votre mission :
 - Examinez ce code
 - Observez la fonction JS `display_card(...)`

Mon catalogue

Produit 1

Ajouter au panier

Produit 2

Ajouter au panier

Produit 3

Ajouter au panier

Produit 4

Ajouter au panier

Produit 5

Ajouter au panier

Produit 6

Ajouter au panier

Dans votre panier :

- produit5
- produit6

Effacer le panier



= BUROTIX 0

AJAX & SESSION : exo 12

- Deux fichiers de départ : idem exo 11
- Votre mission :
 - Quand l'utilisateur supprime son panier (côté client), alors la super globale SESSION est effacée (côté serveur) et donc le panier est vidé aussi à l'écran (côté client).



AJAX & SESSION : exo 13

- Deux fichiers de départ : idem exo 11
- Vos missions :
 - Introduire un bouton pour chaque produit, intitulé "retirer du panier", qui enlève le produit du panier.
 - Tuyau : Le code serveur doit aussi être modifié.
 - Quand l'utilisateur ajoute un produit au panier, alors le bouton "ajouter" propre à ce produit est désactivé et le bouton "retirer" activé.
 - Et vice-versa quand l'utilisateur retire un produit du panier...

Mon catalogue

Produit 1

Ajouter au panier

Retirer du panier

Produit 2

Ajouter au panier

Retirer du panier

Produit 3

Ajouter au panier

Retirer du panier

Produit 4

Ajouter au panier

Retirer du panier

Produit 5

Ajouter au panier

Retirer du panier

Produit 6

Ajouter au panier

Retirer du panier

Dans votre panier :

- produit5
- produit6

Effacer tout le panier



= BUROTIX 0

AJAX & database : exo 21

- Point de départ
- Application "database" du chapitre 11.
 - Application web liée à une base de données SQL
 - Démo de **SELECT**
 - Démo de **INSERT**
- Architecture
 - côté serveur : MVC (heureusement !)
 - côté client : classique, pas d'AJAX
 - Le bouton "**insert new deal**" rafraîchit toute la page.

Handling Database Using PHP, jQuery and AJAX.

Parties en cours

```
SELECT * FROM t_deal
```

- entre amis
- entre collègues
- ISFCE

Insérer une nouvelle partie

```
INSERT INTO t_deal (name_dea)  
VALUES (...)
```

Deal Name:

Insert New Deal

Messages du système



AJAX & database : exo 21

- Point de départ
- Fichiers de départ, à installer et à tester :
 - `exo21-conndb_start.php`
 - Lecture et édition d'une table.
 - Mini-architecture MVC : un seul fichier.
 - *Ajax-free* (au départ)
 - `exo21-db_import.sql`
 - Contenu de la base de données en SQL
 - A importer dans votre base MySQL

Handling Database Using PHP, jQuery and AJAX.

Parties en cours

```
SELECT * FROM t_deal
```

- entre amis
- entre collègues
- ISFCE

Insérer une nouvelle partie

```
INSERT INTO t_deal (name_dea)  
VALUES (...)
```

Deal Name:

Insert New Deal

Messages du système



AJAX & database : exo 21

- Convertissez cette appli en AJAX., càd
 - Supprimez le rafraichissement complet de la page.
 - Ne rafraichissez que les parties de la page nécessaires et suffisantes.
- Tuyaux :
 - Gardez tous les codes dans un seul fichier
 - Le bouton "**insert new deal**" doit être modifié.
 - Un évènement doit être défini pour le bouton.
 - Quand on clique sur le bouton, la base de données est mise à jour.
 - Puis "parties en cours" et "messages du système" sont rafraîchis.
 - L'architecture MVC est d'une grande aide.

Handling Database Using PHP, jQuery and AJAX.

Parties en cours

```
SELECT * FROM t_deal
```

- entre amis
- entre collègues
- ISFCE

Insérer une nouvelle partie

```
INSERT INTO t_deal (name_dea)  
VALUES (...)
```

Deal Name:

Insert New Deal

Messages du système

AJAX, timer & event

- Motivation : Utiliser une page web qui rafraîchit certains de ses éléments à intervalle régulier, sans intervention de l'utilisateur, avec des infos venant d'un serveur.
- Applications
 - Suivi en temps réel : bourse, actualité, météo, trafic, ..
 - Réseaux sociaux
 - Outils collaboratifs : chat, partage de fichiers, ...
- Architecture :
 - jQuery
 - définir le "timer" (cf chapitre jQuery)
 - définir la requête AJAX
 - AJAX
 - aller chercher l'info sur le serveur
 - Serveur
 - fournir l'info en base de données
 - aller chercher l'info sur un serveur externe

Cours
**DOW JONES INDUSTRIAL
AVERAGE**
21 602.56 PTS | **+4.34%**
US2605661048 DJI
INDEX EX DONNÉES TEMPS DIFFÉRÉ
[Politique d'exécution](#)



= BUROTIX 0

AJAX, timer & event : exo 31

- Exemple :
 - Afficher quelques cours de la bourse de New-York
 - Rafraichir ces cours à intervalle régulier, par ex. chaque 15 minutes
 - Fichiers, à installer et à tester :
 - `exo31_timer_client.php`
 - `jQuery`, `setInterval`, `AJAX`, `.append()`, `JSON`
 - `exo31_timer_server.php`
 - `PHP`, `JSON`, `CURL`, `site web externe`

Cours de la Bourse

20:52:06

Facebook, Inc.	173.65 USD
Alphabet Inc.	1208.155 USD
Dow Jones	23070.1 pts

21:07:07

Facebook, Inc.	173.86 USD
Alphabet Inc.	1212.32 USD
Dow Jones	23377.8 pts

21:22:07

Facebook, Inc.	173.46 USD
Alphabet Inc.	1207.86 USD
Dow Jones	23377.8 pts

21:37:07

Facebook, Inc.	173.7 USD
Alphabet Inc.	1209.26 USD
Dow Jones	23377.8 pts

21:52:07

Facebook, Inc.	174.45 USD
Alphabet Inc.	1211.35 USD
Dow Jones	23377.8 pts

AJAX, timer & event : exo 31 : client

```
function RefreshQuote()  
{  
    $.post(  
        'timer_server.php',  
        function(quote) {  
            $.each( quote, function( i, o )  
            {  
                s += o.name + o.value ;  
            });  
            $("div#quote").append(s);  
        },  
        "json"  
    );  
}  
  
// au chargement de la page  
$( function() {  
    setInterval(RefreshQuote, 1000*60*15);  
});
```

Remarque : code incomplet, cf fichier pour le code complet.

1. La fonction **RefreshQuote()** est appelée toutes les **15 minutes** grâce à **setInterval()**.
2. **RefreshQuote()** lance une requête AJAX vers le serveur **timer_server.php**
3. Celui-ci renvoie les données attendues **quote** sous format **JSON**.
4. Ces données **JSON** sont **traduites en HTML** puis **dispatchées sur la page web**.



= BUROTIX 0

AJAX, timer & event : exo 31 : serveur

```
$url = "https://financialmodeling.....";
$channel = curl_init();
curl_setopt($channel, CURLOPT_URL, $url);
$json_o = curl_exec($channel);

$qa = json_decode($json_o);
$facebook_o = some_processing($qa);

$final_a = array(
    array(
        'ticker'=> $facebook_o->symbol,
        'name'    => $facebook_o->name,
        'value'   => $facebook_o->price,
    ),
    ...
);
echo json_encode($final_a);
```

Remarque : code incomplet, cf fichier pour code complet.

1. L'API du serveur [financialmodelingprep](#) fournit les cours des actions d'entreprises cotées en bourse (*quote*).
2. La [technologie CURL](#) permet d'échanger des données avec différents serveurs et de **télécharger les données** du serveur [financialmodelingprep](#).
3. Les données récupérées sous **format JSON** sont **traitées** en PHP et un **array** est composé et retourné avec les informations nécessaires.
4. Cet **array final** est **converti en JSON** et **retourné au navigateur**.



= BUROTIX ()

Fetch

implantation dans Vanilla JS



Architecture

- Pour définir complètement une requête Fetch, on spécifie
 1. L'évènement déclenchant l'appel Fetch
 - .addEventListener()
 2. L'URL à charger
 - avec le header approprié
 - avec des paramètres éventuels, sous format JSON
 3. Les codes de retour .then()
 - traitement des données retournées
 - mise à jour de la page
 4. Les codes de retour .catch()
 - traitement des erreurs
- Serveur web indispensable (p.ex. WAMP)



Fonction `fetch` : exo 51

- Fichier : `exo5X_main.html`
- Chargement d'un contenu HTML statique (fichier HTML)
 - Même si tous les fichiers sont HTML, il faut malgré tout un serveur web
 - Fonction `fetch()` avec
 - arguments : URL
 - chaînage des promesses



Fonction fetch : exo 51

- évènement déclenchant le Fetch
- Élément à mettre à jour
- URL à charger
- code des promesses
- code d'erreur
- objet "response"

```
// add event on button
getElementById('maj1').addEventListener('click', () => fetchResponse(...));
...
function fetchResponse(button_id, target_id)
{
    fetch('exo_maj1.html')
    .then(response => {
        // get response
        if (!response.ok) throw new Error(`Erreur HTTP : ${response.status}`);
        return response.text(); // Récupère Le contenu HTML
    })
    .then(html => {
        // process response
        document.getElementById(target_id).innerHTML = html; // Insère Le HTML
        alert('La première zone a été mise à jour');
    })
    .catch(erreur => {
        alert('Erreur :', erreur);
    });
}
```



Fonction **fetch** : exo 52

- Fichier : exo5X_main.html
- Chargement d'un contenu HTML dynamique (fichier PHP)
 - Fichier PHP appelé (avec params)
 - Fonction **fetch()** avec arguments :
 - URL à charger
 - method (POST ou GET)
 - headers
 - body (paramètres, par ex. d'un formulaire)
 - chaînage des promesses



Fonction fetch : exo 52

```
...addEventListener(... fetchResponse('maj2')); // add event on button
```

```
...
function fetchResponse(button_id, target_id)
{
    // 1. Récupérer la valeur de l'input + conversion "clé=valeur"
    const param = { number: document.getElementById('ref2').value };
    const param_s = new URLSearchParams(param).toString()

    // 2. Envoyer la requête avec Fetch
    fetch('exo_proverbe.php', {
        method : 'POST',
        headers : {
            'Content-Type': 'application/x-www-form-urlencoded', // form & $_POST
        },
        body : param_s,
    })
}
./..
```

- Évènement déclenchant l'appel Fetch
- Élément à mettre à jour
- URL à charger avec params JSON
- paramètres de Fetch



- Élément à mettre à jour
- code des promesses
- code d'erreur
- objet "response"

Fonction fetch : exo 52

```
function fetchResponse(button_id, target_id) {  
  ./..  
  fetch(...)  
  .then(response => {  
    // 3. get response  
    if (!response.ok) throw new Error(`Erreur HTTP : ${response.status}`);  
    return response.text(); // Récupère Le contenu HTML  
  })  
  .then(html => {  
    // 4. Insérer la réponse dans l'élément button_id  
    document.getElementById(target_id).innerHTML = html; // Insère Le HTML  
    alert('La deuxième zone a été mise à jour');  
  })  
  .catch(erreur => {  
    alert('Erreur :', erreur);  
  });  
}
```



Remarque sur header et body

- Deux méthodes pour envoyer les params au serveur
- chaîne de requête URL-encodée
 - méthode traditionnelle : `clé1=valeur1&clé2=valeur2`
 - `headers` : {
 'Content-Type': 'application/x-www-form-urlencoded',
},
 body: `new URLSearchParams(param).toString()`,
- JSON
 - API REST modernes
 - `headers` : {
 'Content-Type': 'application/json',
},
 body: `JSON.stringify(param)`,





Fetch + MVC

implantation dans Vanilla JS

couple au framework AWebWiz

Exo : compteur

- Un compteur est disponible sur la home page. La valeur du compteur est stockée sur le serveur en variable **SESSION**. Un script **JS** avec **Fetch** gère l'évènement click sur le bouton.

Mise en place

- Répertoire (GitHub) :
`/AWebWiz_start_fetch_vuejs`
- Fichiers nouveaux ou modifiés
 - `/public/index.php`
 - `/app/controller/home.php`
 - `/app/view/template.php`
 - `/app/view/home.php`
 - `/public/js/internal/counter.js`
 - `/app/controller/counter_fetch.php`
 - `/public/css/internal/main.css`



/public/index.php

- Le code du router est enrichi pour intégrer la possibilité pour le serveur de retourner autre chose que du HTML, comme du JSON.

/app/controller/home.php

- Appel à la fonction PHP `html_fetch_sample()`

/app/view/template.php

- Inclusion du script `counter.js` sous `<head>`

- `<head>`

...

```
<script src="./js/internal/counter.js">...  
</head>
```

/app/view/home.php

- Déclaration de `html_fetch_sample()`
 - `button, id="b_compteur"`
 - `span, id="compteur"`

/public/js/internal/counter.js

- Définition des évènements sur le bouton **b_compteur**
 - click
 - callback : **getset_counter("increment")**
- Initialisation du compteur (chargement de la page)
 - appel à **getset_counter("get");**

/public/js/internal/counter.js

- Déclaration de la fonction JS `getset_counter(action)`
 - `param`
 - contient e.a. :

```
{  
  header : "application/json",  
  page : "counter_fetch",  
}
```
 - aiguillage de la requête sur le serveur
 - `fetch`
 - URL : `window.location.pathname`, càd le path courant
 - envoie `param`
 - reçoit du `JSON` en retour
 - valide le `JSON`
 - traite le `JSON` : affichage de la nouvelle valeur du compteur

/app/controller/counter_fetch.php

- Déclaration fonction PHP `main_counter_fetch()`
- Traitement du compteur
 - incrémentation,
 - sauvegarde, etc.
- Création d'un assoc array pour le retour
 - conversion en JSON (`json_encode`)
 - return
- Remarques :
 - La fonction retourne uniquement le tableau (JSON encodé).
 - L'en-tête est défini dans `index.php`.

Exo : Ajouter des fonctionnalités

- Ajouter un bouton "decrement".
- Ajouter un bouton "reset" qui initialise le compteur à 0.
- Ajouter un bouton "reset to value" qui initialise le compteur à une valeur donnée en input field.



Fetch + MVC + Vue.js

implantation dans Vanilla JS

couplage au framework AWebWiz

intégration dans Vue.js



Mise en place

- Répertoire (GitHub) :
`/AWebWiz_start_fetch_vuejs`
- Fichiers nouveaux ou modifiés
 - `/app/controller/home.php`
 - `/app/view/template.php`
 - `/app/view/home.php`
 - `/public/components/app.js`
 - `/public/components/counter.js`



/app/controller/home.php

- Appel à la fonction PHP `vuejs_fetch_sample()`

/app/view/template.php

- Inclusion des scripts de **Vue.js** sous **<head>**
- **<head>**

```
...  
  <script src="./js/vue.js/vue.global.js">..  
  <script type="importmap"> ...  
  <script type="module"  
    src="./js/internal/vuejs_fetch_app.js"  
    defer> ...  
</head>
```

/app/view/home.php

- Déclaration de `vuejs_fetch_sample()`
 - composant `Counter`

/public/components/app.js

- création de l'application **Vue.js**
- inclusion du composant **Counter**
- montage sur **section#vuejs_fetch_sample**

/public/components/counter.js

- **template**
 - bouton : évènement **click** vers méthode **fetchCounter**
 - **var compteur_vuejs**
- **data**
 - initialisation de **compteur_vuejs**
- **methods**
 - **fetchCounter**, cf description en section précédente (**counter.js**)
- **mounted**
 - pour initialiser le compteur au chargement de la page

Exo : Ajouter des fonctionnalités

- Ajouter un bouton "decrement".
- Ajouter un bouton "reset" qui initialise le compteur à 0.
- Ajouter un bouton "reset to value" qui initialise le compteur à une valeur donnée en input field.



Bachelier en Informatique de Gestion

Projet de Développement Web

FIN



= BUROTIX ()