Programmation du web - HAI305I

Michel Meynard

UM

Univ. Montpellier

Table des matières I

Table des matières II

Plan

Plan

Prérequis I

Les étudiants doivent avoir des connaissancesde base sur :

- le langage HTML ainsi que le protocole HTTP,
- notions de serveur HTTP, par exemple Apache ou Nginx ou Node.js ou ...
- client léger, navigateur Web, comme chrome, chromium, firefox, opera
- connaissance de base de Javascript : langage événementiel et fonctionnel dont un moteur (interpréteur) est incorporé dans le navigateur
- références :[1]
- conception de sites statiques à l'aide d'outils de son choix : emacs, vscode, sublime ...

Prérequis II

- En TP, il devra être capable de rechercher les références concernant HTML, Javascript, les feuilles de style CSS `https://www.w3schools.com/` ou `https://developer.mozilla.org/fr`
- références biblio. :[1]

Plan

Objectifs

Les objectifs de ce cours sont nombreux :

- apprentissage d'un langage de programmation côté serveur PHP ;
- étude de la dynamique d'une application Web : initialisation, saisie, traitement, ...
- technologies avancées : cookies, sessions, téléchargement, ...
- réalisation d'un projet.

Plan

HTTP et architecture du Web I

Quelques définitions de base sont nécessaires :

HTTP `HyperText Transfer Protocol` est un protocole de communication **client-serveur** développé pour le World Wide Web (www)

HTTP inventé par Tim Berners-Lee avec les adresses web (URL) et le langage HTML

port utilisé par défaut le port 80

HTTP est un protocole **non orienté connexion**

client HTTP (navigateur) envoie une requête au serveur qui lui retourne une réponse généralement sous la forme d'un fichier HTML. Le client affiche alors la réponse qui contient généralement des **liens** hypertextes ou des formulaires, qui une fois cliqués ou soumis génèrent une requête au serveur etc.

HTTP et architecture du Web II

serveur HTTP Apache, Nginx, IIS, ... sont des serveurs logiciels installés sur des machines appelées également "serveur" (matériel)

client HTTP appelé aussi client léger ou navigateur Web, les plus connus sont Firefox, Internet Explorer, Opera, ...

URL *Uniform Resource Locator* est une chaîne de caractères codée en ASCII pour adresser les ressources du World Wide Web : document HTML, image, son, forum Usenet, boîte aux lettres électroniques, etc.

exemple d'URL appelée aussi **adresse web** : `http://www.google.fr/`, `mailto:toto@titi.fr`, `http://www.lirmm.fr/~meynard/Ens2/rubrique.php3?id_rubrique=36&x=1`, ...

HTTP et architecture du Web III

URL relative à l'intérieur d'une page HTML, des liens vers d'autres pages du même site peuvent être définis avec une écriture relative au répertoire courant de la page : par exemple, `../images/toto.jpg` est une URL relative.

Plan

Application Web I

- HTTP n'est pas orienté connexion : le serveur ne mémorise aucune information à propos du client
- entre 2 requêtes du client, il peut se passer 2 secondes ou un temps infini !
- Donc l'écriture des applications côté serveur doit prendre en compte cette absence de mémoire
- une part non négligeable du traitement dans une application peut être effectué côté client afin d'effectuer des vérifications (contrôles) mais aussi des calculs métiers !
- éviter de surcharger le serveur
- des règles de sécurité interdisent aux scripts côté client d'accéder au ressources du poste client (fichiers, imprimante, ...) sauf si l'utilisateur le demande !

Application Web II

- Actuellement, avec les Single Page Application (SPA), l'application est située majoritairement sur le client qui la charge au début puis cette appli. fait appel à des Web Services (API Rest) quand cela est nécessaire.

Côté Serveur I

L'application côté serveur utilise plusieurs technologies possibles :

- **cgi** script (python, bash) ou binaire (compilé par g++) chargé dans un processus externe au serveur http (fork). La sortie standard de ce processus est redirigé dans la réponse envoyée par le serveur au client.
- **inconvénient principal des cgi** perte de temps nécessitée par la création d'un processus pour chaque nouvelle requête ;
- **module** le serveur intégre des modules interprétant des langages de programmation tels que Perl, Python, PHP. L'interprétation de script est beaucoup plus rapide car dans un Thread
- **Node.js** un serveur Node.js est un démon JavaScript qui tourne sur le serveur et qui sert chaque requête sans créer de processus léger ou lourd en utilisant le caractère asynchrone de Js !

Côté Client I

L'application côté client peut utiliser plusieurs technologies possibles :

- **javascript** langage de script interprété par le navigateur et permettant de manipuler l'arborescence du document(DOM) et de nombreuses autres API ;
- **applet Java** mini application Java permettant d'utiliser toute la puissance du langage Java (API, structures de données, ...)
- **ActionScript** langage de script compatible avec JavaScript et permettant de réaliser des animations Flash ou Flex ...

Plan

Plan

Histoire de HTML, XHTML, HTML5 I

- "HyperText Markup Language", HTML, est le langage conçu pour représenter les pages web.
- août 1991 : Tim Berners-Lee (CERN) annonce publiquement le web sur Usenet, en donnant l'URL d'un document de suffixe .html
- HTML basé sur SGML jugé trop complexe
- Différentes versions de HTML jusqu'à la version 4.01 puis ...
- "eXtensible HyperText Markup Language", XHTML 1.0, destiné à remplacer HTML 4 est une application XML.
- HTML5 apparue depuis 2006 sous la houlette du World Wide Web Consortium (W3C) et du Web Hypertext Application Technology Working Group (WHATWG)
- Beaucoup des recommandations de cette norme sont implémentées sur les navigateurs récents mais pas toutes ...

Histoire de HTML, XHTML, HTML5 II

- Nous utiliserons le HTML5 sans examiner toutes les capacités de ce langage.

Plan

Caractéristiques de HTML I

- A l'origine, contenu et format de document étaient mélangés dans un même fichier HTML
- éléments de structuration du contenu tels que `div`, `h1`, `h2`, `table`, ...
- éléments de mise en forme tels que `b` (bold), `i` (italic),`center`, ...
- Certaines mises en page utilisaient les tableaux, ce qui est maintenant blâmé
- Actuellement, feuilles de style (en cascade) "Cascading Style Sheets", CSS, pour le format
- La feuille de style est souvent externe au fichier HTML : fichier .css
- W3C (*World Wide Web Consortium*) chargé de rédiger des recommandations (sorte de norme) concernant les technologies du web `http://www.w3.org/`.

Caractéristiques de HTML II

- valider ses documents en ligne à l'adresse `http://validator.w3.org/`
- WHATWG (`https://whatwg.org/`) rédige lui le HTML living standard
- autre site sans lien avec le W3C `http://www.w3schools.com/` qui est un site pédagogique très pratique même s'il est commercial.

Plan

Vocabulaire I

- document : l'ensemble du texte composé du fichier principal et des fichiers inclus ou référencés (script, feuille de style, image, ...) ;
- Document Object Model (DOM) est le modèle arborescent d'une page Web : ses noeuds sont appelés `Node` ou pour les noeuds correspondant à deux balises `Element` ;
- langage à balisage : `<balise>contenu ...</balise>` ;
- élément : composé d'une balise ouvrante puis d'un contenu (possédant éventuellement d'autres éléments) puis de la balise fermante correspondante ;
- élément vide : ne possédant pas de contenu, la balise ouvrante est également fermante comme dans `<br />` ;
- attribut : information de la forme `nom='valeur'` présente uniquement dans une balise ouvrante ;

Vocabulaire II

- validité : un document HTML est valide s'il correspond aux règles édicté par le "World Wide Web Consortium" (w3c) ;
- URI : *Uniform Ressource Identifier* adresse d'une ressource Internet composé d'un protocole, d'un nom de domaine complètement qualifié (FQDN), d'un chemin, et d'autres paramètres possibles ; Par exemple, `http://www.lirmm.fr/~meynard/ProjetInfoL3/index.php?rubrique=Projet&action=liste` est une URI. Autre exemple, `mailto:toto@tutu.fr` désigne une adresse email ;

Plan

Syntaxe de HTML5 I

- HTML5 extrêmement laxiste vis-à-vis de XHTML 1.0 : les balises `html, head, body` sont optionnelles, les valeurs d'attribut ne sont pas forcément entourées de guillemets, les éléments vides ne sont pas forcément fermés ...
- Pour des raisons de lisibilité du code, nous continuerons à utiliser une syntaxe stricte héritée de xhtml
- les noms d'attribut sont en minuscules
- les valeurs d'attribut doivent être entre apostophes ou bien entre guillemets
- les éléments vides sont définis par une seule balise, comme dans `<br />` et dans `<img src="toto.jpg" />` ;
- les attributs présentiels n'ont pas de valeur puisque leur présence suffit (pseudo valeur booléenne) ; voir l'exemple suivant avec autofocus ou multiple.

Syntaxe de HTML5 II

```
<input type="text" name="prenom" autofocus />
<select multiple name="ues" ...
```


Exemple de document HTML minimum : modele.html I

```
<!doctype html>
<html lang="fr">
<head>
<meta charset="utf-8" />
<!-- <meta charset="iso-latin-1" / -->
<title>mon premier site</title>
</head>
<body>
<header>
<h1></h1>

</header>
<h1>Titre de niveau 1</h1>
<p>hello <b>World</b> ! éàè</p>
```

Exemple de document HTML minimum : modele.html II

```
</body>
</html>
```

Doctype DTD (Document Type Definition) définit la syntaxe d'un type de document : grammaire formelle indiquant les imbrications possibles d'éléments, les attributs obligatoires, optionels, ...

historique La déclaration de la DTD `<!DOCTYPE ...` était auparavant complexe : URI du fichier de DTD. Il y avait plusieurs DTD pour XHTML 1 (strict, transitional, frameset)

L'élément html la racine du document qui possède l'attibut de langue

head l'en-tête du document non visible ;

body le corps du document qui sera affiché sur la page ;

Exemple de document HTML minimum : modele.html III

L'en-tête du document L'élément `head` peut et **doit** contenir d'autres éléments que le titre du document `title` :

- l'élément `meta` fournit des méta-informations (ou métadonnées) sur la page en cours qui pourront être utilisées par les moteurs de recherche ou par le navigateur. Ses attributs :
  - **charset** indique l'encodage du fichier HTML. Pour le français les 2 encodages les plus fréquents sont l'`iso-latin-1` (1 octet par lettre accentuée), ou l'`utf-8` (codage multi-octets mais universel).
  - **name** est optionel et peux prendre les valeurs `author`, `description`, `keywords`. La valeur associée à ce nom sera dans l'attribut `content`.

Exemple de document HTML minimum : modele.html IV

http-equiv permet d'envoyer des en-têtes HTTP au navigateur pour l'aider à interpréter le document. Par exemple, `<meta http-equiv="refresh" content="5" />` permet de raffraîchir la page au bout de 5 secondes. La valeur associée à `http-equiv` sera dans l'attribut `content`.

content définit la valeur de la méta-information. Par exemple,

```
<meta name="keywords" content="fromage, cameml
```

- l'élément `script` permet de référencer un fichier script externe et/ou de définir des fonctions JavaScript locales. Par exemple,

```
<script charset="iso-8859-1" src="monscript.js"></script>
```


Exemple de document HTML minimum : modele.html V

Attention à ne pas créer un élément vide `<script ... />` lorsque tout le code est dans un fichier externe. Cela ne fonctionne pas !

- l'élément `link` permet de lier un autre document. Par exemple, pour définir la feuille de style associée :

```
<link rel="stylesheet" href="messtyles.css" />
```

- l'élément `style` permet de définire des styles en ligne sans utiliser de fichier externe.

Plan

Le corps du document I

Le corps (*body*) du document HTML constitue ce qui va être affiché sur l'écran. Par défaut, les éléments sont affichés selon leur type :

- les éléments en ligne (*inline*) sont placés de gauche à droite jusqu'à ce qu'il n'y ait plus de place dans leur bloc et un retour à la ligne automatique est géré par le navigateur ; ainsi le texte d'un paragraphe est il affiché avec des coupures de lignes dépendant de la taille de la fenêtre du navigateur ; de même les ancres sont des éléments en ligne ;
- les éléments de type `bloc` sont affichés avec une coupure de ligne avant et après eux ; les paragraphes sont de type bloc ainsi que les en-têtes.

Un grand nombre de balises *sémantiques* ont été ajoutées en HTML5 afin d'éviter la multiplication des blocs (`div`) pour des usages différents.

Plan

Attributs de base I

Un certain nombre d'attributs de base communs à quasiment tous les éléments (*core attributes*) peuvent être utilisés.

class classe CSS de l'élément. Cela permet de formatter l'élément selon un style défini dans une classe déjà définie ; par exemple, `class='monvert'` où monvert est une classe de style ;

style style en ligne ; par exemple, `style='color:green;'`

id identifiant **unique** de l'élément permettant sa manipulation en JavaScript grâce à la méthode : `document.getElementById(id)` ;

title *tooltip* affiché lors du survol de l'élément ;

Attributs de base II

De plus, une gestion des événements souris et clavier permettent d'associer des scripts JavaScript. Cette association est réalisée grâce à des attributs tels que `onclick` ou `onkeypress` qui peuvent être associés à un gestionnaire d'événements. Attention à ne pas confondre l'identifiant (id) qui est utilisé pour la manipulation côté client et le nom (name) des champs de formulaire qui lui caractérise les paramètres des requêtes passées au serveur lors de la soumission.

Plan

Eléments indispensables I

a *anchor* les ancres mettent en oeuvre l'hypertexte et peuvent être utilisées de 2 façons :

1. lien hypertexte référençant une autre page web grâce à l'attribut `href`. Par exemple, pour aller sur le site du `w3schools` :

   ```
   <a href='http://www.w3schools.com/'>CLIQUEZ
   ↪  ICI</a>
   ```

2. marque-page indiquant une cible potentielle d'un lien :

   ```
   <a id='toto'>Texte cible</a>
   ```

   Plus loin dans le même document, on pourra référencer cette cible par le lien suivant :

   ```
   <a href='#toto'>aller au texte cible</a>
   ```


Eléments indispensables II

Bien entendu, on peut faire une référence externe sur une cible en suffixant le chemin de l'url par `#toto` ;

Enfin, la valeur d'un href peut-être une pseudo-URL contenant du code JavaScript qui sera exécuté lors de l'activation du lien comme dans l'exemple suivant :

```
<a href="javascript:alert('Bonjour le monde !')">Mon
↪  Lien</a>
```

Remarquons que le code est préfixé par `javascript:` afin d'indiquer le langage ;

script exécution de code `javascript` dans le corps de page ; par exemple :

```
<script>document.writeln('Bonjour le monde
↪  !');</script>
```

p paragraphe contenant une suite de phrases ;

Eléments indispensables III

br *break* élément vide qui permet de passer à la ligne suivante ; en HTML les espaces, tabulations et retour ligne (CR et/ou LF) multiples ne sont perçus que comme un espace séparateur lorsqu'ils sont situés entre deux mots.

h1 *header* de niveau 1 est un titre de section ; on trouve également `h2` ...

ul *unordered list* liste d'items non numérotés ;

ol *ordered list* liste d'items numérotés ;

li *list item* élément d'une liste numérotée ou non ;

div *division* est une section logique du document permettant de regrouper plusieurs blocs dans un même formatage ; on l'utilise souvent dans la mise en page du document pour scinder les différentes parties (bandeau haut, menu de gauche, page centrale, bandeau du bas) ;

Eléments indispensables IV

table les tables HTML sont un moyen d'afficher les tableaux mais pas de mettre en page un document (utiliser plutôt les div). Les tables peuvent ou doivent contenir :

- caption la légende de la table ;
- tr *table row* une ligne ;
- td *table delimiter* une case ; `colspan` et `rowspan` sont des attributs indiquant le nombre de cases à fusionner avec la case courante vers la droite et vers le bas ;
- th *table header* une case de titre de colonne ;

hr *horizontal rule* ligne de séparation horizontale ;

img *image* en format gif, png ou jpg ; les attributs indispensables :

- src *source* chemin du fichier image ;

Eléments indispensables V

alt *alternative* texte utilisé si le navigateur ne sait pas afficher les images ;

width largeur en pixels ;

height hauteur en pixels ;

form les formulaires sont décrits dans la section suivante.

header HTML5 à ne pas confondre avec `head`, cet élément contient l'entête visuel d'un document ou d'une section (appelé aussi bandeau haut). Il inclut souvent une barre de menus (nav).

footer HTML5 contient le pied de page visuel d'un document ou d'une section. (appelé aussi bandeau bas).

nav HTML5 contient une liste de liens externes ou internes. Il est typiquement utilisé pour implémenter un menu ou un fil d'Ariane :

```
rayon informatique > clé USB > par marque
```


Eléments indispensables VI

article HTML5 portion de page indépendante du reste de la page ;

section HTML5 groupement de contenu incluant généralement un titre ;

aside HTML5 groupement de contenu non primordial ;

time HTML5 pour spécifier une date compréhensible par javascript ;

figure, figcaption HTML5 pour insérer un **flottant** composé d'images de textes et d'une légende,

Plan

Les formulaires I

- Les formulaires constituent des éléments indispensables dans la saisie d'informations à destination d'une application web
- Un ou plusieurs formulaires peuvent être présents dans une même page, seul le formulaire soumis sera envoyé
- L'attribut `name` de chaque champ de saisie correspond au nom du "paramètre" qui contiendra l'information
- Plusieurs champs peuvent avoir le même nom, dans ce cas le paramètre sera de type tableau si le nom est suffixé par `[]`
- L'attribut `tabindex` permet d'ordonner les champs de saisie quand l'utilisateur appuiera sur la touche de tabulation

Eléments de formulaires I

Dans HTML5, de nouveaux éléments de saisie sémantiques sont apparus associés à des contrôles de validité automatiques. Ils sont plus ou moins bien implémentés selon l'âge du navigateur.

form élément racine d'un formulaire contenant les attributs suivants :

action est l'URI où sera soumis le formulaire (généralement un script php ou un programme cgi). Cette URI peut être absolue ou relative au répertoire courant. La norme recquiert cet attribut qui doit être une URI, or une URI ne peut être vide mais, la norme HTML4.0 admet l'exception d'attribut vide pour désigner le document courant !

Eléments de formulaires II

method soit get, soit post ; la première (get) insère les champs saisis par l'utilisateur à la fin de l'url après un point d'interrogation sous la forme `http://localhost/index.php?nom1=val1&nom2=val2` ; les noms sont les valeurs des attributs `name` des champs tandis que les valeurs sont les contenus saisis par l'utilisateur. La seconde méthode post "cache" les contenus saisis ;

target avec la valeur `_blank`, une nouvelle fenêtre sera ouverte, avec `_self` (par défaut) on recouvrira la fenêtre actuelle.

onsubmit code javaScript à exécuter avant la soumission ; Si le code JavaScript retourne faux, la soumission **est annulée** !

Eléments de formulaires III

name ou id nom ou id du formulaire pouvant être utile pour le référencement des champs de saisie dans le code javaScript de vérification ;

label texte préfixant le champ de saisie et lié à lui par l'attribut `for`. Indispensable pour l'accessibilité.

input élément **vide** définissant un champ de saisie ayant :

- un attribut **name**, le nom du champ ;
- un attribut **type** permettant d'indiquer la forme du champ de saisie parmi :

  text champ de type texte libre sur une ligne ; autres attributs : `size` la taille du champ de saisie à l'écran, `maxlength` le nombre maximum de caractères à saisir, `value` valeur par défaut ;

Eléments de formulaires IV

password champ de mot de passe caché ;

button bouton permettant de déclencher un script javascript ; autres attributs : `value` valeur affichée à l'écran, `onclick` code javaScript à déclencher ;

checkbox case à cocher ; autres attributs : `value` valeur affectée au nom dans le cas où cette case est cochée, `checked="checked"` si on veut que la case soit cochée par défaut ;

Eléments de formulaires V

radio case à cocher **exclusive** ; tous les boutons radio mutuellement exclusifs doivent avoir le même attribut `name` ; autres attributs : `value` valeur affectée au nom dans le cas où cette case est cochée, `checked="checked"` si on veut que la case soit cochée par défaut ;

hidden champ caché n'apparaissant pas dans le formulaire ; historiquement, les champs cachés permettaient de faire transiter l'information passée de proche en proche lors de la navigation ; autres attributs : `value` ; Actuellement, le mécanisme de session est plus pratique à utiliser.

Eléments de formulaires VI

submit bouton de soumission permettant d'exécuter l'`action` du formulaire ; plusieurs boutons de soumission peuvent coexister dans un même formulaire ;

image bouton de soumission utilisant une image comme visuel ; autres attributs : `src`, `alt`,`width`, `height` ;

reset réinitialisation des champs du formulaire ;

file pour envoyer au serveur un fichier localisé chez le client ; le formulaire doit posséder un attribut `enctype` ayant la valeur `multipart/form-data` et sa méthode doit être `post` ; Il est également possible de limiter la taille du fichier à envoyer en ajoutant un champs caché nommé

Eléments de formulaires VII

`max_file_size` et de valeur la taille maximal acceptée en octets.

date HTML5 permet de saisir une date grâce à un calendrier ;

time HTML5 permet de saisir un horaire ;

datetime-local HTML5 permet de saisir un horaire et un jour ;

time, week, month, datetime HTML5 les dates sont retournées selon la norme RFC 3339, par exemple : `1985-04-12T23:20:50.52Z` ;

number HTML5 permet de saisir un nombre flottant dans un intervalle ;

```
<input type="number" name="qte" min="10" max="2
```

range HTML5 permet de saisir un nombre flottant grâce à un curseur ;

Eléments de formulaires VIII

```
<input type="range" name="son" min="10" ma
```

color HTML5 permet de saisir une couleur au format hexadécimal #1234bb (RVB) grâce à un nuancier de couleur.

```
<input type="color" name="coul" value="#12
```

email HTML5 permet de saisir une adresse email valide.

url HTML5 permet de saisir une url valide.

tel HTML5 permet de saisir un numéro de téléphone valide.

search HTML5 champ de texte assez classique dont la sémantique est d'indiquer un champ destiné à la recherche d'information.

Eléments de formulaires IX

L'attribut pseudo-booléen `disabled` permet de désactiver un champ de saisie (`input`) qui ne permet alors plus la saisie. Il ne sera pas soumis.

textarea élément non vide contenant une zone de texte multi-ligne ; attributs : `name`, `rows` nb de lignes, `cols` nb de colonnes ;

select élément non vide contenant une liste déroulante d'options ; attributs : `name`, `size` nb de lignes visibles, `multiple` pseudo-booléen indiquant que plusieurs options sont possibles ;

option élément non vide contenu dans `select` ; attributs : `value` valeur qui sera associée au `name` du `select` conteneur, `selected` de valeur `selected` indique si cette option est présélectionnée ; le contenu de l'élément option sera le texte affiché dans la liste déroulante ;

Eléments de formulaires X

datalist élément contenant une liste de choix possibles mais non limitatifs. Il permet de suggérer certaines valeurs habituelles pour un champ texte (avec auto-complétion) en permettant une saisie libre. Son attribut `id` devra être référencé en tant que valeur de l'attribut `list` du champ de texte.

```
<datalist id="lprenoms">
  <option value="Pierre">Pierre Durand</option>
  <option value="Michel">Michel Meynard</option>
  <option value="Paul">
</datalist>
<label for="pre">Prénom : </label><input type="text" i
```

button élément non vide permettant de transformer en bouton son contenu (span, text, ...). Son attribut type est très important :

Eléments de formulaires XI

- `submit` (défaut) : permet de valider le formulaire contenant ;
- `button` : permet d'exécuter du code javascript ;
- `reset` : pour effacer les champs déjà saisis.

Un formulaire de login

```
<form action="login.php" method="post" >
<label for="log">Nom : </label><input type="text" id="log"
↪  size="20" name="login" /><br />
<label for="pwd">Mot de passe : </label><input type="password"
↪  id="pwd" size="20" name="passwd" /><br />
<input type="submit" value="Valider">
</form>
```


Plan

La validation des données et l'aide à la saisie I

- Avant HTML5, la validation des données côté client devait être exécutée par du code Javascript
- Désormais, plus simple d'utiliser des éléments de formulaires sémantiques (`email`, `url`, `date`, `tel`, `color`, `number`, `range`) qui, par nature, sont vérifiés par le navigateur avant l'envoi du formulaire
- De plus, d'autres éléments de langages permettent d'assister l'utilisateur lors de la saisie
- **la validation côté client permet de ne pas submerger le serveur avec des données erronées ou incomplètes MAIS qu'en aucun cas, elle ne peut être suffisante**
- il faut absolument vérifier la correction des données soumises côté serveur avant de les traiter (SÉCURITÉ)

Expressions régulières I

- L'attribut `pattern` permet de définir une expression régulière qui doit correspondre au texte saisi dans un champ de type `text` lorsque la soumission est demandée
- Dans le cas où la saisie n'est pas correcte, un message "Veuillez respecter le format requis" sera affiché par le navigateur

Exemple

Exemple de validation

```
<input type="text" name="nomvar"
↪  pattern="[A-Za-z_][A-Za-z_0-9]*">
```

Renseigner un élément de saisie I

En raison des règles d'accessibilité, chaque élément de saisie doit être associé à un élément de `label` ayant un attribut `for` lont la valeur est l'`id` du champ de saisie :
Afin d'indiquer ce que doit contenir un champ, on peut utiliser des attributs tels que :

placeholder affiche un texte indicatif grisé dans le champ avant la saisie.

title affiche une info-bulle (*tooltip*) à côté du champ.

Aides à la saisie

```
<label for="netu">Numéro d'étudiant sur 8 chiffres</label>
<input type="text" name="numetu" id="netu" pattern="[0-9]{8}"
↪  placeholder="Numéro
 d'étudiant" title="par exemple : 20131234">
```


Champ obligatoire et ordre des éléments I

- l'attribut pseudo-booléen `required` d'imposer la saisie de cet élément
- A la validation, les éléments requis non saisis empêcheront la soumission et seront encadrés en rouge
- Afin qu'un élément ait le focus dès l'ouverture de la page, il suffit de lui ajouter (et seulement à lui) l'attribut pseudo-booléen `autofocus`
- Pour les autres champs, on utiliser l'attribut `tabindex` avec une valeur entière (entre 1 et n) pour indiquer l'ordre logique de saisie
- Evidemment, l'élément ayant l'`autofocus` devra avoir un `tabindex="1"`.
- Pour les vieux navigateurs, afin d'assurer une même interface visuelle et la validation aux différents utilisateurs de vieux navigateurs, on peut utiliser une librairie javascript telle que `H5F` ou `Webforms2` qui se chargent de vérifier la compatibilité du navigateur courant et d'émuler les fonctionalités manquantes avec du code javascript.

Plan

Inclusion d'autres objets I

- L'élément `audio` (HTML5) permet d'inclure un lecteur audio pour écouter un son au format mp3, wav, ou ogg. Ses attributs :

  src url du fichier ;
  controls pour afficher les boutons de pause, play, ...
  autoplay pour lancer le son dès le chargement ;
  loop pour répéter le son ;
  preload pour charger le son avant la page ;

- L'élément `video` (HTML5) permet d'inclure un lecteur video pour voir un fil au format mp4, ogg ou wbm. Ses attributs :

  src url du fichier ;
  controls pour afficher les boutons de pause, play, ...
  autoplay pour lancer le film dès le chargement ;
  loop pour répéter ;
  preload pour charger le film avant la page ;

Inclusion d'autres objets II

muted muet ;
poster url de l'image de remplacement pendant le téléchargement du film ou avant sa lecture ;
heighth, width taille de l'écran ;

- Des balises `sources` peuvent être utilisées afin d'indiquer différents fichiers supports en différents formats (audio ou vidéo). Cela assure une plus grande compatibilité car certains navigateurs ne supportent pas certains formats.

Exemple

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  Votre navigateur ne supporte pas la balise video !
</video>
```


Inclusion d'autres objets III

- Object
  L'élément `Object` permet d'inclure un objet typé dans une page HTML. Ses attributs sont :
  type indique le type de l'objet tel que : "text/html", "application/x-shockwave-flash", "video/x-msvideo", ...
  data url du fichier tel que : `../commun.html`, `ma_video.avi`, `animflash.swf`, ...
  width et `height` indique la taille de l'inclusion ;
  Cet élément n'est pas vide et peut ou doit contenir des sous-éléments **paramètres** utilisés par le navigateur pour un type d'objet particulier.

Exemple

```
<param name="autostart" value="true" />
<param name="loop" value="true" />
Texte alternatif
```


Plan

HTML5 c'est beaucoup plus !

Attention, HTML5 fournit bien d'autres fonctionalités qui ne sont pas décrites dans ce cours introductif. Nous citerons par exemple :

- Canvas et son API pour dessiner et traiter des images ;
- drag and drop ;
- XMLHttpRequest level 2 ou AJAX 2 ;
- géolocalisation ;
- applications offline (cache) ;
- WebSocket qui offre une connexion et la prog. événementielle ;
- Microdata pour décrire et marquer des parties de pages afin que des programme tiers (robots) puissent extraire automatiquement de l'information ;

Plan

Plan

Introduction I

Le langage CSS (Cascading Style Sheets : feuilles de style en cascade) sert à décrire la présentation des documents HTML et XML

- standards définissant CSS publiés par le World Wide Web Consortium (W3C)
- introduit au milieu des années 1990
- depuis CSS3, la nouvelle norme est rétrocompatible. Elle est découpée en modules d'un niveau :
  - Selectors :
  - Box Model ;
  - Backgrounds and Borders ;
  - Text Effects ;
  - 2D/3D Transformations ;
  - Animations ;
  - Multiple Column Layout ;
  - User Interface ;

Introduction II

- CSS 4 (2017) n'existera pas : il n'y a que des niveaux (level) d'un module : CSS Selectors Level 3, CSS Cascading and Inheritance Level 4, ... : `http://www.w3.org/TR/`

Plan

Objectifs I

- séparer la structure d'un document de ses styles de présentation ;
- définir le rendu d'un document en fonction du média de restitution et de ses capacités (type de moniteur ou de dispositif vocal), de celles du navigateur textuel, ...
- permettre la cascade des styles, c'est-à-dire un héritage multiple entre les origines (auteur, utilisateur), le média, la localisation des définitions de style (externe, en ligne, ...) ;

Plan

Syntaxe I

Un style est défini par :
`selecteur {prop1:val1; prop2:val2 val3 val4; prop3: v1, v2;}`

Le **sélecteur** définit l'**ensemble** des balises affectées par le style. Chacune des 53 **propriétés** est répartie dans un groupe parmi les 6 suivants :

font	contenant font-family, font-size, font-weight, ...
color, background	contenant color, background-color, background-repeat, background-image, ...
text	contenant text-align, text-indent, ...
box, layout	contenant padding, border, margin, width, height, display, float, ...
list	contenant list-style-type, list-style-position, ...
table	contenant caption-side, padding, border-collapse, ...

Les **valeurs** de propriétés sont réparties dans les catégories suivantes :

Syntaxe II

mot-clé non sensible à la classe tel que : red, underline, bold, top, collapse ...

longueur nombre décimal absolu ou relatif (démarrant par un signe) suivi par une unité sur 2 lettres. Les unités suivantes sont possibles :

- cm, mm centimètre, millimètre ;
- in inch (2.54cm) ;
- pt point (1/72 inch) ;
- pc pica (12pt ou 4.2mm) ;
- px pixel (dépend de la résolution de l'écran) ;
- em hauteur de la police courante ;
- ex hauteur d'un x de la police courante ;

Syntaxe III

Par exemple, +0.25mm, -3em, 100px, sont des longueurs correctes dont les deux premières indiquent un agrandissement et un rétrécissement par rapport à la valeur par défaut ;

pourcentage longueur relative telle que width :50%, line-height :120%, ...

couleur exprimée en notation RGB (Red, Green, Blue) par :

- 6 chiffres hexadécimaux tels que #FFFFFF pour le blanc ;
- 3 chiffres hexadécimaux tels que #000 pour le noir ; #F00 pour le rouge (chaque chiffre est répété) ;
- rgb(128,0,128) pour le pourpre (également rgb(50%,0,50%)) ;

url avec la notation `url(http://toto.fr/chemin)` ;

Plan

Exemple avec styles dans l'en-tête I

- styles définis dans l'en-tête html du document
- pour p, polices séparées par des virgules : **un seul** des éléments de la liste sera choisi (le premier possible)
- pour border de div, **concaténation** (sans virgule) de valeur (color, width, style).

```
<style type="text/css">
<!--
p {
  font-family : cursive, fantasy, monospace;
  line-height : 100%;
}
h1 {
  text-align: center;
  font-variant : small-caps;
}
```

Exemple avec styles dans l'en-tête II

```
div {
  float: left;
  width: 30%;
  border: blue 1px solid;
}
-->
</style>
</head>
<body> ...
```

- commentaires html entourant la définition des styles : navigateurs ne comprenant pas les styles CSS
- valider sa feuille de style : validateur du W3C : `http://jigsaw.w3.org/css-validator/`

Exemple avec styles dans l'en-tête III

- Les divisions flottantes permettent de placer les divisions dans le flot des boîtes affichées. Ici, trois divisions (3*30%<100%) pourront être placées horizontalement dans la page.

positionnement fixe du bandeau haut I

- ensemble de styles permettant de définir un bandeau haut (div) fixée en haut du navigateur
- une division principale qui lorsqu'elle défilera, passera sous le bandeau
- la propriété `z-index` définit l'empilement des boîtes à l'affichage (par défaut 0)

```
<style type="text/css">
<!--
h1 {
  text-align: center;
  color: rgb(20,152,12);
}
div.entete {
  position: fixed;
  z-index: 1;
  background-color:aquamarine;
```


positionnement fixe du bandeau haut II

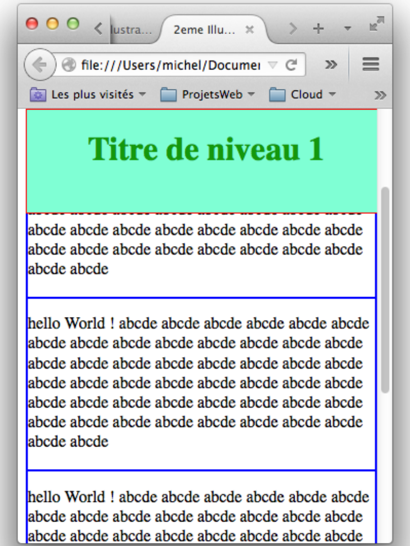
```
  top : 0px;
  height: 100px;
  width: 100%;
  border: red 1px solid
}
div.principale {
  position: absolute;
  top : 100px; // sous l'entete
  border: red 1px solid
}
div {
  float: left;
  border: blue 1px solid
}
-->
</style></head><body>
```

positionnement fixe du bandeau haut III

Dans le corps du document, on aura une structure en division telle que ce qui suit :

```
<div class="entete">
<h1>Titre de niveau 1</h1>
</div>

<div class="principale">

<div>
<p>hello World ! abcde abcde abcde abcde abcde
</div>

<div>
<p>hello World ! abcde abcde abcde abcde abcde
</div>
...
</div> <!-- principale -->
```


positionnement fixe du bandeau haut IV



Plan

Localisation des styles I

Les styles peuvent être définis de différentes façons et avec une priorité **décroissante** :

- **en ligne** à éviter car le style est un attribut de balise et la séparation contenu et forme n'est plus assurée ;
- **dans l'en-tête** comme dans le premier exemple, l'élément style peut être placé dans l'élément `head` du document html ; sa portée concerne **seulement** le document où il est défini ;
- **dans un autre fichier** de style d'extension `.css` qui sera lié aux documents html par un élément `link`. C'est la meilleure méthode ! On peut également lié le document html à plusieurs fichiers de styles : en cas de conflit, c'est le dernier fichier lié qui est prépondérant ;

Exemple de liaison à deux fichiers de style

Localisation des styles II

```
<!doctype html>
<html lang="fr"><head><meta charset="utf-8" />
<title>Fichier CSS externe</title>
<link rel="stylesheet" type="text/css" href="css3.css">
<link rel="stylesheet" type="text/css" href="css32.css">
</head>
```

Supposons que dans le fichier `css3.css`, on a `h1 {color: red;}` et que dans le fichier `css32.css`, on a `h1 {color: blue;}`. Alors, les éléments `h1` seront bleu.

Plan

Sélecteur I

- Le sélecteur définit les éléments affectés par les propriétés
- les sélecteurs simples tels que `p` ou `div` ou `h1` permettent de désigner tous les éléments de cette catégorie
- on peut raffiner la sélection en utilisant les nombreux procédés suivants

Correspondance de modèle (pattern matching) I

- Des expressions régulières permettent de sélectionner certains éléments du DOM
- Ces expressions régulières utilisent des opérandes (éléments notés E, F tels que `div` ou `p` ou des sélecteurs complexes)
- et les nombreux opérateurs suivants :

`*` correspond à tous les éléments de n'importe quel type ;

E F correspond aux élément F descendant de E ; par exemple : `h1 a {color : red}` correspond aux liens situés dans un titre de niveau 1 qui seront en rouge ;

E > F correspond aux éléments F fils de E ;
`ol > li {font-weight: bolder;}` met en gras les listes numérotées mais pas les autres (ul) ;

Correspondance de modèle (pattern matching) II

E + F correspond aux F qui sont un frère suivant d'un E ; par exemple : `h2 + p {color: yellow}` met en jaune le paragraphe qui suit un titre de niveau 2 ;

E [*nom* = "*val*"] correspond aux éléments E ayant un attribut `nom` de valeur `val` ; par exemple : `input[type="text"] {color: blue}` met les champs de saisie en bleu ; On peut ne pas mentionner la valeur afin de sélectionner tous les éléments ayant un attribut : `*[id]` sélectionne tous les éléments possédant l'attribut `id`.

Sélecteur multiple ,

Pour avoir tous les éléments de titre centrés :

```
h1, h2, h3 {text-align: center}
```


Plan

classes de style . I

Une classe de style permet d'affecter différents styles à un même élément HTML selon le contexte où il est utilisé. Par exemple, un paragraphe de résumé pourra être écrit en italique alors que les paragraphes "normaux" ne le seront pas. Pour cela, il suffit de définir un attribut `class` de l'élément à styliser.

```
p {
  font-family : monospace;
}
p.resume {
  font-style : italic;
  line-height : 80%;
}
```

Remarquons que les paragraphes de résumé héritent de la famille de police monospace. Voici l'utilisation des deux styles.

classes de style . II

```
<p class="resume">
  hello World ! abcde abcde abcde abcde abcde abcde abcde abcde abcde abcde
</p>
<p>abcde abcde abcde abcde abcde abcde abcde abcde abcde abcde
abcde abcde abcde abcde abcde abcde abcde abcde abcde abcde abcde abcde abcde
</p>
```

Cette façon de procéder est très fréquente pour définir des divisions sémantiques du document avec des `div` en leur affectant des classes différentes

Classe générique I

On peut créer une classe sans l'associer à un élément html. Dans ce cas, le sélecteur est composé d'un point suivi du nom de classe. Cette classe peut être affectée à des éléments de différents types :

```
.italique {font-style: italic;}
...
<p class="italique">bla bla </p>
<h4 class="italique">titre</h4>
```


Plan

Identifiant d'élément avec le caractère dièse

`#monid` permet de sélectionner l'unique élément html qui possède cet identifiant (`<div id="monid" ...`)

```
#valider { color : yellow;}
h1#premier { font-style : italic;}
```

Plan

Pseudo-classe [:] I

Les pseudo-classes et pseudo-éléments permettent de sélectionner des objets ou des classes qui ne sont pas des noeuds du DOM :

pseudo-élément la première ligne d'un paragraphe, ou un élément généré par la feuille de style ;

pseudo-classe classe acquise dynamiquement ou déduite d'un parcours du DOM ;

:first-child sélectionne seulement les paragraphes premiers fils : `p:first-child { text-indent: 0 }`

:link sélectionne les liens avant qu'ils aient été visités : `a:link { color: blue }`

:visited sélectionne les liens après qu'ils aient été visités : `a:visited { color: blue }`

Pseudo-classe [:] II

:focus sélectionne l'élément ayant le focus : `input[type="text"]:focus {color: red}`. Durant la frappe du texte, celui-ci est rouge ;

:hover sélectionne l'élément ayant le pointeur dessus (souvent utilisé pour les liens) : `input[type="text"]:hover {color: yellow}`. Durant le survol, le texte est jaune ;

:active sélectionne l'élément en train d'être activé (clic souris) : `input[type="text"]:active {color: black}`.

:first-line première ligne d'un paragraphe par exemple ;

:first-letter première lettre ;

:before pour générer du texte avant un certain élément :

:after pour générer un contenu (texte, image, ...) après un élément ;

Pseudo-classe [:] III

A noter que les 4 derniers types de sélecteurs sont des pseudo-éléments et non des pseudo-classes et qu'ils doivent donc être à la fin d'un sélecteur.
Exemples

- Le style suivant permettra de précéder chaque titre de niveau 1 d'un "Chapitre 12. " et chaque titre de niveau 2 d'un "Section 12.3. ".

```
body {
  counter-reset: chapter;       /* Crée un compteur de chapitre */
}
h1:before {
  content: "Chapitre " counter(chapter) ". "; /* contenu généré */
  counter-increment: chapter;  /* chapter++ */
  counter-reset: section;       /* Crée un compteur de section */
}
h2:before {
  content: counter(chapter) "." counter(section) ". ";
```

Pseudo-classe : IV

```
  counter-increment: section;
}
```

- Dans cette autre exemple, le texte “Fin du body” sera affiché en fin de page.

```
body:after {
  content: "Fin du body";
  display: block;
  margin-top: 2em;
  text-align: center;
}
```


Plan

Modèle de visualisation I

Le flux des éléments du document est affiché selon un modèle de visualisation utilisant le principe des “boîtes” TeX. Les éléments peuvent être classés en deux familles :

- Les éléments de type block (h1, p, ul, ol, dl, table, blockquote, etc.) ;
- Les éléments de type inline (a, img, strong, abbr, etc.) ;

Un élément de type block se différencie des éléments de type en ligne sur différents points :

- Il occupe l’entiereté de la largeur de son conteneur ;
- Il permet l’attribution de marges verticales ;
- Il permet la modification de sa hauteur et largeur ;

Tout élément peut être “reclassé” dans la famille opposée grâce à la propriété `display`. Le calcul des dimensions et de la position de ces boîtes est dynamique au fur et à mesure du chargement de la page et/ou de la dynamique interactive.

Modèle de visualisation II

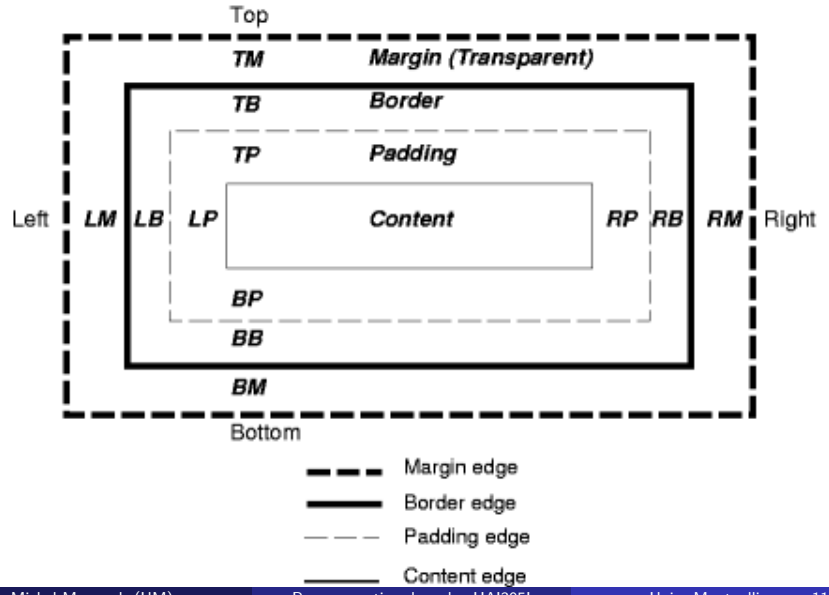


FIGURE – une boîte et son entourage

Plan

La propriété css position I

- On peut sortir un élément bloc du flux lorsqu'on envisage une mise en page un peu sophistiquée en utilisant la propriété **position**
- Une fois le type de `position` fixée, ce sont les propriétés `left, right, top, bottom` qui définiront les décalages.
- **La position statique** (`position:static`) correspond simplement à la valeur par défaut d'un élément du flux. Il n'y a que peu d'intérêt à la préciser, si ce n'est dans le but de rétablir dans le flux un élément en particulier parmi une série qui serait positionnée hors du flux

La propriété css position II

- **La position fixe** (`position:fixed`) positionne l'élément par rapport à la fenêtre du navigateur. L'élément est fixé à un endroit et ne pourra se mouvoir, même lors de la présence d'une barre de défilement. En d'autres termes, la position intiale est fixée au chargement de la page, le fait qu'une éventuelle scrollbar puisse être utilisée n'a aucune influence sur le positionnement de l'élément : il ne bouge plus de la position initialement définie.
- **La position absolue** (`position:absolute`) ressemble à la position fixe sauf qu'elle positionne l'élément par rapport à son premier ancêtre ayant une position autre que statique (body s'il n'en existe pas). `fixed, absolute` permettent la superposition. Un élément bénéficiant d'une position absolue ne bougera pas de sa position initiale tant que l'une des propriétés top, bottom, left ou right n'a pas été précisée ; il s'agit d'ailleurs là d'un comportement appliquable à toutes les positions. Dans le cas où un élément est en position absolue

La propriété css position III

mais n'a pas de coordonnées (left, top, ...), il est placé à la suite comme s'il était dans le flux, mais ses sucesseurs qui sont dans le flux viennent l'écraser (superposition).

- **La position relative** (`position:relative`) permet de décaler un élément par rapport à une position de référence : celle qu'il avait dans le flux. Les éléments qui le suivent et le précèdent ne sont pas influencés par ce décalage puisqu'ils considèrent que l'élément est toujours dans le flux à sa position initiale. Attribuer à un élément une position relative peut être pratique dans les situations suivantes :
  - Servir de référent à un élément enfant positionné en absolu (rappelons qu'un élément positionné absolument grâce aux propriétés top, left, ... le fera par rapport à la fenêtre du navigateur à défaut d'avoir un parent lui-même positionné) ;
  - Bénéficier de la possibilité d'utiliser la propriété z-index pour gérer des superpositions d'éléments (propriété inopérante pour des éléments du flux) ;

Plan

La propriété float I

La propriété float existe avant tout pour répondre à un besoin typographique précis : la création d'habillages. Un habillage est une pratique courante dans le média print consistant à "enrouler" un texte autour d'un élément (graphique ou texte).
À l'instar du positionnement absolu, un élément flottant adopte par défaut la largeur qu'occupe son contenu. Le principe de base est simple : un élément flottant est ôté partiellement du flux et placé à l'extrême gauche (`float:left`) ou droite (`float:right`) de son conteneur, forçant par la même occasion tout contenu du flux qui suit à l'envelopper. Deux objets flottants dans la même direction se rangeront côte à côte, seul un contenu demeuré dans le flux qui les succède *immédiatement* initiera l'habillage.

Plan

Type de media et résolution d'écran I

Depuis CSS2, on peut spécifier une feuille de style différente selon le media d'affichage de la page web :

```
<link rel="stylesheet" media="screen" href="screen.css" type="text/css" />
<link rel="stylesheet" media="print" href="print.css" type="text/css" />
```

On peut également spécifier des propriétés à l'intérieur d'une même feuille :

```
@media print {
  #menu, #footer, aside {
    display:none;
  }
  body {
    font-size:120%;
    color:black;
  }
}
```

Type de media et résolution d'écran II

Les différents media sont : screen, handheld (mobiles), print, aural (Synthèse vocale), braille, projection, tty (police à pas fixe), tv, all.
Afin de prendre en compte, les résolutions d'écran différentes (moniteur, tablette, smartphone), CSS3 a introduit les *media queries* qui permettent de spécifier un style conditionnel en fonction de la taille de la fenêtre. Un exemple suit :

```
.precisions span {
        display: none; // par défaut n'apparait pas
}
@media screen and (min-width: 1024px) and (max-width: 1280px) {
        .wide {
                background: #f11a57;     // change couleur et fond
                color:#fff;
        }
        .precisions span.regle3 {
                display: block;          // sauf si grand écran
```


Type de media et résolution d'écran III

```
        }
}
```

D'autres éléments de CSS3 telles que les grilles fluides dépassent l'objectif de ce cours !

Plan

Responsive web design I

Les tailles d'écrans de visualisation se démultipliant (mobile, tablette, TV, ...), une conception adaptée consiste à prendre en compte ces différents supports afin de permettre à chaque utilisateur de pouvoir utiliser le site web. Une solution simple consiste à définir son site pour l'écran le moins disant au niveau de sa définition, par exemple en définissant de manière fixe les éléments importants de votre blog :

```
article {
  width: 300px;
  height: 600px;
}
```

Une autre solution, bien meilleure, consiste à utiliser un framework CSS tel que bootstrap qui redéfinit le css en fonction de la largeur de l'écran grâce à l'élément méta à insérer dans l'entête (head) de la page :

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Plan

Plan

Introduction I

- Bootstrap est un framework CSS donc côté client (*frontend*) créé pour Twitter
- Il embarque également des composants HTML, JavaScript, JQuery
- Il comporte un système de **grille** simple pour organiser l'aspect visuel d'une page web "responsive"
- Il embarque des thèmes visuels, des icônes ...
- Il apporte du style pour les boutons, les formulaires, la navigation ...
- Il permet ainsi de concevoir un site web rapidement et avec peu de lignes de code (hum hum)
- Il utilise le "responsive web design" qui consiste à adapter la page Web au média utilisé.

Plan

De l'utilisation d'un framework I

Avantages

- les navigateurs ont des comportements très différents malgré leur lente convergence vers les standards. Les frameworks assurent une présentation similaire quel que soit le navigateur utilisé et une parfaite compatibilité ;
- bootsrap assure une accessibilité pour les utilisateurs handicapés ;
- les frameworks CSS font gagner du temps de développement **une fois qu'on les maîtrise** ;
- ils proposent un ensemble homogène de styles ;
- ils proposent en général une grille pour faciliter le positionnement des éléments ;
- ils offrent souvent des éléments complémentaires : boutons esthétiques, fil d'ariane, etc...

De l'utilisation d'un framework II

- la grande variété des nouveaux moyens de visualisation du web (smartphones, tablettes. . .) impose désormais la prise en compte de tailles d'écran très variées ; les frameworks CSS prennent généralement en compte cette contrainte.

Inconvénients

- temps d'apprentissage ;
- version 3 non compatible avec version 2

Plan

Caractéristiques I

- système de grille de 12 colonnes ;
- sous licence Apache, actuellement à la version 4.5 orientée mobile ;
- du code fondé sur HTML5 et CSS3 ;
- une bonne documentation ;
- une architecture basée sur Sass, un outil qui étend les possibilités de CSS ;

Plan

Installation I

- se rendre à l'url `http://getbootstrap.com` ;
- télécharger bootstrap sous forme d'un fichier zip contenant le code css compilé et minimisé ainsi que du javascript et des polices ;
- décompresser l'archive à la racine du site et renommer le répertoire en `bootstrap` (supprimer le numéro de version) ;
- insérer dans l'entête (head) des pages html ou php conforme au langage HTML5 le code suivant :

```
<link href="bootstrap/css/bootstrap.min.css" rel="stylesheet">
<script src="bootstrap/js/jquery.js"></script>
<script src="bootstrap/js/bootstrap.min.js"></script>
```

Il faudra bien entendu télécharger le fichier `jquery.js` dans le répertoire concerné ! En effet, la bibliothèque javascript`jquery` est utilisée par le code javascript de bootstrap. Attention à télécharger une version compatible !

Installation II

Remarques :

- Les fichiers minimisés (.min.cs ou .min.js) sont des versions compressées illisibles dans lesquelles les commentaires ont été supprimés
- Ils sont destinés à être chargés plus rapidement en mode production
- En mode développement, il est conseillé d'utiliser les versions non minimisées.
- Les CDN (*Content delivery network*) sont des serveurs qui mettent à disposition des librairies
- Il devient ainsi inutile de stocker ces librairies sur son propre serveur, il suffit de "pointer" vers eux
- L'avantage à utiliser un CDN est de diminuer l'utilisation de ressources (stockage et bande passante) sur son propre serveur mais surtout de factoriser ces ressources dans le cache du navigateur ce qui accélère le chargement des pages

Plan

La grille Bootstrap I

- découpage de la page Web en 12 colonnes de **même largeur**
- Les différents éléments de la page (banière, barre de menu, ...) seront disposés sur un nombre donné de ces colonnes
- La hauteur d'une ligne est déterminée par la hauteur de son plus gros élément
- Le nombre de colonnes affecté à un bloc (div) peut varier en fonction de la taille de l'écran
- 5 classes CSS génériques (i.e. `col-md-5`) permettent d'indiquer le nombre de colonnes (5) utilisées pour cet élément dans la grille **pour tous les écrans de largeur supérieure ou égale à la taille** (md), sauf si une autre spécification existe pour ces écrans (lg ou xl) !

Tailles d'écran :

rien pour les largeurs d'écrans $<$ 576 px : **Mobile First**

La grille Bootstrap II

-sm 576 $<=$ largeur pour les mobiles et plus ;

-md 768 px $<=$ largeur pour les tablettes et plus ;

-lg 992 px $<=$ largeur pour ordinateurs portables et plus ;

-xl 1200 px $<=$ largeur pour les grands écrans ;

- définir une mise en page de la grille spécifique pour un type d'écran, par exemple `md`
- Si l'utilisateur possède un écran `md` ou `lg`, la disposition prévue sera respectée
- Par contre, sur les écrans plus petits, un empilement vertical des éléments aura lieu afin de permettre leur lisibilité sur des mobiles par exemple
- On peut combiner différentes largeurs pour plusieurs types d'écran pour chaque élément afin de prévoir différentes disposition selon le type d'écran

2 dispositions distinctes de blocs I

Le code source HTML

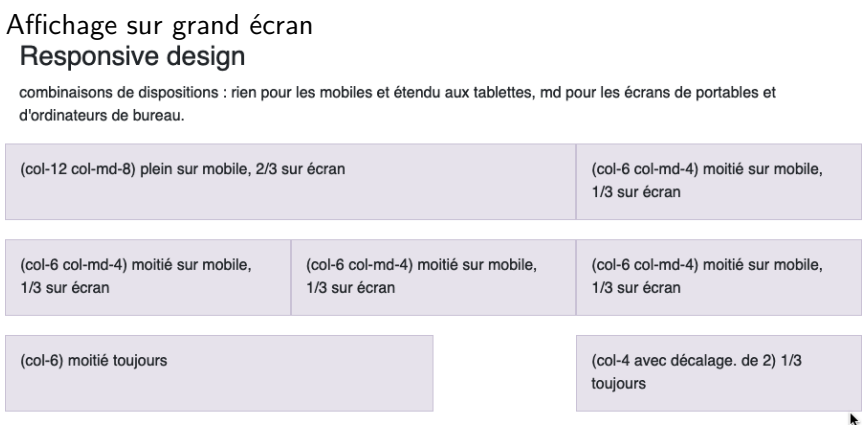
```
<body>
<div class="container-fluid">
<h3>Responsive design</h3>
<p>combinaisons de dispositions : rien pour les mobiles et étendu
aux tablettes, md pour les écrans de portables et d'ordinateurs
de bureau.</p>
<div class="row">
  <div class="col-12 col-md-8">(col-12 col-md-8) plein sur
  ↪  mobile,
2/3 sur écran</div>
  <div class="col-6 col-md-4">(col-6 col-md-4) moitié sur mobile,
1/3 sur écran</div>
</div>
<div class="row">
  <div class="col-6 col-md-4">(col-6 col-md-4) moitié sur mobile,
```


2 dispositions distinctes de blocs II

```
1/3 sur écran</div>
  <div class="col-6 col-md-4">(col-6 col-md-4) moitié sur mobile,
1/3 sur écran</div>
  <div class="col-6 col-md-4">(col-6 col-md-4) moitié sur mobile,
1/3 sur écran</div>
</div>
<div class="row">
  <div class="col-6">(col-6) moitié toujours</div>
  <div class="col-4 offset-2">(col-4 avec décalage. de 2)
1/3 toujours</div>
</div></div>
```

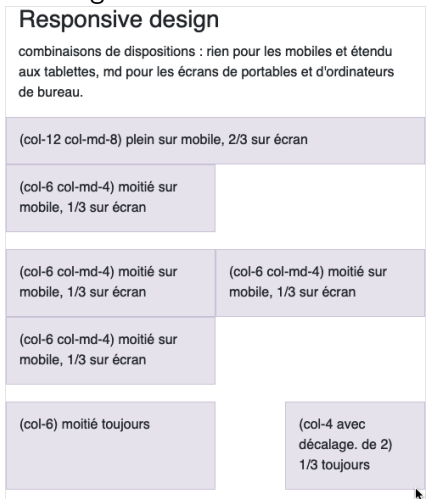
2 dispositions distinctes de blocs III

Affichage sur grand écran



2 dispositions distinctes de blocs IV

Affichage sur mobile



2 dispositions distinctes de blocs V

- le principe de la grille de 12 est récursif, ce qui fait qu'ont peut emboîter des blocs à l'intérieur
- Bootstrap propose sur son site des exemples de dispositions de pages
- Le site `http://www.codeply.com/` propose un éditeur graphique en ligne afin de prototyper l'apparence de son projet

Plan

Eléments de base I

Afin de créer un rendu visuel cohérent pour que tous les éléments cohabitent de façon esthétique, Bootstrap impose une certains nombre de principes :

- Tous les éléments sont englobés dans une racine `<div class="container">` fille de `body` ; Pour les applications (sans marges latérales) la classe `container-fluid` est plus adaptée ;
- l'emboîtement (row in col), le déplacement (offset), le changement d'ordre des colonnes (push) sont possibles ;
- des titres de h1 à h6 sont disponibles avec un sous-titre en ligne grâce à la classe `small` ;
- la taille de police par défaut du body est de 14px, la hauteur de ligne de 1,428 soit près de 20px ;

Eléments de base II

- diverses balises sont stylées pour les listes ul, ol, dl (description list=dictionnaire), les abbréviations (abbr), les citations (blocquote), les alignements gauche droite justifié,
- les tables html possèdent différentes classes intéressantes :

  .table-bordered une bordure simple encadre chaque cellule ;

  .table-hover la ligne est surlignée lorsque la souris survole ;

  .table-responsive emboîter une table dans une table-responsive ajoutera à ascenseur horizontal sur les petits écrans ;
- les formulaires sont divisés en plusieurs classes :

  par défaut les contrôles (input, select,) ont une largeur de 100% (formulaires destinés au téléphones mobiles) ; label et input sont emboîtés verticalement dans une div de classe `form-group` ;

  form-inline permet de placer les labels et leur contrôle en ligne ;

Eléments de base III

form-horizontal permet d'utiliser le principe de la grille afin de placer le label dans la colonne de gauche, et le contrôle dans la colonne de droite ; en cas d'utilisation avec petit écran, l'empilement reprend le dessus et l'aspect redevient celui de la classe par défaut.

Les différents contrôles HTML doivent posséder la class `form-control` et doivent être emboîtés dans une div de class `form-group`. De plus, une autre classe peut être ajoutée (en javascript ou PHP) pour indiquer la qualité de la validation des données saisies : `has-success`, `has-warning`, `has-error`. De nombreux types de boutons existent (forme, couleur) et l'on peut représenter différents éléments html tels que les liens (a href), les button, les input de type button et les input de type submit avec la même représentation graphique !

- Des menus dropdown, des groupes de boutons, ...

Eléments de base IV

- Une navigation est une liste non numérotée (ul) possédant la classe nav et une sous-classe précisant la visualisation des li : nav-tabs pour des onglets, nav-pills pour des boutons accolés. Chaque item de liste peut contenir un simple lien ou un un dropdown-toggle permettant de basculer un menu ;
- Une barre de navigation est un bloc `<nav>` possédant la classe `navbar` et offre un widget réactif (responsive) pour réaliser le menu principal d'un site. Il peut réunir un logo, des dropdown, un formulaire de recherche , un autre de connexion, ...
- un fil d'ariane (breadcrumb) affiche la hiérarchie de navigation en séparant les liens par des slash. Il est réalisé avec une liste ordonnée (ol) possédant la classe breadcrumb.

Plan

Barre de navigation, menu et formulaire I

```
<meta name="viewport" content="width=device-width,
↪  initial-scale=1.0">
<!-- Bootstrap -->
<link rel="stylesheet"
↪  href="https://stackpath.bootstrapcdn.com/...>
</head><body>
<!-- Fixed navbar -->
<div class="navbar navbar-default navbar-fixed-top">
  <div class="container">
    <div class="navbar-header"><!-- réduction sur mobile -->
      <button type="button" class="navbar-toggle"
      ↪  data-toggle="collapse"
        data-target=".navbar-collapse"> <!-- bouton d'expansion
        ↪  du menu -->
        <span class="icon-bar"></span>  <!-- 3 tirets horizontaux
        ↪  -->
```


Barre de navigation, menu et formulaire II

```
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="#">Tournoi</a>  <!-- titre
      ↪  -->
    </div>
    <div class="navbar-collapse collapse">
      <ul class="nav navbar-nav">
        <li class="active"><a href="#">Home</a></li>
        <li><a href="#about">About</a></li>
        <li><a href="#contact">Contact</a></li>
        <li class="dropdown">
        <a href="#" class="dropdown-toggle"
        ↪  data-toggle="dropdown">Equipe <b
        ↪  class="caret"></b></a>
        <ul class="dropdown-menu">
          <li><a href="#">Créer</a></li>
```


Barre de navigation, menu et formulaire III

```
          <li><a href="#">Supprimer</a></li>
          <li><a href="#">Mettre à jour</a></li>
          <li class="divider"></li>
          <li class="dropdown-header">Inscription</li>
          <li><a href="#">Inscrire</a></li>
          <li><a href="#">Désinscrire</a></li>
        </ul>
      </li>
      </ul>
      <ul class="nav navbar-nav navbar-right">
        <li class="active"><a href="./">Lien à droite</a></li>
      </ul>
    </div>  <!--/.nav-collapse -->
  </div>
</div>
<div class="container">
  <!-- Main component -->
```

Barre de navigation, menu et formulaire IV

```
  <div class="jumbotron">
    <h1>Exemple de navbar</h1>
    <p>Cet exemple de barre de menu horizontale fixe en haut de
    ↪  page permet de tester les navbar.
    <p>
      <a class="btn btn-lg btn-primary" href="#">Bouton lien
      ↪  &raquo;</a>
    </p>
  </div>
</div> <!-- /container -->
<div class="container">
<form class="form-horizontal" role="form">
  <div class="form-group">
    <label for="inputEmail1" class="col-lg-2
    ↪  control-label">Email</label>
    <div class="col-lg-10">
      <input type="email" class="form-control" id="inputEmail1"
      ↪  placeholder="Email">
```


Barre de navigation, menu et formulaire V

```
    </div>
  </div>
  <div class="form-group">
    <label for="inputPassword1" class="col-lg-2
    ↪  control-label">Password</label>
    <div class="col-lg-10">
      <input type="password" class="form-control"
      ↪  id="inputPassword1" placeholder="Password">
    </div>
  </div>
  <div class="form-group">
    <div class="col-lg-offset-2 col-lg-10">
      <div class="checkbox">
        <label>
          <input type="checkbox"> Remember me
        </label>
      </div>
```


Barre de navigation, menu et formulaire VI

```
    </div>
  </div>
  <div class="form-group">
    <div class="col-lg-offset-2 col-lg-10">
      <button type="submit" class="btn btn-default">Sign
      ↪  in</button>
    </div>
  </div>
</form>
  <!-- at the end of the document so the pages load faster -->
  <!-- jQuery ! (necessary for Bootstrap's js plugins) -->
<script src="https://code..../jquery.min.js" ...></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js..."></script>
<script src="https://stackpath.../bootstrap.min.js" ...></script>
</body></html>
```


Barre de navigation, menu et formulaire VII

FIGURE – barre de navigation fixe et formulaire horizontal sur grand écran

Barre de navigation, menu et formulaire VIII

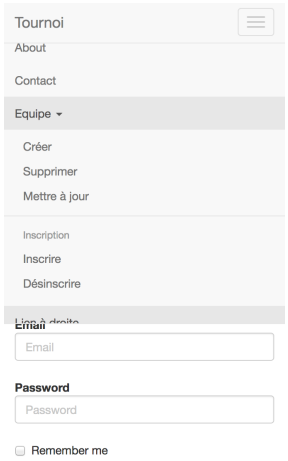


FIGURE – barre de navigation fixe et formulaire sur petit écran

Barre de navigation, menu et formulaire IX

Remarquons que dans cet exemple, la modification de la taille de l'écran induit de grandes différences d'interface du menu et du formulaire.

Plan

Plan

Introduction I

- PHP : acronyme récursif pour "PHP Hypertext Preprocessor"
- langage de script (interprété) et Open Source, spécialement conçu pour le développement d'applications web
- le script PHP est **intégré** au HTML
- La référence du langage est sur un site Web [5] qu'il faut toujours avoir en face des yeux quand on programme

Exemple (hello.php)

```
<html><head><title>premier prg php</title></head><body>
<?php
  echo "Bonjour le monde !";
?>
</body></html>
```


Introduction II

- différence avec les autres scripts CGI (Perl, C, ...) : page HTML avec du code inclus à l'intérieur :
  - pas besoin de générer le code HTML ;
  - prototypage du squelette du site par des spécialistes graphiques ;
- différence avec Javascript : le code est exécuté sur le serveur ;
- langage extrêmement simple et fonctionnalités avancées ;
- l'interprète PHP peut être appelé par :
  - le serveur Web comme module ou comme CGI ;
  - l'interpréteur de commandes de votre système : `php hello.php`.
  - en mode **interactif** (`php -a`), une session d'interprétation s'ouvre et vous pouvez tester vos instructions (en pensant à démarrer par `<?php`)

Caractéristiques I

- Différentes versions : PHP2FI, PHP3, PH4, PHP5, PHP7
- Nous traiterons de **PHP7** (PHP8 en version Beta)
- L'interprète PHP fonctionne sur le fichier interprété comme un **filtre** laissant passer de l'entrée standard vers la sortie standard tout ce qui n'est pas un bloc php (entre `<?php` et `?>`)
- Les sorties des blocs php (`echo, print(), var_dump()`) sont insérés dans le code HTML à la place du bloc
- si plusieurs *blocs* php existent, les définitions du premier bloc sont utilisables dans tous les autres (unique contexte d'exécution)

Caractéristiques II

Exemple

```
<html><head><title>deuxieme prg php</title></head><body>
<?php
$i=123;
?>
bla bla bla
<?php
echo "la variable i vaut : $i";
?>
</body></html>
```

Le fichier d'extension `.php` doit être lisible par le serveur HTTP (`chmod a+r hello.php`). Le droit d'exécution n'est pas nécessaire.

Plan

Structure du langage I

- Au niveau syntaxique, le langage PHP ressemble fortement au langage C++
- Différence essentielle : il est typé dynamiquement
- Toutes les variables doivent être **préfixées par un $** : c'est la cause principale d'erreur syntaxique. Il faut donc répéter que TOUTES LES VARIABLES DOIVENT ÊTRE PRÉFIXÉES PAR UN $!
- Originellement, la structure d'un script était un ensemble de fonctions
- Depuis PHP5 un modèle objet à classe permet de programmer par objets

Types scalaires I

Les types scalaires sont :

- integer avec des littéraux tels que : 10, -454, 0777, 0xABCD ;
- float avec des littéraux tels que : 1.24, -1.2e3, 5E-6 ;
- string avec des littéraux tels que : 'hello world', `"hello $i world"` ; Entre guillemets, les variables scalaires sont substituées par leur valeur (pas entre apostrophes) ! De nombreuses fonctions strxxx existent !
- boolean : true et false (insensibles à la casse) ;
- null : la valeur spéciale NULL représente l'absence de valeur (insensible à la casse).

Typage dynamique I

Une variable n'est pas typée statiquement mais dynamiquement au fur et à mesure de ses affectations successives. La conversion de type est automatique et dépend du contexte.

Exemple

```
<?php
$x = "0";  // $x est une chaîne de caractères (ASCII 48)
$x += 2;   // $x est maintenant du type entier (2)
$x = $x + 1.3;  // $x est maintenant du type double (3.3)
$x = 5 + "10 roses"; // $x est du type entier (15)
?>
```

On peut forcer le type d'une variable avec :

- la fonction settype() ;
- ou le transtypage à la C : `$x = (int) 5.6;`

Typage dynamique II

On peut tester le type d'une variable avec :

- `bool is_int(mixed $var)`
- `bool is_string(mixed $var)`
- ...

le type `mixed` signifie un type quelconque (string ou array ou object)

Les variables I

- Les variables sont créées dynamiquement dans le contexte d'exécution au fur et à mesure de l'exécution du script PHP
- La portée d'une variable peut être **locale** à la fonction où elle est définie ou **globale** lorsqu'elle définie dans le script en dehors de toute fonction
- Les variables globales ne sont pas accessibles simplement dans les fonctions (différent de C++) : Il faut, dans la fonction, utiliser la notation : `$GLOBALS['glob']` ou la déclaration : `global $glob,$g2;` pour y accéder en lecture et écriture
- `$GLOBALS` est le tableau **superglobal** des variables globales

Les variables II

- Un tableau superglobal est une variable globale accessible directement dans les fonctions (comme en C++). Un petit nombre de tableaux superglobaux sont **prédéfinis** en PHP : `$_GET, $_POST, $_SESSION, $_COOKIE, $_ENV, ...` . Il n'est pas possible d'en définir d'autres ;
- La fonction `isset($x)` permet de tester l'existence d'une variable ce qui s'avère primordial dans le cadre de la soumission de formulaire HTML pour savoir si un paramètre a été fourni
- On peut aussi utiliser `empty($x)` qui teste la non-existence ou la vacuité de `$x`.
- Pour supprimer une variable de son contexte, il faut utiliser la fonction `unset($x)`

Évaluation et comparaison I

- Dans les structures de contrôle (if, while, ...), les comparaisons entre variables de différents types sont réalisées par des coercitions (cast) automatiques nombreuses et parfois déconcertantes : `false==0`, `""==0`.
- On peut utiliser l'opérateur `===` qui teste l'égalité de type ET de valeur
- L'opérateur `!==` teste la différence de type ou de valeur
- En cas de doute, il est préférable d'utiliser l'égalité ou la différence triple

Constantes I

Comme en C, les constantes sont des macros, elles sont définies par la fonction `define()` et utilisées **sans $** !

```
define("REP_TELECHARGEMENT", "Telechargement/");
define("TAILLE_MAXI_FICHIER", 1000000);
echo TAILLE_MAXI_FICHIER;
```


Le type tableau associatif I

- le type tableau associatif (array) est constamment utilisé, il faut donc bien comprendre son fonctionnement
- Un tableau PHP est une association (Map) de couples (clé, valeur)
- Les clés sont uniques et sont des entiers ou des chaînes
- Les valeurs sont quelconques (type et valeur)
- Si l'on a besoin d'un tableau à la C indicé par des entiers débutant à 0, il faut considérer ces indices comme des clés.
- Lorsqu'on ajoute des valeurs sans clé dans le tableau, la clé est calculée comme un entier immédiatement supérieur à la plus grande clé entière du tableau !
- **Attention**, l'ordre chronologique des ajouts est conservée lorsqu'on parcourt (`foreach`) le tableau.
- Création d'un tableau vide : `$t=array();` ou `$t2=[];`

Le type tableau associatif II

- Ajout d'une valeur en fin de tableau : `$t[]="push me !";`

Exemple (tableau.php)

```
<html><head><title>tableaux php</title></head><body>
<?php
$tab=array("un"=>123, 2=>"deux", "abc"=>12.3);
$tab[]="toto";$tab[1]="un";
echo "taille tab : ".count($tab)."<br/>\n";
foreach($tab as $c=>$v){echo "$c => $v <br/>\n";}
?>
</body></html>
```


Le type tableau associatif III

Exemple (Affichage de tableau.php)

```
<html><head><title>tableaux php</title></head><body>
taille tab : 5<br/>
un => 123 <br/>
2 => deux <br/>
abc => 12.3 <br/>
3 => toto <br/>
1 => un <br/>
</body></html>
```

Quelques fonctions utiles sur les tableaux :
Lorsque l'on veut insérer dans une chaîne, une valeur présente dans un tableau, il faut utiliser les accolades :
`echo "bonjour {$tab['michel'][5]}";`

Expressions et Opérateurs I

Les opérateurs ont une priorité ("precedence") et une associativité. Voici une liste non exhaustive avec priorités décroissantes :

- ++, -- inc et déc : `++$x;$y--;`
- unaires ~ (not binaire), - (moins arithmétique), ! (not logique) ;
- arithmét. et concat. (*,/,%) puis (+, -, .) (point de concaténation) ;
- décalages (<<,>>)
- comparaison (<,<=,>=,>) puis (==, !=, ===, !==) (comparaison typée). ATTENTION : fonctionne avec les chaînes mais attention aux chaînes numériques : `"1"==" 01.0"` mais `"1"!==" 01.0"`
- bit à bit (&, ^, |)
- logiques (&&, ||)
- $i==$j ? 1 : 2 expression conditionnelle

Expressions et Opérateurs II

- affectation (=,.=,+=,-=,*=,/=,%=)
- logiques (and, xor, or) moins prioritaire que les affectations, utilisé après les appels risqués :
  `$x=danger() or die("erreur dans la fon danger !");)`

Exemple (comparaison.php)

```
<html><head><title>comparaison</title></head><body>
<?php
if (0=="" && null==false && $tab==0 && "2"=="02"){
  echo "BIZARRE";
}
?>
</body></html>
```

affiche : BIZARRE

Expressions et Opérateurs III

Exemple (la fonction strpos())

La fonction `strpos($chaine, $facteur)` retourne false si le facteur est absent, la position comprise entre 0 et `strlen($chaine)-1` si le facteur est trouvé. Le test `if(strpos('abc','ab')){...` est incorrect car `false==0` ! Il faut donc absolument faire le test suivant :
`if (strpos('abc','ab') !== false) {....`

Certaines particularités historiques sont encore présentes :

- $$s valeur de la variable dont le nom est dans s ; on ne peut pas tripler le $!
- eval("...") évaluation d'une chaîne : TRES dangereux !
- commande syst. `` `ls` ``

Expressions et Opérateurs IV

L'affect. de chaîne longue (heredoc) est possible :

```
$s=<<<FIN
bla bla..
etc
FIN; // en début de ligne et avec un ";"
```

Structures de contrôle I

Les coupures de lignes sont traitées comme les espaces. Les structures de contrôle classique à la C existent en PHP :

alternative `if (expr) {instons1} else {instons2}`

choix `if (e1) {ins1} elseif (e2) {ins2} else {ins3}`

choix `switch ($i) {case e0: instons0; break; case e1: instons1; break; default: instons; }`

répétitives
- `while (e) {instons}`
- `do { instons } while (e)`
- `for (exp-init; exp-cond; exp-itér) { instons }`
- ruptures possibles : `break;`, `continue;`

parcours de tableau
- `foreach($tab as $val) { instons utilisant $val}`

Structures de contrôle II

- `foreach($tab as $cle => $val) { instons utilisant $cle et/ou $val }`

inclusion `include 'monfic.php';` inclusion de fichier

inclusion unique `include_once 'monfic.php';` inclusion au plus une fois !

require, require_once même effet que include sauf qu'en cas d'inexistence du fichier requis, une erreur fatale est envoyée (seulement warning pour include)

Fonctions I

- Un nombre impressionnant de fonctions prédéfinies !
- Passage des paramètres scalaires ou tableaux **par valeur** (par référence si précédé de & dans la déclaration)
- Les arguments scalaires peuvent avoir une valeur par défaut (optionnels) et être en nombre variable
- Les arguments ayant une valeur par défaut doivent être à droite de ceux n'en ayant pas
- Les valeurs de retour peuvent être scalaire ou tableaux
- On peut supprimer l'affichage des erreurs produites par l'appel à une fonction f par : `@f($i)`

Fonctions sur les tableaux I

int count($tab) taille du tableau

int array_push($pile,$elem) empile à la fin et ret la taille

mixed array_pop($pile) dépile le sommet

int array_unshift($fifo,$prem) ajoute au début

mixed array_shift($fifo) retire le premier

array array_values($asso) resp. `array_keys`

void shuffle($tab) mélange les valeurs

Fonctions sur les tris de tableau I

void sort($tab) tri croissant (décroissant avec rsort) selon les valeurs

void ksort($asso) tri croissant selon les clés (krsort)

void asort($asso) tri croissant selon les valeurs (arsort)

void usort($tab, moncmp) tri croissant selon la fon de cmp définie par l'utilisateur

uksort, uasort tri utilisateur pour les asso

Fonctions sur les chaînes I

De très nombreuse fonctions dont voici les principales. $s est supposée être une chaîne.

int strlen($s) taille

int strcmp($s1,$s2) comparaison -1, 0, 1

string substr($s, 2, 10) sous-chaîne à partir de 2, de taille 10

string strstr($s,$facteur) ret tout s à partir de la 1ère occurrence de facteur

array split(' ;',$s,10) ret un tableau des 10 (maxi) premiers champs séparés par des ;

string join(' ;',$tab) ret la chaîne constituée des valeurs séparées par ;

string trim($s) retire les blancs de début et de fin de chaîne. Blancs : \n,\r,\t,\v,\0,espace

string chop($s) supprime les blancs de fin de chaîne

Fonctions sur les chaînes II

string ltrim ($s) supprime les blancs de début

string strtolower($s) met en minuscules (resp. strtoupper)

string nl2br($s) remplace les \n par des

Fonctions sur les expressions régulières I

correspondance `int preg_match('/^[^.]*\.(.*)$/',$s,$tabr)` met dans tabr la partie à droite d'un point dans s à partir de l'indice 1. Retourne TRUE si trouvé au moins une.

remplacement

`string preg_replace($patterns, $replacements, $string);` retourne une chaîne obtenue par remplacement dans la chaîne string des facteurs correspondant aux patterns par les replacements.

```
php > echo preg_replace("/tu/", "il","tu manges");
il manges
```

Entrées/Sorties I

De nombreuses fonctions de gestion du système de fichier à la C

echo string, string ... ; affiche plusieurs arguments **chaînes**

print(string) ; affiche un arg. (**classé** dans les fons chaînes)

$desc=fopen("fic.txt","r+") ; ouverture en lecture et écriture. Modes (r, w (création ou raz), w+ , a (append), a+).

$d=fopen("c : \\data\\info.txt","r"); Windows

$d=fopen("ftp ://user :password@example.com/", **"w") ;** ouverture de session ftp

fclose($desc) fermeture

bool feof($desc) test la fin

$ligne=fgets($desc, 4096) lecture de la ligne (maxi 4096 octets)

$ligne=readline("un entier SVP") ; ne fonctionne pas sur le Web !

$car=fgetc($desc) ; lecture d'un car ;

fputs($desc,"chaine") ; écriture d'une chaîne ;

Fonctions utilisateur I

La référence en avant est possible.

définition `function f($arg0, $arg1="toto"){instons; return array(1, 4);}`

pointeur de fonction `$pf='f';list($i,$j)=$pf(1,2);`

Diverses fonctions prédéfinies I

echo exp1, exp2, ... ; affiche les expressions ; ATTENTION, echo n'est pas une fonction mais une structure de contrôle !

print(exp1) affiche une expression ;

var_dump($var) affiche la variable scalaire ou tableau ou objet récursivement (débogage) ;

void exit() termine le script

$l=system("ls") ; exécute une commande

void die ("message ") affiche le message puis termine

void eval("instons PHP") il faut échapper les caractères spéciaux : `eval('$' . f($i) . '= "toto";')` ; Ne pas oublier le ";"

Tableaux super-globaux I

Des tableaux super-globaux sont prédéfinis et permettent d'accéder à diverses informations.

$GLOBALS Contient une référence sur chaque variable qui est en fait disponible dans l'environnement d'exécution global. Les clés de ce tableau sont les noms des variables globales. Ainsi, `$GLOBALS['x']` permet de connaître ou d'affecter la variable globale `$x`.

$_SERVER Les variables fournies par le serveur web, ou bien directement liées à l'environnement d'exécution du script courant, notamment `$_SERVER['PHP_SELF']` qui est le chemin du script en cours d'exécution par rapport à la racine web (sans les paramètres GET) ;

$_GET Les variables fournies au script via la chaîne de requête URL.

Tableaux super-globaux II

$_POST Les variables fournies par le protocole HTTP en méthode POST.

$_COOKIE Les variables fournies par le protocole HTTP, dans les cookies.

$_FILES Les variables fournies par le protocole HTTP, suite à un téléchargement de fichier.

$_ENV Les variables fournies par l'environnement

$_REQUEST Les variables fournies au script par n'importe quel mécanisme d'entrée dans un certain ordre et qui ne doivent recevoir qu'une confiance limitée. Note : lorsque vous exécutez un script en ligne de commande, cette variable ne va pas inclure les variables argv et argc. Elles seront présentes dans la variable $_SERVER. php.ini : `variables_order = "EGPCS"`

Plan

Gestion de formulaire I

Les champs de **formulaire HTML** sont accessibles via les tableaux superglobaux $_GET ou $_POST selon la **méthode** utilisée par le formulaire. Par exemple :

```
<form name='F' method='get'>
<input type='text' name='nom'>
<select multiple name="biere[]">
 <option value="blonde">je préfère la bière blonde</option>
 <option value="brune">je préfère la bière brune</option>
</select>
<input type='submit' name='bouton' value='ok'/>
</form>
<?php if (!empty($_GET['biere']))
        var_dump($_GET['biere']);
?>
```

Après saisie et validation, l'url suivante est requise :

Gestion de formulaire II

`essai.php?nom=toto&biere[]=blonde&biere[]=brune&bouton=ok`

L'affichage suivant aura lieu :

```
Array ( [0] => blonde [1] => brune )
```

Remarques

- si l'attribut `action` n'est pas défini, l'url courante est à nouveau requise (avec les nouveaux paramètres !) ;
- les clés du tableau $_GET sont les valeurs des attributs HTML `name` des champs de saisie (input) ;

Gestion de formulaire III

- si la méthode est `get`, les paramètres passés par le formulaire lors de la soumission sont visibles dans la "query string" : la chaîne de requête est la partie d'URL commençant par un point d'interrogation et finissant à la fin de l'URL. Le séparateur des paramètres est le `&` et chaque paramètre est de la forme : `name=value`. L'URL est visible et modifiable dans la barre d'adresse du navigateur ce qui permet le débogage mais aussi l'accès visuel (password) !
- si la méthode est `post`, les paramètres sont passés dans la requête HTTP sans être visibles dans la barre d'adresse (sécurité).
- lorsqu'on souhaite que PHP recueille une liste de valeurs associée à un nom (i.e. `biere`), il faut suffixer ce nom par `[]` **dans le HTML**, sinon c'est la dernière valeur uniquement qui sera prise en compte ! Ceci est valable pour les listes à sélection multiple (select) mais aussi pour les cases à cocher. Dans le code PHP, on ne met pas les crochets (`$_GET['biere']`) !

Gestion de formulaire IV

Exercice (Multiplication)

Ecrire :

- une page HTML "Multiplication" contenant un formulaire ayant deux champs de saisie numériques x et y et un bouton `Multiplier !` ;
- un script php appelé par ce formulaire et qui affiche le résultat de la multiplication $x * y$;

Gestion de formulaire V

Solution (multiplication.html)

```
<html><head><title>Multiplication</title></head><body>
<h1>Multiplication</h1>
<form action="multiplication.php" method="get">
X<input type="number" name="x" size="10"><br>
Y<input type="number" name="y" size="10"><br>
<input type="submit" value="Multiplier !" name="mult">
</form>
</body></html>
```


Gestion de formulaire VI

Solution (multiplication.php)

```
<html><head><title>Multiplication</title></head><body>
<h1>Multiplication</h1>
<?php
if (isset($_GET['x']) && isset($_GET['y'])){
  echo "Résultat {$_GET['x']} * {$_GET['y']} = ",
  ↪  $_GET['x']*$_GET['y'], " !";
}
?>
</body></html>
```

Exercice (Unique fichier)

Réécrire le même exercice en un seul fichier !

Gestion de formulaire VII

Solution (mult.php)

```
<html><head><title>Multiplication</title></head><body>
<h1>Multiplication</h1>
<?php
if (isset($_GET['mult']) && isset($_GET['x']) &&
↪  isset($_GET['y'])){
  echo "Résultat {$_GET['x']} * {$_GET['y']} =
  ↪  ".$_GET['x']*$_GET['y']." !<br/>";
  echo "<hr/> Nouvelle Multiplication :<br/>";
}
?>
<form action="" method="get">
<label for="x">X</label><input type="number" name="x" id="x"
↪  size="10"><br>
<label for="y">Y</label><input type="number" name="y" id="y"
↪  size="10"><br>
<input type="submit" value="Multiplier !" name="mult">
</form>
</body></html>
```


Plan

Le modèle objet depuis PHP5 I

- Depuis PHP 5, modèle objet réécrit
- proche de celui du C++
- Une classe peut contenir ses propres constantes, variables (appelées "propriétés" ou "attributs"), et fonctions (appelées "méthodes")
- Une classe est introduite par le mot-clé `class` suivi de son nom
- Le constructeur se nomme `__construct()` et le destructeur `__destruct()`
- Par défaut, l'appel au constructeur de la classe parente n'est pas fait ! Pour le faire : (`parent::__construct()`)
- Les méthode magiques `__toString()` et `__clone()` sont également très utiles

Le modèle objet depuis PHP5 II

- L'affectation et le passage de paramètre objet est effectué **par référence** : il n'y pas de duplication de l'objet mais la variable affectée ou le paramètre formel contient l'adresse de l'objet

Le modèle objet depuis PHP5 III

Exemple (Une classe Point)

```
class Point{
  public static $nbPoints=0;
  public $x,$y;
  public function __construct($px=0,$py=0){ // constructeur
    $this->x=$px;$this->y=$py;
    self::$nbPoints+=1;
  }
  public function getxy(){
    return array($this->x, $this->y);
  }
}
$p1=new Point(); // (0,0)
$p2=new Point(3,4);
list($a,$b)=$p2->getxy();
```


Propriétés, constructeur, méthodes, ... I

Propriétés

- Les attributs d'instance ou propriétés doivent être déclarés à l'aide de `public`, `protected` ou `private`
- L'ancien déclarateur `var` est obsolète et est remplacé par `public`
- Une déclaration initialisante est permise en utilisant une valeur constante
- Dans les méthodes d'instance, ces propriétés sont accessibles via la notation "fléchée" `$this->x`
- Une propriété statique (attribut de classe) peut être définie grâce au mot-clé `static`. L'accès à cette propriété à l'intérieur d'une classe se fait par `self::$nbPoints`.

Propriétés, constructeur, méthodes, ... II

Constructeur et destructeur

- Les constructeurs d'une classe se nomment `__construct(...)` et **le** destructeur `__destruct()`
- Par défaut, l'appel au constructeur de la classe parente n'est pas fait ; pour le faire : (`parent::__construct()`)
- En l'absence de constructeur explicite, toute classe admet un constructeur par défaut qui ne fait rien.

Méthodes

Les méthodes des classes peuvent être définies en tant que publiques, privées ou protégées. Les méthodes sans déclaration seront automatiquement définies comme étant publiques.

Propriétés, constructeur, méthodes, ... III

Héritage (extends), redéfinition, surcharge

- Une classe peut hériter des méthodes et des attributs d'**une** autre classe en utilisant le mot-clé extends dans la déclaration
- Il n'y a pas d'héritage multiple.
- Les méthodes et attributs hérités peuvent être redéfinis en les redéclarant avec le même nom que dans la classe parente (sauf si la classe parente a défini la méthode comme `final`) (Java)
- Il est possible d'accéder à une méthode ou un membre parent avec l'opérateur `parent::`
- Lors de la redéfinition de méthodes, la signature doit rester la même sinon PHP génèrera une erreur de niveau `E_STRICT` (**pas de surcharge**)
- Ceci ne s'applique pas au constructeur, qui accepte la surcharge (avec des paramètres différents)

Interfaces et classes abstraites I

Interfaces

- Une interface permet de spécifier quelles méthodes une classe doit implémenter
- Les interfaces sont définies en utilisant le mot-clé `interface`, de la même façon qu'une classe standard mais sans aucun contenu de méthode
- Toutes les méthodes déclarées dans une interface doivent être publiques.
- Pour implémenter une interface, l'opérateur `implements` est utilisé. Toutes les méthodes de l'interface doivent être implémentées dans une classe ; si ce n'est pas le cas, une erreur fatale sera émise
- Les classes peuvent implémenter plus d'une interface en séparant chaque interface par une virgule (extends one, implemments many)

Interfaces et classes abstraites II

Classe abstraite

- On ne peut créer une instance d'une classe définie comme `abstract`
- Toutes les classes contenant au moins une méthode abstraite doivent également être abstraites
- Pour définir une méthode abstraite, il faut simplement déclarer la signature de la méthode (précédée de `abstract` et ne fournir aucune implémentation
- Lors de l'héritage d'une classe abstraite, toutes les méthodes marquées comme abstraites dans la déclaration de la classe parente doivent être définies par l'enfant
- de plus, ces méthodes doivent être définies avec la même visibilité, ou une visibilité moins restreinte
- Par exemple, si la méthode abstraite est définie comme protégée, l'implémentation de la fonction doit être définie comme protégée ou publique, mais non privée.

Méthodes magiques et clônage I

Les méthodes :
`__construct`, `__destruct`, `__toString`, `__clone`, `__get`, `__set`, `__call`, `...` sont magiques en PHP. Vous ne pouvez pas utiliser ces noms de méthode dans vos classes, sauf si vous voulez implémenter le comportement associé à ces méthodes magiques.

toString : détermine comment l'objet doit réagir lorsqu'il est traité comme une chaîne de caractères (`echo` ou `print`) ;

clone : une fois le clonage effectué, si une méthode `__clone()` est définie, celle-ci sera appelée sur le nouvel objet ;

get, set : `void __set(string $name, mixed $value)` sera appelé lorsque l'on essaie d'affecter une valeur à un attribut inaccessible. Cela permet d'ajouter dynamiquement de nouveaux attributs (prototype).

Méthodes magiques et clônage II

call, callStatic : appelé lorsque l'on essaie d'appeler une méthode inaccessible. Cela permet d'ajouter dynamiquement de nouvelles méthodes (prototype).

Clônage

Lorsqu'un objet est cloné, PHP effectue une copie **superficielle** de toutes les propriétés de l'objet. Toutes les propriétés qui sont des références à d'autres variables demeureront des références. L'opérateur `clone` est utilisé comme suit :

```
$copie = clone $objet;
```

Méthodes magiques et clônage III

Comparaison d'objets

Lors de l'utilisation de l'opérateur de comparaison ==, deux objets sont égaux s'ils ont les mêmes attributs et valeurs, et qu'ils sont des instances de la même classe. Lors de l'utilisation de l'opérateur d'identité ===, les objets sont identiques uniquement s'ils font référence à la même instance de la même classe.

Méthodes magiques et clônage IV

L'interface iterator

Cette interface permet d'utiliser la structure de contrôle `foreach` afin de parcourir tous les élément dans une itération. Voici la liste des méthodes à implémenter :

```
Iterator extends Traversable {
/* Methods */
abstract public mixed current ( void )
abstract public scalar key ( void )
abstract public void next ( void )
abstract public void rewind ( void )
abstract public boolean valid ( void )
}
```


Espaces de nom I

Comme dans les autres langages à objet, les espaces de nom sont utilisés :

- afin d'éviter les collisions entre des noms utilisés dans votre code mais également dans des fichiers inclus ;
- afin d'éviter de manipuler des noms très longs pour éviter les collisions.

Les noms d'espace ne sont pas sensibles à la casse et doivent commencer par une lettre. Ils concernent : classes, interfaces, fonctions et constantes. Ils peuvent être organisés en hiérarchie en utilisant le séparateur anti-slash \

Espaces de nom II

Exemple (Déclaration d'un espace de noms)

```
<?php
namespace MonProjet\MaBD; // DOIT ABSOLUMENT DEMARRER le fichier
const TAILLE = 100;
class Connexion { /* ... */ }
function connecte() { /* ... */  }
?>
```

On peut utiliser un bloc pour encadrer les définitions comprises dans l'espace de nom. Un espace de nom global préexiste, il n'a pas de nom.

Espaces de nom III

Exemple (Utilisation des espaces de noms (edn))

On peut accéder aux noms d'un espace de 3 façons à rapprocher de la façon dont on référence un fichier dans un système Unix :

- nom sans séparateur (`toto`) : résolu en `ednCourant\toto`
- nom avec des séparateurs mais pas au début (`titi\toto`) est résolu en `ednCourant\titi\toto`
- nom commençant par un antislash (`\MonProjet\MaBd\TAILLE`) est absolu

Attention, dans un edn, l'accès aux fonctions PHP globales est réalisé en préfixant ces noms globaux par antislash.

```
<?php
namespace MonProjet\MaBD; // DOIT DEMARRER le fichier
function connecte($n) { if (\strlen($n)<=10) ...  } // strlen
?>
```


Importation et aliassage de noms I

Exemple (Importation et alias avec l'opérateur use)

Après avoir défini des noms dans des espaces de nom, on peut les utiliser en les important (use) et éventuellemnt en les aliassant :

```
use MonProjet\Bd\Connexion as MaCo;

// les 2 lignes suivantes sont équivalentes :
use MonProjet\Bd\Connexion as Connexion;
use MonProjet\Bd\Connexion;

// importation d'une classe globale
use ArrayObject;
```


Plan

Session I

- HTTP n'étant pas un protocole orienté connexion, chaque nouvelle requête semble être la première
- Depuis PHP4, Les **sessions** PHP permettent de conserver de l'information **du côté serveur** sur la suite de requêtes émises par le même client (navigateur)
- Les variables enregistrées dans le tableau super-global `$_SESSION` peuvent être de type **quelconque**
- Pour les variables de type tableau ou objet, elles seront sérialisées et désérialisées automatiquement dans le fichier de session côté serveur (copie profonde)
- Attention cependant à définir la classe d'un objet enregistré en session **avant** l'appel à `session_start()`

La communication des identifiants de sessions est réalisée :

Session II

- soit par cookie ce qui est le plus simple ;
- soit par URL de manière quasi-transparente pour le programmeur.

Par défaut, la durée de vie du cookie de session (0) est égale à la durée de vie du navigateur.

bool session_start() démarre une session ; doit être réalisé en tout **début de script** avant tout en-tête !

string session_id() retourne l'identifiant de session qui est une suite de chiffres hexadécimaux.

$_SESSION['z']="contenu" ; ajoute une variable de session ;

echo $_SESSION['z'] ; affiche le contenu d'une variable de session ;

bool session_is_registered("mavar") teste si $mavar a été sauvé dans la session ;

bool session_unregister("x") supprime la variable de session x ;

Session III

$_SESSION=array() ; réinitialise la session !

bool session_destroy() supprime toutes les données de la session ; le cookie de session sera supprimé dès la prochaine page.

bool session_register("x", "y", ...) démarre une session si pas encore fait et y enregistre les variables $x et $y ;

Session IV

Session sans Coockie

Afin de gérer les sessions de manière transparente, le script PHP doit tenir compte du fait que le navigateur client peut accepter ou refuser les cookies. Pour ce faire, il suffit d'indiquer dans chaque référence interne du site (href d'ancre, action de formulaire, ...), la constante SID comme paramètre de l'URL afin de transmettre l'identifiant de session. En effet, la constante SID a comme valeur :

- '' chaîne vide lorsque le navigateur accepte les cookies ;
- 'PHPSESSID=0c92dbd...51ff2' lorsque le navigateur ne les accepte pas ;

Ainsi dans un formulaire, l'attribut action devra avoir la forme suivante :

```
action="<?php echo "{$_SERVER['PHP_SELF']}".(strlen(SID)?'?'.SID:''); ?>"
```

L'exemple suivant illustre les sessions avec un historique de multiplications.

Multiplication avec mémorisation des résultats dans une session I

```
<?php session_start(); ?>
<!doctype html><html lang="fr"><head><meta charset="utf-8" />
<title>Multiplication et session</title></head><body>

<h1>Multiplication et session</h1>
<?php
if(isset($_SESSION['historique'])){ // var_dump($_SESSION);
?><h2>Historique des multiplications</h2>
<?php
  foreach($_SESSION['historique'] as $tab){ // [['x'=>2, 'y'=>3,
  ↪  'r'=>6], ...]
    echo "{$tab['x']} * {$tab['y']} = {$tab['r']}<br/>\n";
  }
  echo "<hr/>\n";                    // pour délimiter l'historique
} else {                            // début de session
```

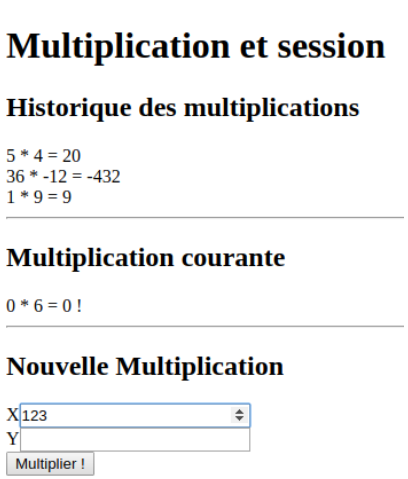
Multiplication avec mémorisation des résultats dans une session II

```
  $_SESSION['historique']=array(); // init tableau de tableau
}

if (isset($_GET['mult'])){                  // nouvelle mult
?><h2>Multiplication courante</h2><?php
  echo " {$_GET['x']} * {$_GET['y']} = ".$_GET['x']*$_GET['y'].
    " !<br/>";
  $tab=array('x'=>$_GET['x'], 'y'=>$_GET['y'],
  ↪  'r'=>$_GET['x']*$_GET['y']);

  $_SESSION['historique'][]=$tab;               // ajout dans
  ↪  l'historique
  echo "<hr/>";
}
?>
```


Multiplication avec mémorisation des résultats dans une session III

```
<h2>Nouvelle Multiplication</h2>
<form action="<?php echo
↪  "{$_SERVER['PHP_SELF']}".(strlen(SID)?'?'.SID:''); ?>"
     method="get">
X<input type="number" name="x" size="10"><br>
Y<input type="number" name="y" size="10"><br>
<input type="submit" value="Multiplier !" name="mult">
</form>
</body></html>
```


Multiplication avec mémorisation des résultats dans une session IV



Multiplication avec mémorisation des résultats dans une session V

Si les cookies ne sont pas acceptés dans `php.ini`, voici le contenu de la barre d'adresse :

`.../multsession.php?PHPSESSID=1d5192e149789a9a52b10f948bbf6843`

Que mettre comme variable de session ? I

- Si l'on utilise une BD pour stocker les informations, il est souhaitable de ne mettre en session qu'un identifiant afin de récupérer à chaque fois les informations courantes
- Si l'on stocke un objet, il faut comprendre que tous les objets référencés par cet objet sont représentés dans la variable de session (faire un `var_dump`)
- Aussi, les mises-à-jour dans la BD d'objets référencés peuvent devenir invisibles depuis l'objet de session !
- Si l'on n'utilise pas de BD, alors toute l'information étant dans l'objet de session, il conviendra de le mettre à jour.

Session et Objets I

- Si l'on souhaite mettre un objet en variable de session, il faut que la définition de la classe soit chargée avant le début de session (`session_start()`)
- Ceci peut se faire par un `include` classique ou bien par l'auto-chargement (`autoload`) qui permet de charger le fichier définissant la classe à la demande

```
// classLoader.php
function __autoload($class_name) {
    include 'MesClasses/' . $class_name . '.php';
}

// testClass.php dans le rép MesClasses
<?php
class testClass {
  private $prop1;
```


Session et Objets II

```
  function __construct($propValue) {
    $this->prop1 = $propValue;
  }

  function showProp() {
    return $this->prop1;
  }
}
?>

// page1.php
<?php
require_once('classLoader.php');
session_start();
$_SESSION['testObj'] = new testClass('foo');
echo '<a href="page2.php">Go to page 2</a>';
```


Session et Objets III

```
?>

// page2.php
<?php
require_once('classLoader.php');
session_start();
echo $_SESSION['testObj']->showProp(); // displays foo
?>
```

Plan

Bases de données avec PDO I

- Les fonctions PHP d'accès aux Systèmes de Gestion de Bases de Données sont nombreuses
- Il existait des APIs spécialisées natives pour des sgbds tels que MySQLi ou Oracle OCI8 ou PostgreSQL
- depuis PHP 5.1, l'extension PHP Data Objects (PDO) est installée par défaut et **doit être utilisée** car elle est indépendantes du sgbd cible
- Cette interface d'abstraction à l'accès de données signifie que vous utilisez les mêmes fonctions pour exécuter des requêtes ou récupérer les données quelque soit la base de données utilisée (portabilité) (MySQL, Oracle, ...)
- De plus, les requêtes préparées permettent d'évicter les attaques par injection de code SQL

Connexions et gestionnaire de connexion I

Les connexions sont établies en créant des instances de la classe de base (PDO) quel que soit le pilote (driver) de SGBD utilisé. Le constructeur nécessite un paramètre pour spécifier la source de la base de données (Data Source Name) et optionnellement, le nom d'utilisateur et le mot de passe (s'il y en a un). Le DSN est une chaîne de caractères composée :

- d'un préfixe indiquant le gestionnaire, par exemple `mysql:` ou `pgsql:` ou ... ;
- d'une suite de paramètres séparés par des **point-virgules** de la forme `param1=val1;param2=val2;...`. Ces paramètres peuvent être : `host, port, dbname, user, password, charset`

Remarquons que le couple (nom d'utilisateur, mot de passe) peut être fourni dans le DSN (prioritaire) ou comme paramètres du constructeur.

Connexions et gestionnaire de connexion II

```php
<?php // Connexion à MySQL
$user="mmeynard";$pass="XXX";
try{
  $dbh = new PDO('mysql:host=mysql.etu.umontpellier.fr;
  ↪  dbname=p00000010402; charset=UTF8', $user, $pass,
  ↪  array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,));
} catch(PDOException $e){
  echo $e->getMessage();
  die("Connexion impossible !");
}
?>
```

S'il y a des erreurs de connexion, un objet PDOException est lancé. Vous pouvez attraper cette exception si vous voulez gérer cette erreur, ou laisser le gestionnaire global d'exception la traiter via la fonction : `set_exception_handler()`

Connexions et gestionnaire de connexion III

```php
<?php // Connexion à PostgreSQL avec gestion des erreurs
try {
  $dbh = new PDO("pgsql:dbname=$dbname; host=$host;
  ↪  username=$username; password=$password");
  foreach($dbh->query('SELECT * from etudiant') as $row) {
    var_dump($row);
  }
  $dbh = null;    // déconnexion
} catch (PDOException $e) {
  print "Erreur !: " . $e->getMessage() . "<br/>";
  die();
}
?>
```

Pour fermer la connexion, il suffit de déréférencer l'objet PDO en affectant la variable à nul : `$dbh=null;`

Connexion persistante I

- Les connexions **persistantes** ne sont pas fermées à la fin du script, mais sont mises en cache
- puis réutilisées lorsqu'un autre script demande une connexion en utilisant les mêmes paramètres.
- Le cache des connexions persistantes permet d'éviter d'établir une nouvelle connexion à chaque fois qu'un script doit accéder à une base de données
- améliore la vitesse de l'application web

```php
<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass,
 array(PDO::ATTR_PERSISTENT => true));
?>
```


Requêtes uniques I

Pour des requêtes individuelles, on peut utiliser la méthode `query` pour une consultation ou la méthode `exec` pour une modification. Il faut noter que l'objet `PDOStatement` retourné par `query` est `Traversable`, c'est-à-dire qu'on peut itérer directement dessus avec `foreach` (`fetch` automatique).

Requêtes uniques II

Exemple (Requête unique)

```php
<?php
try {
   $sql =  'SELECT nom, ue FROM utilisateur u, etudiant e WHERE
   u.id=e.id ORDER BY ue, nom';
   foreach($dbh->query($sql) as $row) {
        print($row['nom'] . "\t" . $row['ue'] . "<br />");
   }
   $nb=$dbh->exec("DELETE FROM utilisateur WHERE id NOT IN
   ↪  (SELECT id FROM etudiant)");
   print("Nb de suppressions : " . $nb);
?>
```

Requêtes préparées (PDOStatement) I

Une requête préparée est une sorte de modèle compilé pour la(es) requête(s) SQL que vous voulez exécuter. Les requêtes préparées offrent deux fonctionnalités essentielles :

- La requête ne doit être analysée (ou préparée) qu'une seule fois, mais peut être exécutée plusieurs fois avec des paramètres identiques ou différents. L'optimisation de son plan d'exécution permet à la requête préparée d'utiliser moins de ressources et de s'exécuter plus rapidement.
- Les paramètres pour préparer les requêtes (`:nomparam`) n'ont pas besoin d'être entre guillemets ; le driver gère l'association avec une valeur grâce à l'association (liaison) **bind**. Si votre application utilise exclusivement les requêtes préparées, vous pouvez être sûr qu'aucune injection SQL n'est possible.

Requêtes préparées (PDOStatement) II

PDO émule les requêtes préparées pour les drivers qui ne les supportent pas. Ceci assure de pouvoir utiliser la même technique pour accéder aux données, sans se soucier des capacités de la base de données. **Attention, les paramètres ne peuvent pas être utilisés pour remplacer des noms SQL** (mot-clé, identificateurs de fonction ou de colonne ! Par exemple, `order by :col` est interdit. Ils ne peuvent que remplacer des valeurs (chaines, entier, ...).

Exemple d'insertions répétées en utilisant les requêtes préparées I

Cet exemple effectue une requête INSERT en y substituant un nom et une valeur pour les **marqueurs nommés**.

```php
<?php
$stmt = $dbh->prepare("INSERT INTO REPERTOIRE (name, value)
↪  VALUES (:name, :value)");
$stmt->bindParam(':name', $name);   // association paramètre
↪  marqueur nommé
$stmt->bindParam(':value', $value); // avec variable PHP

// insertion de deux lignes
$name = 'Dupont';
$value = 0612345678;
$stmt->execute();

$name = 'Durand';
```


Exemple d'insertions répétées en utilisant les requêtes préparées II

```php
$value = 0468901234;
$stmt->execute();
?>
```

Utilisation de marqueur anonyme <?> I

```php
<?php
<$stmt = $dbh->prepare("INSERT INTO REPERTOIRE (name, value)
↪  VALUES (?,?)");
$stmt->bindParam(1, $name);   // association paramètre marqueur
↪  anonyme
$stmt->bindParam(2, $value);  // avec variable PHP

// insertion
$name = 'Martin';
$value = 0612435678;
$stmt->execute();
?>
```


Utilisation de marqueur anonyme <?> II

Exemple (Consultation paramétrée avec marqueur anonyme)

```php
<?php
$stmt = $dbh->prepare("SELECT * FROM REPERTOIRE where name = ? or
↪  value = ? ");
if ($stmt->execute(array($_POST['nom'],$_POST['numero']))){ //
↪  liaison implicite
  while ($row = $stmt->fetch()) {
    print($row['name']." : ".$row['value']);
  }
}
?>
```

La récupération (fetch) des lignes résultant d'une consultation est réalisée séquentiellement selon la valeur du paramètre optionel de la méthode fetch. Par défaut, la valeur `PDO::FETCH_BOTH` permet de retourner la ligne

Utilisation de marqueur anonyme <?> III

suivante en tant que tableau indexé par le nom et le numéro de la colonne (0 à n-1). Mais on peut également :

- récupérer un objet dont les attributs correspondront aux colonnes ;
- se déplacer selon une orientation (vers le bas ou le haut) ;
- indiquer un déplacement différent de 1 dans la séquence de récupération.

Transaction I

- Les transactions offrent 4 fonctionnalités majeures : Atomicité, Consistance, Isolation et Durabilité (ACID)
- Le travail d'une transaction (suite de requêtes) peut être annulé (abort) à votre demande ou bien validé (commit)
- Malheureusement, toutes les bases de données ne supportent pas les transactions, donc, PDO doit s'exécuter en mode "autocommit" lorsque vous ouvrez pour la première fois la connexion
- Le mode "autocommit" signifie que toutes les requêtes que vous exécutez ont leurs transactions implicites, si la base de données le supporte ou aucune transaction si la base de données ne les supporte pas
- Si vous avez besoin d'une transaction, vous devez utiliser la méthode `PDO::beginTransaction()` pour l'initialiser

Transaction II

- Si le driver utilisé ne supporte pas les transactions, une exception PDO sera lancée
- Une fois que vous êtes dans une transaction, vous devez utiliser la fonction `PDO::commit()` ou la fonction `PDO::rollBack()` pour la terminer
- **Attention**, PDO ne vérifie la possibilité d'utiliser des transactions qu'au niveau du pilote
- Si certaines conditions à l'exécution empêchent les transactions de fonctionner, `PDO::beginTransaction()` retournera tout de même TRUE sans erreur si le serveur accepte de démarrer une transaction
- Ce sera le cas en utilisant le pilote MySQL sur des tables au format MyISAM car ce dernier ne supporte pas les transactions (utiliser plutôt InnoDB)

Transaction III

- Lorsque le script se termine ou lorsque la connexion est sur le point de se fermer, si vous avez une transaction en cours, PDO l'annulera automatiquement. Ceci est une mesure de sécurité afin de garantir la consistance de vos données dans le cas où le script se termine d'une façon inattendue. Si vous ne validez pas explicitement la transaction, alors, on présume que quelque chose s'est mal passé et l'annulation de la transaction intervient afin de garantir la sécurité de vos données.

```php
<?php
try {
  $dbh = new PDO('mysql:host=localhost;dbname=test', $user,
  ↪  $pass,
      array(PDO::ATTR_PERSISTENT => true));
  echo "Connecté\n";
} catch (Exception $e) {
  die("Impossible de se connecter: " . $e->getMessage());
}
```


Transaction IV

```php
try {
  $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  // pour lancer une exception sur chaque erreur
  $dbh->beginTransaction();
  $dbh->exec("insert into utilisateur (id, nom, prenom) values
  ↪  (12, 'Dupont', 'Jean')");
  $dbh->exec("insert into etudiant (id, ue) values (12, 'l3'");
  $dbh->commit();
} catch (Exception $e) {
  $dbh->rollBack();
  echo "Impossible d'ajouter l'étudiant : " . $e->getMessage();
}
?>
```


Synopsis des classes PDO et PDOStatement I

```
 PDO {
public __construct ( string $dsn [, string $username [, string
↪  $password [, array $driver_options ]]] )
public bool beginTransaction ( void )
public bool commit ( void )
public bool rollBack ( void )
public int exec ( string $statement )
public static array getAvailableDrivers ( void )
public bool inTransaction ( void )
public PDOStatement prepare ( string $statement [, array
↪  $driver_options = array() ] )
public PDOStatement query ( string $statement )
...
}
 PDOStatement implements Traversable {
/* Propriétés */
```

Synopsis des classes PDO et PDOStatement II

```
readonly string $queryString;
/* Méthodes */
public bool bindColumn ( mixed $column , mixed &$param [, int
↪  $type [, int $maxlen [, mixed $driverdata ]]] )
public bool bindParam ( mixed $parameter , mixed &$variable [,
↪  int $data_type = PDO::PARAM_STR [, int $length [, mixed
↪  $driver_options ]]] )
public bool bindValue ( mixed $parameter , mixed $value [, int
↪  $data_type = PDO::PARAM_STR ] )
public bool closeCursor ( void )
public int columnCount ( void )
public string errorCode ( void )
public array errorInfo ( void )
public bool execute ([ array $input_parameters ] )
public mixed fetch ([ int $fetch_style [, int $cursor_orientation
↪  = PDO::FETCH_ORI_NEXT [, int $cursor_offset = 0 ]]] )
public array fetchAll ([ int $fetch_style [, mixed
↪  $fetch_argument [, array $ctor_args = array() ]]] )
```


Synopsis des classes PDO et PDOStatement III

```
public string fetchColumn ([ int $column_number = 0 ] )
public mixed fetchObject ([ string $class_name = "stdClass" [,
↪  array $ctor_args ]] )
public int rowCount ( void )
public bool setFetchMode ( int $mode )
...
}
```


Un exemple de TP I

```
<html><head>  <title>Test de PDO et du driver MySQL</title>
    <meta http-equiv="Content-Type" content="text/html;
    ↪  charset=utf-8">
  </head>
  <body>
    <h1>Test de PDO</h1>
<?php
try {
  $dbh = new PDO('mysql:host=venus;dbname=mmeynard', "mmeynard",
  ↪  "XXXX");
  foreach($dbh->query('SELECT * from test') as $row) {
    print_r($row);
  }
  $dbh = null;   // fermeture connexion
} catch (PDOException $e) {
  print "Erreur !: " . $e->getMessage() . "<br/>";
```


Un exemple de TP II

```
  die();
}
?>
  </body>
</html>
```

MySQL et phpMyAdmin I

- phpMyAdmin est un outil d'administration d'une BD MySQL écrit en PHP
- Cet outil est accessible via le web et permet donc d'administrer à distance une BD
- On peut : créer, supprimer, modifier (alter) des tables, interroger, ajouter, supprimer, modifier des lignes, importer des fichiers textes contenant les données ...
- Le site de référence où l'on peut tester en direct est : `http://www.phpmyadmin.net/home_page/index.php`
- MySQL supporte plusieurs moteurs de stockage, qui gère différents types de tables. Les moteurs de tables MySQL peuvent être transactionnels ou non-transactionnels

MySQL et phpMyAdmin II

- Historiquement et par défaut, le moteur de stockage est `MyISAM` : il ne permet ni contraintes d'intégrité référentielles (clés étrangères), ni transactions mais est très rapide
- Le moteur de stockage `InnoDB` est une alternative à privilégier puisqu'elle offre ces fonctionalités indispensables (transactions, références) en plus d'autres.

Plan

Cookies I

- Un cookie est stocké du côté client par le navigateur et contient un couple `nom=valeur` ainsi qu'un domaine, un chemin et une date d'expiration
- Quand le client envoie une requête à une URI d'un domaine et d'un chemin dont il possède des cookies, tous ces cookie sont envoyés au serveur
- Un cookie a une durée de vie (expire) par défaut égale à la durée de vie de la session
- Pour accepter/afficher les cookies sur `Firefox`, utiliser le menu Outils, Option, Vie Privée, Accepter/Afficher les cookies.

Fonctions PHP :

Cookies II

int setcookie('x','Hello !',int expire, string path, `string domain, int secure`) doit être exécutée avant toute chose et permet de positionner la variable $x avec la valeur 'Hello !' ; attention le cookie ne sera renvoyé au serveur qu'à la prochaine requête ;

$_COOKIE['x'] accès à une variable de cookie ;

setcookie('x','',time()-1000) efface la variable x du cookie ;

setcookie('y',$value,time()+3600) ; expire dans une heure ;

setcookie('x','345') ; expire en fin de session c'est-à-dire lorsque le navigateur sera fermé (expire=0) ;

Cookies et session I

- Les variables de session sont conservées côté serveur
- Du côté client, si les cookies sont acceptées, une variable de cookie, généralement nommée `PHPSESSID` est présente
- Elle contient un identifiant qui est envoyé au serveur à chaque requête ce qui permet de l'associer à la session correspondante
- La variable de configuration `session.cookie_lifetime` spécifie la durée de vie du cookie en secondes
- Par défaut, la valeur de 0 signifie : "Jusqu'à ce que le navigateur soit éteint"
- Cependant, une autre variable de configuration `session.gc_maxlifetime` qui vaut généralement 1440 soit 24 minutes, indique le temps maximum dont jouissent les données de session sur le serveur

Cookies et session II

- Par conséquent, au bout de 24 minutes d'INACTION, les données de session seront supprimées même si le cookie de session perdure côté navigateur

Plan

Développement et débogage I

Le développement d'un projet PHP est réalisé, en général, avec un serveur local à la machine de développement (`localhost`). Puis en production, le projet est déplacé sur une machine serveur qui n'aura pas la même configuration, notamment pour l'affichage des erreurs. Pour déboguer, voici quelques conseils

- l'utilisation d'un bon IDE est indispensable (Zend, VSCode, Sublime)
- l'exécution en ligne de commande du fichier php (`php index.php`) permet de voir sur la sortie d'erreur standard (le terminal) les erreurs de l'interprète PHP que l'on ne voit pas sur le navigateur **notamment les erreurs de syntaxe**. Cependant, d'autres erreurs (session, cookies, ...) apparaîtront et tout ce qui concerne les paramètres (`GET, POST`) ne sera pas testable !
- Visualiser le code HTML sur le navigateur (F12) à l'aide des outils de débogage est **indispensable**

Développement et débogage II

- La console Javascript ou console d'erreurs permet également de visualiser les problèmes locaux aux navigateur : javascript et feuille de style.
- Les données complexes (objets, tableaux) peuvent être affichés récursivement de manière indentée grâce à `var_dump($tab)` ou `print_r($tab)`. Cependant, il faut privilégier `var_dump` car son comportement est modifié par l'utilisation de l'extension PHP indispensable **xdebug**.

Contexte de débogage I

Les fonction `array debug_backtrace();` et `debug_print_backtrace()` permettent de récupérer ou d'afficher le contexte de débogage sous la forme d'un tableau de tableau. Chaque case correspond à un appel de fonction emboité depuis la courante à l'indice 0 (celle qui appele `debug_backtrace`) jusqu'à la plus lointaine (*main*). Chaque case contient un tableau associatif :

function	nom de la fonction courante. Voir aussi __FUNCTION__ ;
line	numéro de la ligne courante. Voir aussi __LINE__ ;
file	nom du fichier courant. Voir aussi __FILE__ ;
class	nom de la classe si on est dans une méthode ;
object	objet courant si on est dans une méthode ;
type	type d'appel : méthode d'instance "->", méthode statique " : :" ;

Contexte de débogage II

args	liste des arguments ou liste des fichiers inclus.

Certaines constantes magiques peuvent être affichées à des fins de débogage :

__FUNCTION__	nom de la fonction courante ;
__FILE__	nom du fichier exécuté ;
__LINE__	numéro de ligne courante ;

Gestion des erreurs I

L'affichage des erreurs de l'interprète PHP est géré par la directive de configuration `display_errors` de `php.ini`. Si cette directive est `on`, on doit fixer un niveau de rapport exigeant à l'interprète afin de vérifier tous les avertissements et erreurs possibles : `error_reporting(E_ALL);`
Si cette directive `display_errors` est à `Off`, ce qui est souhaitable en mode production, les erreurs PHP sont loggées dans un fichier `php_error.log` qu'il faut ouvrir à chaque fois que l'on suspecte une erreur ! Si l'on est hébergé et que l'on ne peut modifier le `php.ini`, on peut modifier la directive `display_errors` :

- soit en débutant chaque script par :

```php
<?php
ini_set('display_errors', 1); error_reporting(E_ALL);
```

- soit en ajoutant un fichier `.htaccess` dans le répertoire (à condition que la directive Apache `AllowOverride` le permette) :

Gestion des erreurs II

```
php_flag display_startup_errors on
php_flag display_errors on
php_flag html_errors on
```


Plan

Débogage avec l'Extension PHP xdebug I

Un débogage professionnel nécessite un IDE permettant le débogage et une extension PHP Xdebug :

- Une extension PHP est une bibliothèque de fonctionalités permettant d'enrichir le langage de base (core)
- Certaines extensions sont destinées à réaliser des graphiques (gd), d'autres à dialoguer avec des SGBD (mysql), etc
- La liste des extensions chargées ainsi que des infos sur la configuration de ces extensions est affiché par `phpinfo()`
- Xdebug permet de déboguer du php en permettant le pas à pas, les points d'arrêts, etc
- Pour cela, Xdebug utilise un pprotocole de débogage `dbgp` qui utilise un **port différent de celui du serveur Web** (par défaut 9000)
- Il contient d'autres fonctionalités (surcharge de `var_dump()` en limitant à 3 la profondeur d'affichage des objets, ...)

Débogage avec l'Extension PHP xdebug II

- Afin de permettre un débogage graphique, il existe des extensions des principaux IDE (VSCode, Sublime, ...) pour dialoguer avec Xdebug

Installation et utilisation de Xdebug sous VSCode I

- télécharger l'extension Xdebug dans le répertoire prévu à cet effet (dans les XAMP, cette phase est généralement inutile) ;
- l'activer dans le `php.ini` du serveur Apache en décommentant la ligne de la section `[xdebug]` correspondant à `zend_extension` ;
- sur MacOS X, modifier 2 fichiers php.ini : `/Applications/MAMP/conf/php7.4.2/php.ini` et `Applications/MAMP/bin/php/php7.4.2/conf/php.ini`
- paramétrer l'extension en modifiant des directives de configuration de sa section ; (ou en utilisant `ini_set(clé,valeur)`)

Installation et utilisation de Xdebug sous VSCode II

```
xdebug.remote_enable=1
xdebug.remote_host=127.0.0.1    ; ip de VSCode
xdebug.remote_connect_back=1  ; Not safe for production
↪  servers
xdebug.remote_port=9000      ; port par défaut de l'IDE
↪  serveur
xdebug.remote_handler=dbgp
xdebug.remote_mode=req
xdebug.remote_autostart=0    ; à éviter sinon toute page
↪  locale
xdebug.idekey=debug    ; à transmettre au serveur dbgp
↪  (VSCode)
```

- télécharger éventuellement l'extension de votre IDE correspondante : par exemple PHP Debug 1.13 de Felix Becker pour VSCode ;
- Sélectionner l'icône de Run/Debug dans la barre d'activité

Installation et utilisation de Xdebug sous VSCode III

- Ouvrir la configuration de lancement en cliquant sur la roue dentée du haut gauche
- éditer le fichier `launch.json` afin de spécifier la config. :

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Listen for XDebug",
      "type": "php",
      "request": "launch",
      "port": 9000
    },
    {
      "name": "Launch currently open script",
      "type": "php",
      "request": "launch",
```

Installation et utilisation de Xdebug sous VSCode IV

```
        "program": "${file}",
        "cwd": "${fileDirname}",
        "port": 9000
      }
    ]
}
```

- Les 2 entrées permettent :

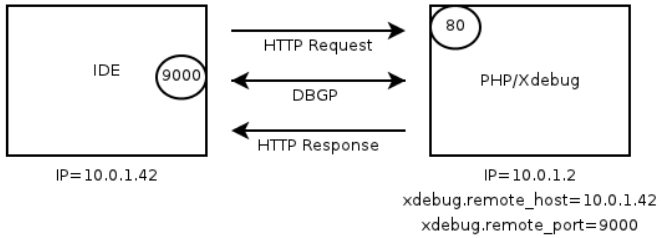
  Launch currently open script permet de déboguer un script en visualisant ses variables, ses superglobales avec des points d'arrêts : mode classique de débogage d'un script

Installation et utilisation de Xdebug sous VSCode V

Listen for XDebug permet à VSCode de créer un serveur dbgp qui écoute sur le port 9000 local une requête provenant de Xdebug agissant comme client dbgp suite à une requête http sur le serveur http. Cela permet de déboguer depuis une interaction humaine dans le navigateur (get ou post)

Installation et utilisation de Xdebug sous VSCode VI

FIGURE – fonctionnement du DeBuG Protocol



La transmission de la clé `idekey` depuis le navigateur vers le serveur http et son extensio XDebug peut être réalisé de plusieurs façons :

Installation et utilisation de Xdebug sous VSCode VII

- via un paramètre GET dans la "query string" : `http://localhost:8888/Preinscriptions/index.php?XDEBUG_SESSION_START=debug`
- via une variable d'environnement : `export XDEBUG_CONFIG="idekey=debug"` pour les scripts lancés depuis un terminal
- via une extension de navigateur comme "Xdebug Helper" pour Firefox

Déboguer sans droit administrateur I

Côté Serveur HTTP

- sans les droits administrateurs, impossible de modifier les fichiers de configuration dont le `php.ini` !
- Il faut donc lancer un serveur `http` local à la ligne de commande (`php -S localhost:8080`) **dans votre répertoire php** `~/public_html/Archiweb/`
- De plus, pour pouvoir déboguer, il faut fournir un certain nombre d'options :

  -dzend_extension=xdebug.so pour utiliser l'extension `Xdebug`
  -dxdebug.remote_enable=1 pour déboguer à distance
  -dxdebug.idekey=debug pour indiquer à l'extension la clé partagée (n'importe quel mot peut être utilisé)
- Au final, la commande peut être sauvée dans le `.bashrc` afin d'éviter les oublis :

Déboguer sans droit administrateur II

```
alias phps="php -S localhost:8080 -dxdebug.remote_enable=1
↪  -dxdebug.idekey=debug"
```

- l'option `-dxdebug.remote_autostart=1` peut aussi être utilisé pour toujours tenter d'ouvrir une session de débogage dans l'IDE sans utiliser de clé partagée
- les options `remote_host` et `remote_port` sont les bonnes par défaut

Côté IDE (serveur `dbgp`) à la FDS

- A la FDS, utiliser VSCodium et installer les deux extensions PHP situées dans le "PHP Extension Pack" de F. Becker (intellisense et debug)
- Ouvrir un dossier (projet) `~/public_html/Archiweb/` contenant des sources php
- choisir la vue PLAY/DEBUG

Déboguer sans droit administrateur III

- cliquer sur la roue dentée pour visualiser le `launch.json`
- ouvrir un fichier php, i.e. `bataille.php`, puis poser un point d'arrêt (disque rouge) en cliquant dans la gouttière verticale à gauche de la ligne désirée
- lancer une session de débogage en choisissant le mode `Listen for XDebug` (la ligne d'état change de couleur et passe à l'orange : on est dans une session)
- ouvrir son navigateur préféré sur l'url `http://localhost/bataille.php?XDEBUG_SESSION_START=debug`
- visualiser dans l'IDE la position courante du débogueur dans le code matérialisé par un triange
- ensuite, exécution pas à pas, examen des variables y compris les superglobales ...
- Ne pas oublier de fermer la session à la fin en cliquant sur l'icône carrée (stop)

Plan

Caractères spéciaux et validation des données I

Sources de séances de débogages longues et pénibles, il est important de comprendre les transformations des caractères spéciaux. Auparavant, la configuration `magic_quotes_gpc` ajoutait automatiquement des **échappement** par un anti-slash de certains caractères. Cette configuration est obsolète. L'échappement doit être réalisé en fonction de chaque formulaire.

- Lors de l'affichage d'une chaîne contenant des caractères html spéciaux (`&"<'>`), il faut les transformer en entité HTML (`&quot;`). Pour afficher proprement une chaîne contenant ces caractères, il faut au préalable la traiter avec `htmlspecialchars($champ, ENT_QUOTES))`.
- **Attention**, si le champ posté doit être remis en tant que "value" dans un `<input type='text'` de formulaire, il faut donc transformer ces caractères en entités HTML.

Caractères spéciaux et validation des données II

- Par conséquent, pour un champ interactif, on écrira : `echo '<input name="c" value="'.htmlspecialchars( $_POST['c'],ENT_QUOTES).'">';`
- les fonctions `html_entities(\$s)` et `html_entity_decode($s)` réalisent le même encodage/décodage pour toutes les entités HTML existantes (e.g. `&Bigcup;` pour le symbole Union)
- Lors de l'envoi d'une requête à un SGBD, il faut parfois échapper les caractères spéciaux tels que : `',\,",NULL`. La fonction `addslashes($chaine)` effectue ce travail. La fonction : `addcslashes (string $str, string $charlist) : string` permet de d'échapper les caractères de str qui sont dans une liste spécifiée dans charlist.
- De même en JavaScript, la fonction `addslashes()` permetra d'échapper du code.

Caractères spéciaux et validation des données III

- Remarquons que dans une session, les caractères spéciaux ne sont pas échappés.

Plan

Administration PHP I

Pour visualiser la configuration PHP, il suffit d'appeler la fonction `phpinfo()` pour voir :

- la version de php ;
- la localisation du fichier de configuration de php : `php.ini` ;
- les librairies incluses et leur configuration (mysql, gd, ...) ;
- de nombreuses autres choses importantes.

La configuration de l'interprète PHP est réalisé dans le fichier `php.ini`. Ce fichier `php.ini` contient des directives (sous forme d'affectation de variables) fondamentales :

register_globals=On si **on** alors les variables d'Environnement, Get, Post, Cookie, Serveur sont globales ; si **off** alors les variables sont accessibles via `$_POST[]`, ... (Sécurité +) ;

Administration PHP II

variables_order="EGPCS" ordre de résolution des conflits de noms de variables Env, GET, POST, COOKIE, Serveur ;

magic_quotes_gpc=On permet d'anti-slasher les caractères anti-slash, guillemet et apostrophe dans les variables GPC.

Plan

Authentification I

L'authentification par un couple (login, password) peut être réalisé de bien des façons différentes sur le web. Dans la plupart des cas, on peut réaliser l'authentification par un formulaire HTML classique puis interrogation en PHP d'une base de données des utilisateurs contenant les couples (login, password **crypté**). Une fois vérifié la concordance, il suffira de conserver dans le tableau de session les informations concernant l'utilisateur loggé (id, nom, type, ...) : `$_SESSION['uid']`, ...
Cependant, il faut également connaître les autres types d'authentification !
La procédure d'**authentification HTTP** est associée à un **nom de domaine** (realm ou AuthName) et à un répertoire. Elle peut être déclenchée :

- soit par Apache, indépendamment de PHP ;
- soit en utilisant PHP.

Authentification II

L'authentification **HTTP via Apache** est valable pour toute une **arborescence**. Un fichier `.htaccess` spécifie les règles d'authentification valables pour ce répertoire et tous ses **descendants** :

```
AuthType Basic
AuthUserFile "/auto/.../AuthApache/.htpasswd"
AuthName "Le Domaine Privé"
<Limit GET POST>
 require valid-user
</Limit>
```

Le fichier `.htpasswd` contient la liste des utilisateurs et leurs mots de passe cryptés. Il est obtenu grâce à l'utilitaire `htpasswd` fourni par Apache. Par exemple : `htpasswd -c .htpasswd un` ; crée un fichier avec l'utilisateur `un`.
Lors de tout accès à un fichier descendant de `AuthApache`, Apache va envoyer un en-tête au navigateur client qui va afficher une fenêtre popup

Authentification III

d'authentification. Par la suite, l'utilisateur reste authentifié pour "Le Domaine Privé" jusqu'à la fin du navigateur ou si une nouvelle authentification PHP est lancée.

Authentification HTTP via PHP I

PHP doit être un module d'Apache. On utilise alors la fonction `header` pour demander une authentification ("WWW-authenticate") au client, générant ainsi l'apparition d'une fenêtre de demande d'utilisateur et de mot de passe. Une fois que les champs ont été remplis, l'URL sera de nouveau appelée, avec les variables `$_SERVER['PHP_AUTH_USER']`, `$_SERVER['PHP_AUTH_PW']` et `$_SERVER['PHP_AUTH_TYPE']` contenant respectivement le nom d'utilisateur, le mot de passe et le type d'authentification. Actuellement, seule l'authentification de type "Basic" est supportée. Si l'authentification est réalisée via Apache (`.htaccess`), la variable `$_SERVER['REMOTE_USER']` est égale à `$_SERVER['PHP_AUTH_USER']`.

Authentification HTTP via PHP II

Exemple (Exemple d'authentification HTTP par PHP)

```php
<?php
if (!isset($_SERVER['PHP_AUTH_USER']) || !verifierAuth())
  header("WWW-Authenticate: Basic realm=\"Le Domaine Privé\"");
  header("HTTP/1.0 401 Unauthorized");
  echo "Texte à envoyer si le client annule\n";
  exit;
  } else {
  echo "Bonjour ",$_SERVER['PHP_AUTH_USER'];
  // suite de la page privée
  }
?>
```

La fonction booléenne `verifierAuth()` utilisera tout moyen nécessaire à la vérification du nom et du mot de passe (Base de données, ...).
Remarquons que les variables `$_SERVER['PHP_AUTH_...']` sont

Authentification HTTP via PHP III

utilisables même si l'authentification n'a pas été effectuée par le module PHP.
Les variables d'authentification PHP sont valables pour tous les fichiers descendants du répertoire. Par contre, tout fichier html ou php ne testant pas l'authentification est accessible, contrairement à l'authentification par .htaccess (Apache) qui sécurise tout le répertoire.
Désauthentification PHP : Le navigateur écrase le cache d'authentification client d'un domaine quand il reçoit une nouvelle demande d'authentification. Cela permet de déconnecter un utilisateur, pour le forcer à entrer un nouveau nom et son mot de passe. Si l'utilisateur annule l'authentification, il est alors désauthentifié ! Penser à recharger la page.
Certains programmeurs changent dynamiquement le nom de domaine pour donner un délai d'expiration, ou alors, fournissent un bouton de réauthentification.

Téléchargement I

- du site web vers le client : download ou téléchargement ;
- du client vers le site web : upload ou chargement ou téléversement ;

En HTML, pour permettre le download, on réalise un lien référençant le fichier à télécharger, par exemple :
`<ahref="Public/toto.pdf">Cliquerici</a>`. Si le type du fichier est affichable par le navigateur, il sera affiché (après son téléchargement), sinon il sera proposé à l'utilisateur soit de sauver le fichier, soit de lancer l'application associée. Remarquons que tout lien peut être enregistré en utilisant le bouton droit de la souris sur le lien.

Téléversement I

On veut réaliser un formulaire de chargement envoyant une requête POST. L'action effectuée (script) après soumission doit vérifier que le fichier chargé est du bon type et de la bonne taille puis l'afficher depuis la zone temporaire (tmp/) ou il est chargé. Le fichier temporaire sera **automatiquement effacé** de la zone à la fin du script, s'il n'a pas été déplacé ou renommé :

```
<FORM ENCTYPE="multipart/form-data" ACTION="traitement.php"
↪  METHOD="POST">
  <INPUT TYPE="hidden" NAME="MAX_FILE_SIZE" value="1000">
  Envoyez ce fichier : <INPUT NAME="fichier" TYPE="file"
  ↪  SIZE=15>
  <INPUT TYPE="submit" VALUE="Envoyer le fichier">
</FORM>
```

Dans le script `traitement.php` on a accès à différentes variables (différent selon les versions de PHP) :

Téléversement II

- `$_FILES['fichier']['name']` le nom du fichier original chez le client ;
- `$_FILES['fichier']['type']` le type mime du fichier "image/gif, text/plain, ..." ;
- `$_FILES['fichier']['size']` la taile du fichier chargé ;
- `$_FILES['fichier']['tmp_name']` le nom du fichier temporaire sur le serveur ;
- `$_FILES['fichier']['error']` le code d'erreur (PHP 4.2.0).

Remarquons que tous les navigateurs ne vérifient pas tous le `MAX_FILE_SIZE`.

SPL et Phar I

- La *Standard PHP Library (SPL)* fournit par défaut (sans include, sans modifier le php.ini) des interfaces et des classes permettant de gérer des collections de données. Par exemple, suit une liste de quelques classes utiles de la SPL : SplDoublyLinkedList, SplStack, SplQueue, SplHeap, SplPriorityQueue, SplObjectStorage. Mais aussi des fonctions comme `spl_autoload()`.
- `phar` est un format de fichier archive (PHp ARchive) permettant de sauvegarder une application (arborescence) dans un unique fichier d'extension `phar`. C'est l'équivalent des fichiers `jar` de Java.
  L'interprète PHP en ligne de commandes est capable d'exécuter un fichier `phar` directement : `$ php monappli.phar`
  On peut également définir une bibliothèque dans un fichier phar et n'inclure que certaines parties grâce à une syntaxe appropriée :

SPL et Phar II

```php
<?php
include 'malib.phar'; // inclusion de tous les fichiers
include 'phar://malib2.phar/Commun/inc-Html.php'; // Commun
↪  est un répertoire de malib2
```

Le format interne des fichiers de l'archive peut utiliser différents formats de compression (zip, phar ou tar).

Sérialisation des tableaux et objets I

La sérialisation permet de transformer toute variable complexe (objet, tableau) de PHP en une chaîne de caractères qui pourra être désérialisée ultérieurement. La chaîne sérialisée peut être sauvée dans un fichier ou transmise sur un réseau.

```
string serialize  ( mixed $value  )
mixed unserialize  ( string $str  )
```

Dans une session PHP, tableau superglobal `$_SESSION`, les variables sont sérialisées automatiquement dans le fichier du serveur qui les conserve entre 2 requêtes.
La sérialisation réalise une copie **profonde** d'un objet contrairement à l'opérateur clone qui effectue une copie superficielle.

```
$copieprof=unserialize(serialize($object))
```


Sérialisation des tableaux et objets II

Attention, les attributs de type ressource (fichier, connexion) ne sont pas sérialisables.

Plan

Patron de conception (design pattern) I

- en développement logiciel, un patron de conception (design pattern) est une structure de code reconnu comme **bonne pratique** en réponse à un problème de conception d'un logiciel
- solution standard, utilisable dans la conception de différents logiciels dans différents langages de programmation
- un patron = une manière d'organiser des modules ou des classes afin de répondre à un besoin récurrent
- 1994 « Gang of Four » (GoF, Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides : Design Patterns – Elements of Reusable Object-Oriented Software
- "Les patrons offrent la possibilité de capitaliser un savoir précieux né du savoir-faire d'experts" (Buschmann, 1996)

Formalisme I

Structure d'un patron de conception :

Nom identifiant du problème (singleton, proxy, ...)

Description du problème à résoudre

Solution les éléments de la solution, avec leurs relations. La solution est appelée patron de conception (design pattern)

Conséquences résultats issus de la solution

Formalisme II

Orthogonalité

- Chaque patron doit correspondre à une approche différente, qui ne répète pas les idées ou stratégies présentes dans d'autres patrons
- Le concepteur analyse un problème complexe et en résoud chaque aspect à l'aide un design pattern
- Il combine ensuite les patrons pour construire une solution
- Certains auteurs de patrons (Vlissides, Grand) proposent davantage d'orthogonalité que les patrons GoF

Quelques un des vingt-trois patrons GoF I

- Fabrique et Fabrique abstraite (Factory) : Ce patron fournit une interface pour créer des familles d'objets sans spécifier la classe concrète. Une fabrique simple retourne une instance d'une classe parmi plusieurs possibles, en fonction des paramètres qui ont été fournis. Toutes les classes ont un lien de parenté, et des méthodes communes, et chacune est optimisée en fonction d'une certaine donnée. Le patron fabrique abstraite est utilisée pour obtenir un jeu d'objets connexes (e.g. composant graphique (boutons, menus) dans le style de Windows, ou bien dans le style de Motif, ou bien dans le style de MacOS.

Quelques un des vingt-trois patrons GoF II

- Adaptateur (adaptator) : Ce patron convertit l'interface d'une classe en une autre interface exploitée par une application. Permet d'interconnecter des classes qui sans cela seraient incompatibles. Il est utilisé dans le cas où un programme se sert d'une bibliothèque de classe qui ne correspond plus à l'utilisation qui en est faite, à la suite d'une mise à jour de la bibliothèque dont l'interface a changé. Un objet adaptateur (en anglais adapter) expose alors l'ancienne interface en utilisant les fonctionnalités de la nouvelle
- Pont (bridge) : Ce patron permet de découpler une abstraction de son implémentation, de telle manière qu'ils peuvent évoluer indépendamment. Il consiste à diviser une implémentation en deux parties : une classe d'abstraction qui définit le problème à résoudre, et une seconde classe qui fournit une implémentation. Il peut exister plusieurs implémentations pour le même problème et la classe d'abstraction comporte une référence à l'implémentation choisie, qui

Quelques un des vingt-trois patrons GoF III

peut être changée selon les besoins. Fréquemment utilisé pour réaliser des récepteurs d'événements

- Monteur (builder) : Ce patron sépare le processus de construction d'un objet du résultat obtenu. Permet d'utiliser le même processus pour obtenir différents résultats. C'est une alternative au pattern fabrique. Au lieu d'une méthode pour créer un objet, à laquelle est passée un ensemble de paramètres, la classe fabrique comporte une méthode pour créer un objet. Cet objet comporte des propriétés qui peuvent être modifiées et une méthode pour créer l'objet final en tenant compte de toutes les propriétés. Ce pattern est particulièrement utile quand il y a de nombreux paramètres de création, presque tous optionnels

Quelques un des vingt-trois patrons GoF IV

- Chaîne de responsabilité (chain of responsibility) : Ce patron vise à découpler l'émission d'une requête de la réception et le traitement de cette dernière en permettant à plusieurs objets de la traiter successivement. Dans ce patron chaque objet comporte un lien vers l'objet suivant, qui est du même type. Plusieurs objets sont ainsi attachés et forment une chaîne. Lorsqu'une demande est faite au premier objet de la chaîne, celui-ci tente de la traiter, et s'il ne peut pas il fait appel à l'objet suivant, et ainsi de suite
- Commande (command) : Ce patron emboîte une demande dans un objet, permettant de paramétrer, mettre en file d'attente, journaliser et annuler des demandes. Dans ce patron un objet commande correspond à une opération à effectuer. L'interface de cet objet comporte une méthode execute. Pour chaque opération, l'application va créer un objet différent qui implémente cette interface — qui comporte une méthode execute. L'opération est lancée lorsque la

Quelques un des vingt-trois patrons GoF V

méthodeexecute est utilisée. Ce patron est notamment utilisé pour les barres d'outils

- Décorateur (decorator) : Le patron décorateur permet d'attacher dynamiquement des responsabilités à un objet. Une alternative à l'héritage. Ce patron est inspiré des poupées russes. Un objet peut être caché à l'intérieur d'un autre objet décorateur qui lui rajoutera des fonctionnalités, l'ensemble peut être décoré avec un autre objet qui lui ajoute des fonctionnalités et ainsi de suite. Cette technique nécessite que l'objet décoré et ses décorateurs implémentent la même interface, qui est typiquement définie par une classe abstraite

Quelques un des vingt-trois patrons GoF VI

- Iterator Ce patron permet d'accéder séquentiellement aux éléments d'un ensemble sans connaitre les détails techniques du fonctionnement de l'ensemble. Selon la spécification originale, il consiste en une interface qui fournit les méthodes Next() et Current(). L'interface en Java comporte généralement une méthode nextElement et une méthode hasMoreElements
- Proxy : Ce patron est un substitut d'un objet, qui permet de contrôler l'utilisation de ce dernier. Un proxy est un objet destiné à protéger un autre objet. Le proxy a la même interface que l'objet à protéger. Un proxy peut être créé par exemple pour permettre d'accéder à distance à un objet (via un middleware). Le proxy peut également être créé dans le but de retarder la création de l'objet protégé — qui sera créé immédiatement avant d'être utilisé. Dans sa forme la plus simple, un proxy ne protège rien du tout et transmet tous les appels de méthode à l'objet cible

Quelques un des vingt-trois patrons GoF VII

- Singleton : Ce patron vise à assurer qu'il n'y a toujours qu'une seule instance d'une classe en fournissant une interface pour la manipuler. C'est un des patrons les plus simples. L'objet qui ne doit exister qu'en une seule instance comporte une méthode pour obtenir cette unique instance et un mécanisme pour empêcher la création d'autres instances.
- Observer Ce patron établit une relation un à plusieurs entre des objets, où lorsqu'un objet change, plusieurs autres objets sont avisés du changement. Dans ce patron, un objet le sujet tient une liste des objets dépendants des observateurs qui seront avertis des modifications apportées au sujet. Quand une modification est apportée, le sujet émet un message aux différents observateurs. Le message peut contenir une description détaillée du changement. Dans ce patron, un objet observer comporte une méthode pour inscrire des observateurs. Chaque observateur comporte une méthode Notify.

Quelques un des vingt-trois patrons GoF VIII

Lorsqu'un message est émis, l'objet appelle la méthode Notify de chaque observateur inscrit

- Strategy : Dans ce patron, une famille d'algorithmes est encapsulée de manière qu'ils soient interchangeables. Les algorithmes peuvent changer indépendamment de l'application qui s'en sert. Il comporte trois rôles : le contexte, la stratégie et les implémentations. La stratégie est l'interface commune aux différentes implémentations (classe abstraite). Le contexte est l'objet qui va associer un algorithme avec un processus.
- Composite : Le patron composite permet de composer une hiérarchie d'objets, et de manipuler de la même manière un élément unique, une branche, ou l'ensemble de l'arbre. Il permet en particulier de créer des objets complexes en reliant différents objets selon une structure en arbre. Ce patron impose que les différents objets aient une même interface, ce qui rend uniformes les manipulations de la structure.

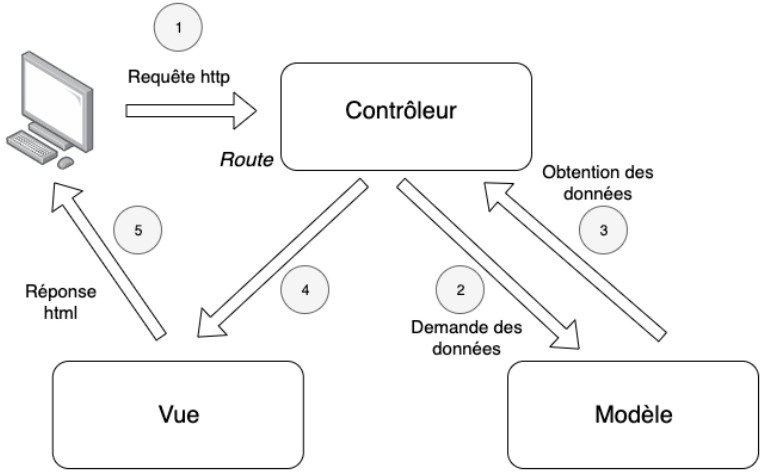
Plan

Model View Controller I

Un patron d'architecture est une combinaison de patrons de conception :

- MVC : architecture des applications avec interface graphique. Le modèle est chargé de fournir et de sauvegarder les données (BD SQL ou NoSQL). La vue est le rendu graphique de l'application. Le contrôleur reçoit des messages provenant de l'interaction utilisateur (submit, ...) ou de déclencheur (trigger) du modèle et répond à ceux-ci en consultant et/ou modifiant le modèle et en mettant à jour la vue.
- inversion de contrôle (IoC) : le framework (cadriciel) prend en charge l'exécution principale du programme ; il coordonne et contrôle l'activité de l'application. Le programme utilisateur définit alors les blocs de codes (composants) en utilisant l'API fournie à cet effet par le framework, sans relation dure entre eux. Ces blocs de codes sont laissés à la discrétion du framework qui se chargera de les appeler.

Model View Controller II



Plan

Pourquoi utiliser MVC ? Un exemple de code I

```
<!doctype html><html lang="fr"><head>
<meta charset="utf-8" />
<title>Liste des étudiant.e.s</title>
</head>
<body>
<center>
  <h1>Liste des étudiant.e.s</h1>
</center>
<?php
ini_set('display_errors', 1); error_reporting(E_ALL);
include("perso.php");
try {
  $dbh = new PDO($dsn, $login, $passwd);
} catch (PDOException $e) {
  die("Impossible d'ouvrir la base de données !: " .
  ↪  $e->getMessage() . "<br/>");
```


Pourquoi utiliser MVC ? Un exemple de code II

```
}
$req="SELECT * ";
$req.="FROM etudiant ";
$req.="ORDER BY nom, prenom ;";
// echo $req;
$res = $dbh->query($req);
if (!$res) die("Impossible d'exécuter la requête !");
// Début de l'affichage
echo '<table><tr><th>nom<th>prenom<th>statut<th>groupe<th>email
↪  <th>opt<th>numStageA</tr>';
foreach ($res as $ligne){        // tq il reste des Stage-étud
  echo '<tr>';
  for($i=0; $i<7; $i++){
    echo "<td>{$ligne[$i]}</td>";
  }
}
echo '</table>';
```


Pourquoi utiliser MVC ? Un exemple de code III

```
$dbh = null;    // fermeture connexion
?>
<br>
<h1><a href="index.html">Retour</a></h1>
</body>
</html>
```


Problèmes I

- mélange d'opérations en PHP, de requête SQL et de code HTML
- on pourrait avoir également du JavaScript et du code CSS au milieu
- Code très difficile voire impossible à maintenir par différents programmeurs (projet en groupes)
- On va donc séparer les codes correspondant à différents langages et à différents objectifs

Séparation en 3 fichiers I

On va répartir chaque page en 3 fichiers :

- Le contrôleur reçoit la requête et ses paramètres, lance le modèle qui effectue l'accès aux données via PDO, puis lance la vue qui produit le HTML ; Le contrôleur fait le lien entre Modèle et Vue (index.php)
- Le modèle se connecte à la BD, puis réalise une requête ou plusieurs (transaction), puis se termine (modele.php)
- La vue récupère les données produites par le modèle et effectue l'affichage (HTML) (vue.php)

L'inclusion du modèle et de la vue dans le contrôleur sera effectuée par `require()` qui engendrera une erreur si le fichier correspondant n'existe pas.

Liste des étudiants MVC I

Le contrôleur

```php
<?php
  ini_set("display_errors", 1); error_reporting(E_ALL);
  require("listeEtudiantsM.php"); // modèle
  require("listeEtudiantsV.php"); // vue
?>
```

Le modèle

```php
<?php
require("perso.php"); // dsn login pass
try {
  $dbh = new PDO($dsn, $login, $passwd);
} catch (PDOException $e) {
  die("Impossible d'ouvrir la base de données !: " .
  ↪  $e->getMessage() . "<br/>");
}
$req="SELECT * ";
```


Liste des étudiants MVC II

```php
$req.="FROM etudiant ";
$req.="ORDER BY nom, prenom ;";
$res = $dbh->query($req);
if (!$res) die("Impossible d'exécuter la requête !");
$dbh = null;   // fermeture connexion

?>
```

La Vue

```
<!doctype html><html lang="fr"><head>
<meta charset="utf-8" />
<title>Liste des étudiant.e.s</title>
</head>
<body>
<center><h1>Liste des étudiant.e.s</h1></center>
<?php  // Début de l'affichage
echo '<table><tr><th>nom<th>prenom<th>statut
↪  <th>groupe<th>email<th>opt<th>numStageA</tr>';
```


Liste des étudiants MVC III

```php
foreach ($res as $ligne){        // tq il reste des étud.
  echo '<tr>';
  for($i=0; $i<7; $i++){
    echo "<td>{$ligne[$i]}</td>";
  }
}
$dbh = null;   // fermeture connexion (entorse MVC ...)
echo '</table>';
?>
<br><h1><a href="index.html">Retour</a></h1>
</body></html>
```

Perfectionnement de la liste des étudiants I

On souhaite améliorer la modularité de notre conception MVC :

- la connexion à la base de données doit être fermée à la fin du parcours de la table par la vue
- on crée une classe singleton `Connexion` qui permet de n'ouvrir qu'une seule fois la connexion dans le contrôleur même en cas de vues multiples
- cette connexion sera fermée par le contrôleur
- le modèle définit une fonction `getEtudiants()` permettant de sélectionner certains étudiants selon certains critères
- utiliser la balise raccourcie `<?= $x ?>` au lieu de `<?php echo $x; ?>`

Connexion.php I

```php
<?php
class Connexion {
  private static $connexion=null;   // attribut de classe
  ↪ conservant la connexion

  private function __construct(){} // constructeur inaccessible

  public static function getConnexion(){
    if (!self::$connexion){    // si pas encore créé
      require_once("perso.php"); // dsn login pass
      try {
        self::$connexion = new PDO($dsn, $login, $passwd); //
        ↪ création et sauvegarde
      } catch (PDOException $e) {
        die("Impossible d'ouvrir la base de données !: " .
        ↪ $e->getMessage() . "<br/>");
```


Connexion.php II

```php
      }
    }
    return self::$connexion;
  }
  public static function close(){
    self::$connexion=null;
  }
}
?>
```


Le Contrôleur : listetuC.php I

```php
<?php
  ini_set("display_errors", 1); error_reporting(E_ALL);

  // 1. validation des paramètres
  if(isset($_GET['statut']) && ($_GET['statut']=='FI' ||
  ↪ $_GET['statut']=='FP')){
    $statut=$_GET['statut'];
  } else $statut=null;
  if(isset($_GET['groupe']) &&
  ↪ preg_match("/^[0-9]/",$_GET['groupe'])){
    $groupe=$_GET['groupe'];
  } else $groupe=null;

  // 2. appel au modèle
  require("Connexion.php"); // classe singletion Connexion
  $pdo=Connexion::getConnexion();
```

Le Contrôleur : listetuC.php II

```php
  require("listetuM.php"); // modèle
  $listetu=getEtudiants($pdo, $statut, $groupe);
      // var_dump($listetu);
  // 3. Vue
  require("listetuV.php"); // vue
  Connexion::close(); // fermer la connexion à la fin car curseur
  ↪  parcouru
?>
```


Le Modèle :listetuM.php I

```php
<?php
function getEtudiants($connexion, $statut=null, $groupe=null){
  $req="SELECT * FROM etudiant ";
  $clauseWhere=0; // pas de clause Where
  if ($statut){
    $req.=" WHERE statut='$statut' " ;
    $clauseWhere++;
  }
  if ($groupe){
    $req.=($clauseWhere?" AND " : " WHERE") . " groupe='$groupe'
    ↪  " ;
    $clauseWhere++;
  }
  $req.=" ORDER BY nom, prenom ; ";
  $res = $connexion->query($req);
  if (!$res)
```


Le Modèle :listetuM.php II

```php
    die("Impossible d'exécuter la requête : $req !");
  return $res;
}
?>
```


La Vue :listetuV.php I

```php
<!doctype html><html lang="fr"><head>
<meta charset="utf-8" />
<title>Liste des étudiant.e.s</title>
</head>
<body>
<center><h1>Liste des étudiant.e.s</h1></center>

<table><tr><th>nom<th>prenom<th>statut<th>groupe<th>email
<th>opt<th>numStageA</tr>

<?php  // Début de l'affichage php
foreach ($listetu as $ligne){      // tq il reste des étudiants
  echo '<tr>';
  for($i=0; $i<7; $i++){
    echo "<td>{$ligne[$i]}</td>";
  }
```

La Vue :listetuV.php II

```
}
?>
</table><br><h1><a href="index.html">Retour</a></h1>
</body></html>
```


Aller plus loin ... I

- utiliser le tampon de sortie (Output Buffer) avec les fonctions PHP `ob_start()` et `ob_end_flush()`
- Les frameworks PHP tels que Symfony ont développé un langage de template (appelé Twig pour Symfony), qui permet de ne pas utiliser du tout de code PHP dans la vue
- Pour le Modèle, des Object Relational Mapping (ORM) existent également dans les cadriciels afin de manipuler des objets PHP sans avoir à réaliser de requêtes SQL (Doctrine pour Symfony)
- Le contrôleur doit généralement exécuter des vérifications des paramètres passés dans la query string (`?nom=toto&prenom=michel`) ou par la méthode POST
- La multiplication des pages d'une application Web va engendrer 3 fois plus de fichiers : une structuration en répertoires est souhaitable

Aller plus loin ... II

- Si la mise en page est la même dans tout le site (header, nav, main, footer), il faut factoriser au moins l'entête et le pied de page dans des fichiers inclus par chaque vue
- Il est parfois souhaité qu'une seule page index.php représente tout le site ! Il faudra utiliser alors un paramètre GET (action ou page) pour définir la page à atteindre : `index.php?page=listeEtudiantC`. Le fichier `index.php` peut être vu alors comme un super contrôleur du site qu'on appellera routeur
- Un mécanisme de réécriture d'URL (URL rewriting) existe dans la configuration d'Apache qui permet de router sans utiliser de paramètres mais en allongeant le chemin

Plan

Plan

Introduction I

- Doctrine 2 est un ORM depuis PHP 5.4+
- C'est un middleware (intergiciel) situé entre la couche métier (objets PHP) et la couche Données (SGBD Relationnel)
- Il est possible de l'utiliser avec les frameworks Symfony 2, Zend Framework, ...
- Une correspondance objet-relationnel (en anglais object-relational mapper ou ORM) est un patron de conception (design pattern) qui permet de manipuler une base de données relationnelle en utilisant des objets PHP persistants
- Ainsi le code PHP est plus homogène, ce qui facilite sa programmation et son débogage
- Doctrine 2 est une refondation complète de Doctrine 1 et utilise un grand nombre de patron de conception (Design Pattern)

Introduction II

- La manipulation des objets PHP (persistance, consultation) est réalisée grâce à un Gestionnaire d'Entités (EntityManager) qui implémente le "data mapper pattern"
- Une **entité** est un object PHP identifié par un attribut unique qui est lié à la clé primaire de la ligne qui lui correspond
- Une classe d'entités possède des entités, chaque entité correspondant à une ligne d'une table relationnelle
- génération de la BD à partir de classes d'entités métiers ou construction du modèle objet à partir d'une BD existante (rétro-ingéniérie) ;
- pas de schéma XML intermédiaire entre modèle objet et BD (Propel)
- DQL (Doctrine Query Language) inspiré par Hibernate Query Language (ORM Java)

Introduction III

- les entités sont liées entre elles par des associations : une association matérialise sur les objets PHP une contrainte d'intégrité référentielle de la BD ;
- des transactions ACID sont présentes
- l'actualisation des associations (références à d'autre(s) entité(s)) dans les entités est réalisée de manière **fainéante** (lazy) ; c'est-à-dire au fur et à mesure des accès ;
- Plusieurs démarches de conception de projet existent :
  - "code first" consiste à suivre la procédure suivante consistant à écrire le code PHP des entités et leurs métadonnées puis à créer la BD ;
  - "database first" permet de construire automatiquement le modèle objet PHP à partir d'une BD existante (rétro-ingéniérie) ;
  - "model first" devrait permettre de tout construire (entités et BD) à partir d'un modèle UML ...

Introduction IV

Actuellement, seule la démarche “code first” est totalement fonctionnelle. La création programmée d’entités à partir d’une BD existante nécessite de rajouter par la suite certaines associations entre entités oubliées par l’outil.

Plan

Composer : un gestionnaire de dépendances I

- Historiquement `pear` fut un gestionnaire de paquetages PHP centralisé qui permettait de nombreuses fonctionalités dont la gestion des dépendances entre paquetages mais qui était un peu rigide
- les auteurs de paquetages ou de librairies PHP sont maintenant hébergés sur différents gestionnaires de version (`GitHub`, `GoogleCode`, `...`) et proposent leur librairie sous différents formats (zip, tgz, phar, ...)
- Composer est un gestionnaire de dépendances de paquetages
- Il est associé à un gestionnaire de paquetage `Packagist` qui lui maintient en dépôt (git) tous les paquetages utilisables avec Composer
- Propel dépend du projet Phing qui est un portage de Ant en PHP
- Ant, make, Phing sont des constructeurs de projet qui permettent d’automatiser les tâches de fabrication d’une application.

Plan

Installation de Composer I

```
cd public_html/
mkdir MonProjet
cd MonProjet
curl -sS https://getcomposer.org/installer | php
```

- La commande curl permet de télécharger l'exécutable php `installer.phar` (PHp ARchive)
- Puis on passe via un tube ce même fichier à l'interprète `php` qui exécute l'installateur
- A partir de là, le fichier `composer.phar` est installé et est disponible en ligne de commande
- Pour installer des paquetages dans un projet, il suffit d'éditer le fichier JSON `composer.json` indiquant ce que l'on veut installer et de lancer la ligne de commande `php composer.phar install`

Plan

Installation de doctrine I

Doctrine s'installe, comme Propel, en utilisant `Composer`. On crée le fichier `composer.json` suivant :

```
{
  "require": {
    "doctrine/orm": "*"
  }
}
```

Puis on lance la ligne de commande suivante :

```
php composer.phar install
```

Le paquetage Doctrine installé, on peut avoir besoin d'outils en ligne de commande de la façon suivante :

```
$ php vendor/bin/doctrine ...
```


Plan

Le gestionnaire d'entité et la persistance I

La classe EntityManager fournit le point d'accès au cycle de vie complet des entités. C'est une classe singleton utilisant le patron de conception factory :

```php
<?php // obtaining the entity manager
$entityManager = EntityManager::create($conn, $config);
```

Cet objet PHP `$entityManager` nous permettra par la suite d'enregistrer les entités devant persister en deux phases :

- `$entityManager->persist($product);` permet de tamponner l'entité `product` dans le gestionnaire d'entités
- cette méthode `persist` sera utilisée plusieurs fois avant de lancer la transaction de BD qui effectuera l'ensemble des requêtes (insert ou update ou delete).
- `$entityManager->flush();` lance la transaction sur la BD

Le gestionnaire d'entité et la persistance II

- Remarquons que toute entité enregistrée (tamponnée) par `persist()` peut être modifiée par la suite, l'état final de l'entité sera enregistré lors du `flush()`
- la création de l'objet `$entityManager` ainsi que d'autres initialisations est effectuée une fois en début de tout script PHP en incluant un fichier `bootstrap.php` : `require_once "bootstrap.php";`

Plan

Classe d'entité et métadonnées I

Cependant, avant de rendre persistantes les entités, il faut les définir :

- la classe d'entité en PHP : ensemble d'attributs protégés ayant chacun une paire d'accesseur/mutateur (get/set) et un attribut `id` qui correspond à la clé primaire et qui lui ne possède qu'un accesseur (dans le répertoire src)
- chaque classe peut posséder des méthodes **métiers**
- les métadonnées de la classe qui représentent la liaison entre classe et table et entre attribut et colonne
- Ces métadonnées peuvent être fournies en XML, en YAML (XML indenté) dans un fichier externe ou en PHP grâce à des annotations (`/** @Entity ...`) dans le fichier définissant la classe d'entité

Classe d'entité et métadonnées II

```php
<?php          // src/Product.php
/** @Entity **/
class Product{
  /**
  * @var int
  */
  protected $id;
  /**
  * @var string
  */
  protected $name;

  public function getId(){
    return $this->id;
  }
  public function getName(){
    return $this->name;
```


Classe d'entité et métadonnées III

```
  }
  public function setName($name){
    $this->name = $name;
  }
}

# Code des métadonnées en YAML
# config/yaml/Product.dcm.yml (dans le répertoire config/yaml)
Product:
  type: entity
  table: products
  id:
    id:
      type: integer
      generator:
        strategy: AUTO
  fields:
```


Classe d'entité et métadonnées IV

```
    name:
      type: string
```


Plan

Génération du schéma de BD et CLI doctrine I

Afin de générer le schéma de BD en SQL (CREATE TABLE ...) et de l'exécuter, il faut utiliser la commande `doctrine` suivante depuis un terminal :

```
$ vendor/bin/doctrine orm:schema-tool:create  --dump-sql
CREATE TABLE products (id INTEGER NOT NULL, name VARCHAR(255)
  NOT NULL, PRIMARY KEY(id));
```

- On peut vérifier grâce à PhpMyAdmin que la table `products` a bien été créée
- En cas de modification du schéma de BD, ajout de classe d'entités ou modifications, il faut penser à chaque fois à mettre à jour le schéma.

```
$ vendor/bin/doctrine orm:schema-tool:update --force
```

L'option `force` permet d'éviter de changer de schéma de BD par mégarde alors qu'on est en environnement de production. En effet, Cela risque de modifier complètement le comportement des scripts.

Plan

Utilisation de la BD I

Afin de tester, nous allons créer un script interactif php ajoutant des produits ayant comme nom les arguments passés à la ligne de commande :

```php
<?php          // creerProd.php
require_once "bootstrap.php";  // fichier de configuration et em
$p=array();
for ($i=1; $i<count($argv);$i++){
  $p[$i] = new Product();
  $p[$i]->setName($argv[$i]);
  $entityManager->persist($p[$i]);
  echo "produit enregistré (id,name) = (".$p[$i]->getId().",
  ".$p[$i]->getName().")\n";
}
$entityManager->flush();
echo "Produits insérés en BD !\n";
for ($i=1; $i<count($argv);$i++){
  echo "produit (id,name) = (" . $p[$i]->getId() . "," .
```


Utilisation de la BD II

```php
  $p[$i]->getName() . ")\n";
}
```

Examinons une exécution :

```
$ php creerProd.php toto abc foo
produit enregistré (id,name) = (,toto)
produit enregistré (id,name) = (,abc)
produit enregistré (id,name) = (,foo)
Produits insérés en BD !
produit (id,name) = (9,toto)
produit (id,name) = (10,abc)
produit (id,name) = (11,foo)
```

Remarquons qu'avant le vidage (flush) les entités n'ont pas d'identifiant car celui-ci est ensuite fourni par MySQL puisque c'est une clé primaire entière auto-incrémentée.

Plan

EntityRepository (dépôt d'entités) I

- pour manipuler l'ensemble des lignes de la table `products`, on utilise un dépôt (repository) via l'entityManager
- puis, on lui applique une requête (findAll).

```php
<?php        // listeProduits.php
require_once "bootstrap.php";

$productRepository = $entityManager->getRepository('Product');
$products = $productRepository->findAll();

foreach ($products as $p) {
  echo sprintf("%d %s<br>\n",$p->getId(), $p->getName(),
    $p->getId());
}
```


Consultation des entités par clé primaire I

La méthode `find($entityName, $id)` de l'`entityManager` permet de récupérer une entité par son identifiant. Le formulaire web suivant permet d'afficher un produit :

```
<form method="get">      <!-- chercheProduit.php -->
  Id<input type="number" name="id">
  <input type="submit" Value="Chercher !">
</form>
```

```php
<?php
if(isset($_GET["id"])){
  require_once "bootstrap.php";
  $p = $entityManager->find('Product',$_GET["id"]);
  echo sprintf("%d %s<br>\n",$p->getId(), $p->getName());
}
```


Consultation par des conjonctions d'égalité (where name='x' and id=5) I

On peut utiliser `findBy()` ou `findOneBy()` sur un dépôt en fournissant un tableau de (clé, valeur) ou chaque clé est un nom d'attribut et chaque valeur l'unique valeur d'attribut accepté. L'exemple du script suivant recherche dans la table produit s'il y a un produit dont la valeur d'id est fourni en premier argument ET dont le name est fourni en second argument.

```php
<?php          // prodIdName.php
require_once "bootstrap.php";  // fichier de configuration et em
$p=$entityManager->getRepository('Product')->findBy(array(
  'id' => $argv[1],
  'name' => $argv[2]
));
if (count($p)){     // au moins un produit (ici 0 ou 1 car id)
  echo "Trouvé ! (". $p[0]->getId() . ", " . $p[0]->getName() .
```

Consultation par des conjonctions d'égalité (where name='x' and id=5) II

```
    ")\n";
} else {
  echo "Aucun Produit !\n";
}
```

La méthode `findBy()` permet également :

- de fournir un ensemble de valeurs possibles pour un attribut (id IN (1,2,3)) en indiquant un tableau de valeur (array()) ;
- de trier selon une colonne (ORDER BY) (2nd paramètre) ;
- de limiter le nb de résultats (LIMIT x) (3eme paramètre) ;
- de définir un offset (résultats sauf les n premiers) (4eme paramètre) ;

L'exemple de script suivant affiche tous les produits dont le nom est 'un' ou 'deux' ou 'trois' en ordre décroissant sur les noms, croissant sur les id, en en prenant que 10 au maximum et en ne prenant pas les 2 premiers !

Consultation par des conjonctions d'égalité (where name='x' and id=5) III

```
<?php                // produndeuxtrois.php
require_once "bootstrap.php";  // fichier de configuration et em
$ps=$entityManager->getRepository('Product')->findBy(
  array('name' => array("un","deux", "trois")), // IN
  array('name' => 'DESC', 'id' => 'ASC'),       // ORDER BY
  10,              // LIMIT
  2                // OFFSET
);
if (count($ps)){                // plusieurs produits
  foreach ($ps as $p) {
    echo sprintf("%d %s\n",$p->getId(), $p->getName());
  }   //print_r($ps);
} else {
  echo "Aucun Produit !\n";
}
```


Plan

Doctrine Query Language (DQL) I

Le langage DQL ressemble au SQL mais n'est pas du SQL ! Il permet de créer une requête complexe (createQuery()) puis de l'exécuter (getResult()) en récupérant un tableau d'entités.

```
<?php                // dql.php
require_once "bootstrap.php";  // fichier de conf.
$q=$entityManager->createQuery("select p from Product p where
  p.id>8 or p.name in ('un', 'deux', 'trois') order by
  p.name asc");
$ps=$q->getResult();
if (count($ps)){                // plusieurs produits
  foreach ($ps as $p) {
    echo sprintf("%d %s\n",$p->getId(), $p->getName());
  }
} else {
  echo "Aucun Produit !\n";
}
```

Doctrine Query Language (DQL) II

On remarquera que l'on récupère une collection (sorte de tableau) d'entités (ps) selon des conditions diverses puis on la parcourt.

Plan

Associations entre entités I

- Les associations entre entités permettent de modéliser les relations existant entre les objets métiers
- Elles sont, pour certaines, liées à des contraintes d'intégrité référentielles dans la BD mais **pas toujours**
- e.g. l'héritage entre classe d'entités ne donnera pas forcément lieu à une relation dans la BD
- certaines BD n'implémentent pas certaines contraintes d'ntégrité

Quelques règles :

- une association unidirectionnelle ne possède qu'un côté propriétaire (owning side)
- une association bidirectionnelle possède un côté propriétaire (owning side) et un côté inverse

Associations entre entités II

- l'entité du côté inverse utilise l'attribut `mappedBy` pour désigner l'attribut de l'entité propriétaire
- l'entité du côté propriétaire utilise l'attribut `inversedBy` pour désigner l'attribut de l'entité inverse
- seuls les changements (ajout, suppression) côté propriétaire seront pris en compte par `doctrine` lors du `flush` : effectuer les changements des deux côtés ou bien seulement du côté propriétaire

Association unidirectionnelle "one to one" I

Cas d'utilisation : un étudiant est un utilisateur, certains utilisateurs ne sont pas étudiant

code PHP annoté (méta-données) suivi du code SQL associé :

```php
<?php              // Etudiant.php
/** @Entity **/
class Etudiant{
...
/**
* @OneToOne(targetEntity="User")
* @JoinColumn(name="user_id", referencedColumnName="id")
**/
private $user;
...
```


Association unidirectionnelle "one to one" II

```sql
CREATE TABLE Etudiant (       # SQL
...,
user_id INT DEFAULT NULL,
UNIQUE INDEX UNIQ_6FBC94267FE4B2B (user_id),
PRIMARY KEY(id)
) ENGINE = InnoDB;

CREATE TABLE User (
id INT AUTO_INCREMENT NOT NULL,
PRIMARY KEY(id)
) ENGINE = InnoDB;
ALTER TABLE Etudiant ADD FOREIGN KEY (user_id) REFERENCES
User(id);
```

Remarquons l'index **unique** créé sur la colonne `user_id`

Association bidirectionnelle "one to one" I

- Cas d'utilisation : un caddy appartient à un unique client qui ne peut en avoir qu'un au maximum
- On veut conserver du côté de l'entité client une référence au caddy et réciproquement (bidirectionnel)
- ci-dessous le code PHP annoté (méta-données) ainsi que le code SQL associé :

```php
<?php       // Customer.php
/** @Entity **/
class Customer{
// ...
/**
* @OneToOne(targetEntity="Cart", mappedBy="customer")
**/
  private $cart;
}
```


Association bidirectionnelle "one to one" II

```php
/** @Entity **/
class Cart{
// ...
/**
* @OneToOne(targetEntity="Customer", inversedBy="cart")
* @JoinColumn(name="customer_id", referencedColumnName="id")
**/
  private $customer;
}
...
```

```sql
CREATE TABLE Cart (               # SQL
  id INT AUTO_INCREMENT NOT NULL,
  customer_id INT DEFAULT NULL,
  PRIMARY KEY(id)
) ENGINE = InnoDB;
CREATE TABLE Customer (
```

Association bidirectionnelle "one to one" III

```
  id INT AUTO_INCREMENT NOT NULL,
  PRIMARY KEY(id)
) ENGINE = InnoDB;
ALTER TABLE Cart ADD FOREIGN KEY (customer_id) REFERENCES
  Customer(id);
```

- la table propriétaire de clé étrangère (`customer_id`) est le caddy
- Au niveau des entités, elles sont symétriques en possédant chacune une référence à l'autre
- Au niveau des méta-données, c'est l'attribut inversedBy qui désigne le propriétaire de clé étrangère tandis que le référencé possède un attribut "mappedBy"
- L'entité propriétaire est importante car au moment du **flush, la sauvegarde en BD se basera sur les attributs des propriétaires**

Association bidirectionnelle "one to one" IV

- Dans le cas des associations bidirectionnelles (A(b) - B(a)), la métadonnée `mappedBy` associée à b indique l'attribut de l'autre classe d'entité associée qui est source de l'association inverse (a)
- Réciproquement, la métadonnée `inversedBy` associée à a indique l'attribut (b) de l'autre classe d'entité associée qui est source de l'association inverse
- Comme dans le modèle relationnel de la BD, une seule contrainte d'intégrité référentielle existe (`Bt(a_id)->At(id)`), c'est dans les métadonnées de l'entité inversedBy (B) qu'on implantera la liaison avec la BD grâce à la métadonnées JoinColumn(name) qui désigne la colonne clé étrangère de la table (`a_id`) et JoinColumn(referencedColumnName) qui désigne la colonne cible de la référence (id)

Association unidirectionnelle many-to-one I

Plusieurs utilisateurs (many) peuvent partager la même adresse. On indique ci-dessous le code PHP annoté (méta-données) ainsi que le code SQL associé :

```
<?php            // User.php
class User{
...
/**
* @ManyToOne(targetEntity="Address")
* @JoinColumn(name="address_id", referencedColumnName="id")
**/
private $address;
}
...
```


Association unidirectionnelle many-to-one II

```
CREATE TABLE User (       # SQL
  ...,
  address_id INT DEFAULT NULL,
  PRIMARY KEY(id)
) ENGINE = InnoDB;
CREATE TABLE Address (
  id INT AUTO_INCREMENT NOT NULL,
  PRIMARY KEY(id)
) ENGINE = InnoDB;
ALTER TABLE User ADD FOREIGN KEY (address_id) REFERENCES
  Address(id);
```

Remarquons que ce cas étant extrêmement fréquent, la ligne `@JoinColumn` n'est pas nécessaire si le nommage des attributs et des colonnes suit la convention. Remarquons encore que l'adresse est optionnelle (NULL).

Association bidirectionnelle one-to-many I

- Un produit possédant plusieurs caractéristiques aura une propriété oneToMany "features" qui sera forcément en correspondance (mappedBy) par une propriété "product" manyToOne dans l'entité Feature
- Remarquons que la contrainte d'intégrité réfrentielle sera dirigée depuis feature(product_id) vers product(id)
- A l'inverse, la propriété "features" de l'entité Product sera une `collection` de caractéristiques (`Doctrine\Common\Collections\Collection`)

Association bidirectionnelle one-to-many II

```php
<?php              // Product.php, Feature.php
class Product{
  // ...
  /**
  * @OneToMany(targetEntity="Feature", mappedBy="product")
  **/
  private $features;
  // ...
public function __construct() {
  $this->features = new ArrayCollection();
  ...
class Feature{
  // ...
  /**
  * @ManyToOne(targetEntity="Product", inversedBy="features")
  * @JoinColumn(name="product_id", referencedColumnName="id")
  **/
```


Association bidirectionnelle one-to-many III

```php
private $product;
// ...
```


Association unidirectionnelle many-to-many I

- Des utilisateurs sont affiliés à différents groupes d'utilisateurs
- Dans la BD, une table de jointure (non modélisée par une entité !) permettra de représenter les associations multiples de groupes et d'utilisateurs

```php
<?php          // User.php
class User{
// ...
/**
* @ManyToMany(targetEntity="Group")
* @JoinTable(name="users_groups",
*  joinColumns={@JoinColumn(name="user_id",
*               referencedColumnName="id")},
*  inverseJoinColumns={@JoinColumn(name="group_id",
*                      referencedColumnName="id")}
* )
```

Association unidirectionnelle many-to-many II

```php
**/
  private $groups;
// ...
public function __construct() {
  $this->groups = new \Doctrine\Common\Collections\
      ArrayCollection();
}
...
class Group{
  // ...
}
```


Association unidirectionnelle many-to-many III

```sql
CREATE TABLE User (              # SQL
  id INT AUTO_INCREMENT NOT NULL,
  PRIMARY KEY(id)
) ENGINE = InnoDB;
CREATE TABLE users_groups (
  user_id INT NOT NULL,
  group_id INT NOT NULL,
  PRIMARY KEY(user_id, group_id)
) ENGINE = InnoDB;
CREATE TABLE Group (
  id INT AUTO_INCREMENT NOT NULL,
  PRIMARY KEY(id)
) ENGINE = InnoDB;
ALTER TABLE users_groups ADD FOREIGN KEY (user_id)
  REFERENCES User(id);
ALTER TABLE users_groups ADD FOREIGN KEY (group_id)
  REFERENCES Group(id);
```


Héritages I

Plusieurs héritages sont techniquement possibles grâce aux associations Doctrine :

- Mapped Superclass : permet de définir les attributs et associations communs dans une superclasse qui ne donnera pas lieu à création de table mais chaque entité fille donnera une table contenant les champs de sa superclasse
- Single Table Inheritance : dans ce patron, toute la hiérarchie est représentée dans une unique table qui contient un champ discriminant la classe. Les champs spécifiques à chaque sous-classe doivent absolument être NULL puisqu'ils existeront pour tout le monde !
- Class Table Inheritance : chaque classe fille correspondra à une table la représentant et autant de tables que nécessaire pour représenter ses ancêtres avec des clés étrangères les reliant ;

Héritages II

- Overrides : permet d'indiquer qu'un attribut ou une association de la classe fille prend le pas sur un attribut ou une association de la super-classe (valable dans le cadre de Mapped Superclass).

Remarquons encore que l'association OneToOne uni ou bi-directionnelle permet de modéliser une sorte d'héritage : cependant les attributs communs seront dupliqués dans chaque table ...

Plan

Rétro-ingéniérie : génération des Métadonnées I

Afin de fabriquer les métadonnées associées à une BD préexistante, on peut utiliser la commande doctrine afin de générer un fichier de métadonnées yml, xml ou même php (à éviter) :

```
$ vendor/bin/doctrine orm:convert-mapping --from-database yml .
```

Cela produira un fichier `Product.dcm.yml` dans le répertoire courant (.) dont le contenu suit :

```
Products:
  type: entity
  table: products
  id:
    id:
      type: integer
      nullable: false
      unsigned: false
      id: true
```


Rétro-ingéniérie : génération des Métadonnées II

```
      generator:
      strategy: IDENTITY
  fields:
    name:
      type: string
      nullable: false
      length: 255
      fixed: false
  lifecycleCallbacks: {  }
```

Attention, cette technique ne permet pas de récupérer :

- les associations inverses (mapped by) ;
- les héritages ;
- les clés étrangères qui sont aussi clés primaires ;
- les cascades ;

Rétro-ingéniérie : génération des Métadonnées III

De plus, cette technique créera des métadonnées d'entités à partir des tables de jointures (users-groups) et ne définira que des associations ManyToOne (clé étrangères).
Par conséquent, il faudra ensuite reprendre les entités créées afin qu'elles correspondent aux fonctionalités souhaitées par l'application !

Rétro-ingéniérie : retour d'expériences I

- Dans un schéma de BD dense (beaucoup de clés étrangères), doctrine va tenter de créer toutes les associations bi-directionnelles possibles ce qui crèe une erreur dans le cas où le même nom de propriété existe dans 2 tables connexes : "Property "tournoi" in "Equipe" was already declared, but it must be declared only once". En effet, Equipe est reliée à tournoi via la table inscription (en début de tournoi) mais aussi via la table classement (en fin de tournoi) ! Il faut donc renommer certaines colonnes : tournoi_id devient tournoi2_id !
- De plus, chaque table doit avoir une clé primaire
- Les tables de jointures ne donnent pas lieu à création d'entité (pas d'entité Inscription !)
- La création des méta-données en php ne permet pas de créer les entités (préférer yml)

Rétro-ingéniérie : retour d'expériences II

- Des tables de jointures possédant des colonnes ne donnent pas lieu à création d'entités donc on perd ces propriétés !

Rétro-ingéniérie : génération des entités I

Attention, le "convert-mapping" ne permet que de créer les métadonnées : afin de créer les fichiers php définissant les classes d'entités, il faut utiliser la commande "generate-entities" qui utilise les méta-données (quelque soit leur format (xml, yml, php)) et quelle que soit la façon dont elles ont été produites, manuellement ou automatiquement (convert-mapping) :

```
$ vendor/bin/doctrine orm:generate-entities src/
```

Cette commande va créer le fichier Product.php ou le modifier afin d'ajouter des attributs **privés** pour les références aux objets liés, accesseurs et mutateurs.
Encore une fois, il est préférable de créer les entités à la main si l'on veut spécifier de l'héritage entre classes ou définir des méthodes spécifiques métier.

Plan

Installation I

L'installation est réalisée via composer :

```
cd public_html/
mkdir DctExemple
cd DctExemple
curl -sS https://getcomposer.org/installer | php
xemacs composer.json
{
  "require": {
    "doctrine/orm": "2.4.*",
    "symfony/yaml": "2.*"
  },
  "autoload": {
    "psr-0": {"": "src/"}
  }
}
php composer.phar install
```


Installation II

Un répertoire `DctExemple/vendor` est créé qui contient :

`autoload.php`	qui permet le chargement automatique des classes d'entités
`bin`	qui contient des liens vers des exécutables php
`composer`	qui contient les fonctionnalités de composer
`doctrine`	qui contient les fonctionnalités de doctrine
`symfony`	qui contient les fonctionnalités de Yaml

Le répertoire `vendor/bin` contient notamment la commande doctrine. Il ne reste plus qu'à créer les fichiers `bootstrap.php` et `cli-config.php`.

bootstrap.php I

Le fichier `bootstrap.php` suivant contient 3 sections après l'utilisation des espaces de noms et le lancement du mécanisme d'autoload :

- configuration des métadonnées liant entités et tables soit en annotations PHP, soit en XML, soit en YAML (notre choix) ;
- définition de la configuration d'accès à la BD qui sera utilisée par DBAL (DataBase Abstraction Layer) puis par PDO puis par MySQL (dans notre cas) ;
- création de l'entityManager ;

bootstrap.php II

```php
<?php          // bootstrap.php
use Doctrine\ORM\Tools\Setup;
use Doctrine\ORM\EntityManager;

require_once "vendor/autoload.php";

// Create a simple "default" Doctrine ORM configuration for ...
$isDevMode = true;
//$config = Setup::createAnnotationMetadataConfiguration(array...
// or if you prefer yaml or XML
//$config = Setup::createXMLMetadataConfiguration(array(__DIR ...
//METADATA YAML
$config = Setup::createYAMLMetadataConfiguration(
  array(__DIR__."/config/yaml"), $isDevMode);

// database configuration parameters
$dsn =  array(
```

bootstrap.php III

```
  'dbname' => 'tempo',
  'user' => 'tempo',
  'password' => 'tempo',
  'host' => 'localhost',    // '127.0.0.1' ne fonctionne pas !
  'driver' => 'pdo_mysql',
  'charset' => 'utf8',  // sinon les char utf-8 ne passent pas en BD
);
// obtaining the entity manager
$entityManager = EntityManager::create($dsn, $config);
```


Ligne de commande doctrine I

Pour exécuter les commandes `doctrine`, il est nécessaire de définir un fichier de configuration de l'interpréteur en ligne de commande désigné par `cli-config.php` :

```
<?php              // cli-config.php
require_once "bootstrap.php";
return \Doctrine\ORM\Tools\Console\ConsoleRunner::
  createHelperSet($entityManager);
```


Bugs et astuces (MAMP spécifique) I

Il faut créer un lien symbolique vers la socket mysql de MAMP
depuis le répertoire où l'interprète en ligne de commande l'attend (`php -i`).
`$ sudo ln -s /Applications/MAMP/tmp/mysql/mysql.sock /var/mysql/mysql.so`
On pourrait également modifier le `php.ini` !

Ligne de commande I

Attention à distinguer l'interprète php de la ligne de commande (celui qui exécute les commandes doctrine) et celui du serveur Apache : ils n'ont pas forcément la même version ni les mêmes modules installés et pas le même php.ini !

- `which php` renverra `/usr/bin/php`
- `php -i` affichera les infos (`phpinfo()`)
- `php -ini` indique la localisation du php.ini (`/etc/`)
- `php -interactive` permet de tester après avoir tapé `<?` et terminé par Ctrl-D ;

Plan

DBAL DataBase Access Layer I

Doctrine 2 fournit DBAL qui est la couche entre l'ORM et PDO. DBAL permet notamment de contrôler les limites d'une transaction si l'on veut être plus précis qu'avec l'utilisation de la méthode `flush()` pour valider ses modifications.

```php
<?php          // transaction.php
// $em instanceof EntityManager
$em->getConnection()->beginTransaction(); // suspend auto-commit
try {
  //... do some work
  $user = new User;
  $user->setName('George');
  $em->persist($user);
  $em->flush();
  $em->getConnection()->commit();
} catch (Exception $e) {
  $em->getConnection()->rollback();
```


DBAL DataBase Access Layer II

```
  throw $e;
}
```

Remarquons que dans cet exemple, c'est le `beginTransaction()` qui supprime le mode auto-commit par défaut de PDO. Par conséquent, ensuite le `flush` ne suffira pas pour valider et il faudra le `commit`. Enfin, DBAL permet également de manipuler des verrous (lock) sur les tables.

Conclusion I

- L'utilisation d'un ORM tel que Doctrine ne peut être envisagé que dans le cadre d'un développement à plein temps d'un projet
- En effet, le nombre de concepts employés ainsi que le vocabulaire utilisé nécessite un long temps d'apprentissage
- Intégré dans Symfony 2, un framework PHP complet, Doctrine 2 semble avoir pris le pas sur les autres ORMs PHP
- De plus, ses concepts proviennent d'ORM leaders dans d'autres langages (Hibernatus en Java) et son apprentissage restera utile

Plan

Une étude de cas Doctrine I

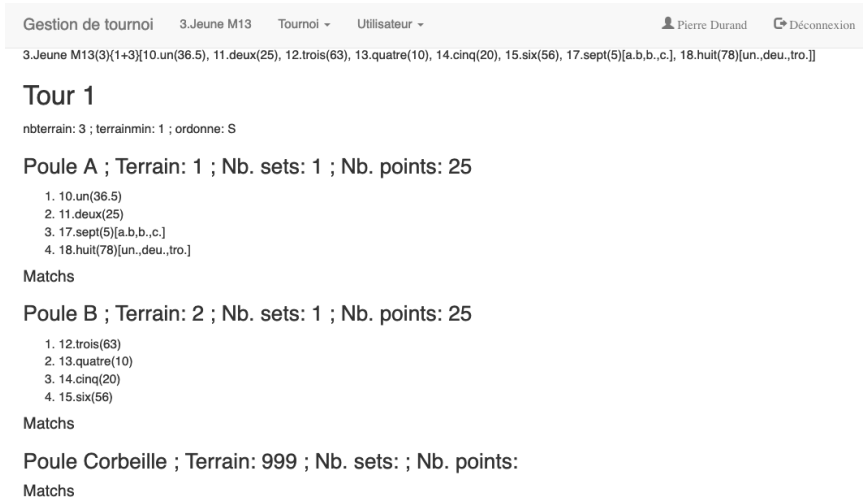
- on souhaite prototyper la gestion de tournois de volley à l'aide de Doctrine
- l'accent sera mis sur les fonctionalités plus que sur la charte graphique
- on utilisera les **annotations PHP** pour la définition des métadonnées our éviter YAML
- on utilisera le patron d'architecture MVC
- framework Bootstrap pour la mise en page sommaire

Plan

Architecture de l'application I

- MVC : le Modèle sera constitué des entités PHP définies dans le répertoire `src/`
- les contrôleurs et les vues seront situées dans les répertoires `Utilisateur/` et `Tournoi/` qui correspondent à la gestion des utilisateurs du site et à la gestion du tournoi
- dans le répertoire racine de l'application, on trouvera :
  - `index.php` l'unique page du site qui chargera des contrôleurs et des vues par `include` en fonction des paramètres
  - `bootstrap.php` le fichier de configuration de Doctrine accompagné de `cli-config.php` pour la ligne de commande
  - `composer.json` pour l'installation de doctrine
  - `vendor/` pour les paquets installés par composer

index.php I



index.php II

```php
<?php
ini_set('display_errors', 1);
error_reporting(E_ALL); // en mode développement !

require_once "bootstrap.php";          // config dct et autoload
session_start();                  // session PHP

if (!empty($_SESSION['userid'])){
  $user = $em->find("Utilisateur",$_SESSION['userid']);
}
if (!empty($_SESSION['tournoiid'])){
  $tournoi = $em->find("Tournoi",$_SESSION['tournoiid']);
}
?>
<!doctype html>
<html lang="fr">
<head>
```


index.php III

```
  <meta charset="utf-8" />
  <script src="global.js"></script>
  <link rel="stylesheet" href="https://maxcdn.bootstrap ...

  <meta name="viewport" content="width=device-width,
↪   initial-scale=1">

  <title>Gestion de tournoi</title>
</head>
<body style="padding-top: 50px; ">
  <div class="navbar navbar-default navbar-fixed-top">
    <div class="container" id="menu">
      <?php include_once("menu.php"); ?>
    </div>
  </div>
  <div class="container" id="principale">
    <?php
```


index.php IV

```
    include_once("principale.php");
    $em = NULL; // désallocation
    ?>
  </div>
</body>
</html>
```

Plan

Les utilisateurs : le modèle I

3 types d'utilisateurs : anonyme, administrateur, gestionnaire

```php
<?php
use Doctrine\ORM\Mapping as ORM;
/**
  * Utilisateur
  * @Entity
  * @Table(uniqueConstraints={@UniqueConstraint(name="login_idx",
↪   columns={"login"})})
  */
class Utilisateur{
  /**
    * @var integer
    * @Id
    * @Column(type="integer", nullable= false,
↪   options={"unsigned":true})
    * @GeneratedValue
```


Les utilisateurs : le modèle II

```php
    */
  protected $id;
  /**
    * @var string
    * @Column(type="string",length=20,nullable=false,
↪   options={"fixed":false})
  */
  protected $login;
  /**
    * @var string
    * @Column(type="string",length=50,nullable=false,
↪   options={"fixed":false})
    */
  protected $password;
  /**
    * @var string
    * @Column(type="string",length=10,nullable=false,
↪   options={"fixed":false})
```


Les utilisateurs : le modèle III

```php
    */
  protected $type;
  /**
    * @var \Evenement
    * @ManyToMany(targetEntity="Evenement",
↪   inversedBy="utilisateurs")
    * @JoinTable(name="utilisateurs_evenements")
    */
  protected $evs;
  /**
    * Get id
    * @return integer
    */
  public function getId(){
      return $this->id;
  }
  /**
```

Les utilisateurs : le modèle IV

```
   * Set login
   * @param string $login
   * @return Utilisateur
   */
public function setLogin($login){
    $this->login = $login;
    return $this;
}
/**
   * Get login
   * @return string
   */
public function getLogin(){
    return $this->login;
}
/**
   * Add ev
```


Les utilisateurs : le modèle V

```
   * @param \Evenement $ev
   * @return Utilisateur
   */
public function addEv(\Evenement $ev){
    $this->evs[] = $ev;
    return $this;
}
/**
   * Remove ev
   * @param \Evenement $ev
   */
public function removeEv(\Evenement $ev){
    $this->evs->removeElement($ev);
}
/**
   * Get evs
   * @return \Doctrine\Common\Collections\Collection
```


Les utilisateurs : le modèle VI

```
   */
public function getEvs(){
    return $this->evs;
}
/**
* Constructor
*/
public function __construct(){
  $this->evs = new
  ↪  \Doctrine\Common\Collections\ArrayCollection();
}
/**
* @return string représentation chaîne d'un Utilisateur
*/
public function __toString(){
    return $this->getId()."-".$this->getPrenom()."
    ↪  ".$this->getNom(); }}
```


Les Evénements : le modèle I

Les événements sont gérés par des U. gestionnaires et contiennent des tournois

```
<?php
use Doctrine\ORM\Mapping as ORM;
/**
  * Evenement
  * @Entity
  */
class Evenement{
  /**
    * @var integer
    * @Id
    * @Column(type="integer", nullable=false)
    * @GeneratedValue
    */
  protected $id;
```

Les Evénements : le modèle II

```php
/**
  * @var string
  * @Column(type="string", length=30, nullable=false,
↪  options={"fixed":false})
  */
protected $nom;
/**
  * @var \Doctrine\Common\Collections\Collection
  * @OneToMany(targetEntity="Tournoi", mappedBy="ev")
  */
protected $tournois;
/**
  * @var \Doctrine\Common\Collections\Collection
  * @ManyToMany(targetEntity="Utilisateur", mappedBy="evs")
  * @JoinTable(name="utilisateurs_evenements")
  */
protected $utilisateurs;
```


Les Evénements : le modèle III

```php
/**
  * Get id
  * @return integer
  */
public function getId(){
    return $this->id;
}
/**
  * Set date
  * @param \DateTime ou string $date
  * @return Evenement
  */
public function setDate($date){
  if (is_string($date)){
    $date = DateTime::createFromFormat('d/m/Y', $date,new
    ↪  DateTimeZone('Europe/Paris'));  //$date="01/02/3000"
  }
```


Les Evénements : le modèle IV

```php
  if (is_a($date, "DateTime")){
    if (checkdate($date->format('m'), $date->format('d'),
    ↪  $date->format('Y'))) { // Valide
      $this->date = $date;
      return $this;
    }
  } else {
    throw new Exception("Date invalide : " .
    ↪  var_export($date,TRUE));
  }
}
/**
  * Add tournoi
  * @param \Tournoi $tournoi
  * @return Evenement courant
  */
public function addTournoi(\Tournoi $tournoi){
```


Les Evénements : le modèle V

```php
  $this->tournois[] = $tournoi;
  return $this;
}

/**
* constructeur
* @author Michel Meynard
*/
public function __construct() {
  $this->tournois = new
  ↪  \Doctrine\Common\Collections\ArrayCollection();
  $this->utilisateurs = new
  ↪  \Doctrine\Common\Collections\ArrayCollection();
}
/**
* @return string représentant l'événement
*/
```

Les Evénements : le modèle VI

```
public function __toString(){
  return $this->getId()." ".$this->getNom()."
  ↪  (".$this->getDate()->format('d/m/Y').")";}}
```


Plan

la ligne de commande Doctrine : création BD I

Une fois construites les 12 entités dans le répertoire src (Tournoi, Tour, Poule, ...), on peut créer le script sql de création de la BD :

```
php vendor/bin/doctrine orm:schema-tool:create --dump-sql

The following SQL statements will be executed:

CREATE TABLE Utilisateur (id INT UNSIGNED AUTO_INCREMENT NOT
↪  NULL, nom VARCHAR(30) NOT NULL, prenom VARCHAR(30) NOT NULL,
↪  login VARCHAR(20) NOT NULL, password VARCHAR(50) NOT NULL,
↪  email VARCHAR(50) NOT NULL, type VARCHAR(10) NOT NULL, UNIQUE
↪  INDEX login_idx (login), PRIMARY KEY(id)) DEFAULT CHARACTER
↪  SET utf8 COLLATE `utf8_unicode_ci` ENGINE = InnoDB;

CREATE TABLE utilisateurs_evenements (utilisateur_id INT UNSIGNED
↪  NOT NULL, evenement_id INT NOT NULL, INDEX
↪  IDX_6B9CBF92FB88E14F (utilisateur_id), INDEX
↪  IDX_6B9CBF92FD02F13 (evenement_id), PRIMARY
↪  KEY(utilisateur_id, evenement_id)) DEFAULT CHARACTER SET utf8
↪  COLLATE `utf8_unicode_ci` ENGINE = InnoDB;
```


la ligne de commande Doctrine : création BD II

```
CREATE TABLE Classement (eq_id INT UNSIGNED NOT NULL, tour_id INT
↪  UNSIGNED NOT NULL, rang INT NOT NULL, INDEX
↪  IDX_1AB39EBDDD73D4FB (eq_id), INDEX IDX_1AB39EBD15ED8D43
↪  (tour_id), PRIMARY KEY(eq_id, tour_id)) DEFAULT CHARACTER SET
↪  utf8 COLLATE `utf8_unicode_ci` ENGINE = InnoDB;

CREATE TABLE Evenement (id INT AUTO_INCREMENT NOT NULL, nom
↪  VARCHAR(30) NOT NULL, date DATE DEFAULT '0000-00-00' NOT
↪  NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8 COLLATE
↪  `utf8_unicode_ci` ENGINE = InnoDB;
CREATE TABLE Joueur (id INT UNSIGNED AUTO_INCREMENT NOT NULL,
↪  eq_id INT UNSIGNED DEFAULT NULL, nom VARCHAR(30) DEFAULT
↪  NULL, prenom VARCHAR(30) NOT NULL, email VARCHAR(30) DEFAULT
↪  NULL, mobile VARCHAR(15) DEFAULT NULL, INDEX
↪  IDX_FADDACF3DD73D4FB (eq_id), PRIMARY KEY(id)) DEFAULT
↪  CHARACTER SET utf8 COLLATE `utf8_unicode_ci` ENGINE = InnoDB;
```

la ligne de commande Doctrine : création BD III

```
ALTER TABLE utilisateurs_evenements ADD CONSTRAINT
↪ FK_6B9CBF92FB88E14F FOREIGN KEY (utilisateur_id) REFERENCES
↪ Utilisateur (id) ON DELETE CASCADE;
ALTER TABLE utilisateurs_evenements ADD CONSTRAINT
↪ FK_6B9CBF92FD02F13 FOREIGN KEY (evenement_id) REFERENCES
↪ Evenement (id) ON DELETE CASCADE;

ALTER TABLE Tournoi ADD CONSTRAINT FK_D712E04340A4EC42 FOREIGN
↪ KEY (ev_id) REFERENCES Evenement (id);
```

Remarquons la création de la table `utilisateurs_evenements` qui ne correspond pas à une entité mais à une association many-to-many ...

Vérification des entités I

- On peut créer la BD en lançant la commande précédente sans l'option `dump-sql`
- Ensuite, lors du développement on peut modifier les entités et demander la mise à jour de la BD : `php vendor/bin/doctrine orm:schema-tool:update`
- Cependant, il vaut mieux auparavant :
  - sauvegarder la BD actuelle et ses données
  - vérifier la correction des entités : `php vendor/bin/doctrine orm:info`
  - vérifier la synchronisation de la BD avec les entités : `php vendor/bin/doctrine orm:validate-schema`
- Bien entendu, pour que ces commandes fonctionnent il faut avoir défini le fichier de configuration de la ligne de commande doctrine :

Vérification des entités II

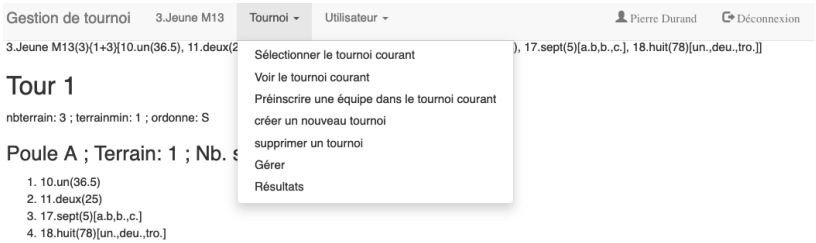
```php
<?php    // cli-config.php
require_once "bootstrap.php";
return \Doctrine\ORM\Tools\Console\ConsoleRunner
↪  ::createHelperSet($entityManager);
```


Plan

Contrôleurs sans Symfony I

- Sans utiliser Symfony ni Twig, on peut utiliser Doctrine pour développer des pages web
- Quelques exemples de scripts caractéristiques de l'utilisation de l'ORM suivent avec leur visuel si nécessaire

Le menu d'entête I




```php
<?php // menu.php
/** Ecrit un menu déroulant en  bootstrap css
* @param h le titre du menu
* @param items les Permissions (options de menu)
*/
function ecrireMenu($h, $items){
  echo "<li class='dropdown'><a href='' class='dropdown-toggle'
  ↪  data-toggle='dropdown'>$h <b class='caret'></b></a> <ul
  ↪  class='dropdown-menu'>\n";
```


Le menu d'entête II

```php
  foreach($items as $p){ // pour chaque permission (Tournoi,
  ↪  select)
    echo '<li><a
    ↪  href="?rubrique=',$p->getRubrique(),'&action=',$p->getAction(),'">',
    ↪  $p->getTexte(),"</a></li>\n";
  }
echo "</ul></li>\n";
}
?>

<div class="navbar-header"><!-- en-tête permettant la réduction
↪  sur mobile -->
  <button type="button" class="navbar-toggle"
↪  data-toggle="collapse"
    data-target=".navbar-collapse"> <!-- bouton d'expansion du
↪  menu -->
  <span class="icon-bar"></span>  <!-- composé de 3 tirets
↪  horizontaux -->
```


Le menu d'entête III

```php
  <span class="icon-bar"></span>
  <span class="icon-bar"></span>
  </button>
  <a class="navbar-brand"
↪  href="?rubrique=Utilisateur&action=bienvenue">Gestion de
↪  tournoi</a>
</div>
<div class="navbar-collapse collapse">
  <ul class="nav navbar-nav">

<?php
if(!empty($tournoi)){ // tournoi courant
  echo '<li><p class="navbar-text">' .
  ↪  substr(htmlentities($tournoi->getNom()),0,15)."</p></li>\n";
  }
// création des menus en fonction des permissions et de l'user
↪  connecté
```

Le menu d'entête IV

```php
$q=$entityManager->createQuery('SELECT p FROM Permission p ORDER
↪  BY p.id ASC');
$ps=$q->getResult();
$rub=""; // rubrique courante
$items=array();                          // items de menu
$type=(isset($user) ? $user->getType() : 'x'); // x|g|a
// type d'utilisateur (x si anonyme)
foreach($ps as $p){
  if ($rub==""){ // toute première fois !
    $rub=$p->getRubrique();
  }
  if ($rub==$p->getRubrique()){  // on reste dans la même
  ↪  rubrique
    if (strpos($p->getType(),$type)!==FALSE){ // alors ajouter la
    ↪  permission
        $items[]=$p;
     }
```


Le menu d'entête V

```php
  } else { // changement de rubrique
     if  (count($items)){ // si items non vide écrire le menu :
     ↪  entête $rub, les items
       ecrireMenu($rub,$items); // fon produisant le code
       ↪  bootsstrap
     }
     $rub=$p->getRubrique();
     $items=array();
     if (strpos($p->getType(),$type)!==FALSE){ // alors ajouter
     ↪  la permission
       $items[]=$p;
     }
  }
}
if  (count($items)){ // dernière rubrique : écrire le menu
  ecrireMenu($rub,$items); // fon produisant le code bootsstrap
}
```

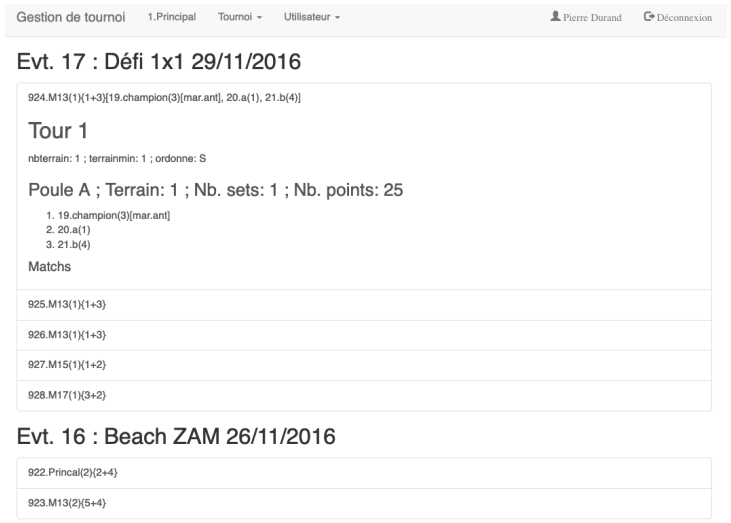

Le menu d'entête VI

```php
//  CONnexion ou BONjour
echo "</ul>\n",'<ul class="nav navbar-nav navbar-right"
↪  id="conbon">',"\n" ;

if(!isset($_SESSION['userid']) || (isset($_GET['action']) &&
↪  $_GET['action']=="deconnexion")){  // ANONYME ou déconnexion
↪  ------------------------
  echo '<li><a
  ↪  href="?rubrique=Utilisateur&action=connexion"><span
  ↪  class="glyphicon glyphicon-log-in">
  ↪  Connexion</span></a></li>',"\n";
} else {                        // Connecté
↪  -------------------------
  echo '<li><p class="navbar-text"><span class="glyphicon
  ↪  glyphicon-user">'," ",htmlentities($user->getPrenom()).'
  ↪  '.htmlentities($user->getNom())."</span></p></li>\n";
```


Le menu d'entête VII

```php
  echo '<li><a
  ↪  href="?rubrique=Utilisateur&action=deconnexion"><span
  ↪  class="glyphicon glyphicon-log-out">
  ↪  Déconnexion</span></a></li>',"\n";
}
?>
</ul>
</div><!--/.nav-collapse -->
```

Sélection d'un tournoi I



Sélection d'un tournoi II

```
<form class="form-horizontal" role="form" method="post">
<h1>Sélection d'un tournoi</h1>

<?php
$evs=$entityManager->getRepository('Evenement')->findAll();
// $q=$entityManager->createQuery('SELECT e FROM Evenement e
↪  ORDER BY e.date DESC');
// $evs=$q->getResult();

foreach($evs as $ev){ // pour chaque événement
  echo '<h2>';
  echo "Evt. ".$ev->getId()," : ",$ev->getNom()."
  ↪  ",$ev->getDate()->format("d/m/Y"),"</h2>";
  if(!$ev->getTournois()->isEmpty()){
    echo '<div class="list-group">';
    foreach($ev->getTournois() as $t){ // pour chaque tournoi
      echo '<button type="submit" name="id" value="',$t->getId(),
      ↪  '" class="list-group-item">';
```


Sélection d'un tournoi III

```
      echo $t,"</button>";
    }
    echo '</div>';
  }// fin du if
} // fin foreach
echo '</form>';
?>
```


Conclusion I

- Doctrine permet d'écrire un code métier en se préoccuppant moins de la couche Modèle
- On écrit en PHP avec un peu de DQL
- Utiliser Symfony comme framework global est une suite logique car il ne reste plus qu'à étudier Twig comme moteur de template
- La compréhension de Doctrine est un processus plus ou moins rapide mais une fois acquis, cela permet de développer un code métier spécialisé dans les entités
- Seule la persistance des entités doit être gérée dans les contrôleurs (persist , flush)

Plan

Plan

Introduction I

- Les "ressources web" ont été définies pour la première fois sur le World Wide Web comme des documents ou des fichiers identifiés par leur URL
- Cependant, elles ont aujourd'hui une définition beaucoup plus générique et abstraite qui inclut toute chose ou entité pouvant être identifiée, nommée, adressée ou gérée d'une façon quelconque sur le web
- Dans un service web REST, les requêtes effectuées sur l'URI d'une ressource produisent une réponse qui peut être en **HTML, XML, JSON** ou un autre format
- La réponse confirme que la ressource stockée a été altérée et elle peut fournir des liens hypertextes vers d'autres ressources ou collection de ressources liées

Introduction II

- Lorsque le protocole HTTP est utilisé, comme c'est souvent le cas, les opérations disponibles sont POST, GET, PUT, DELETE ou d'autres méthodes HTTP
- REST (representational state transfer) est un style d'architecture définissant un ensemble de contraintes et de propriétés basées sur le protocole HTTP
- Les services web conformes au style d'architecture REST, aussi appelés services web RESTful, établissent une interopérabilité entre les ordinateurs sur Internet
- Les services web REST permettent aux systèmes effectuant des requêtes d'accéder et de manipuler des représentations textuelles de ressources web à travers un jeu d'opérations uniformes et prédéfinies **sans état**

Introduction III

- D'autres types de services web tels que les services web SOAP exposent leurs propres jeux d'opérations arbitraires.
- Avec l'utilisation d'un protocole sans état et d'opérations standards, les systèmes REST visent la réactivité, la fiabilité et l'extensibilité, par la réutilisation de composants pouvant être gérés et mis à jour sans affecter le système global, même pendant son fonctionnement
- Le terme REpresentational State Transfer (REST) a été défini pour la première fois en 2000 par Roy Fielding dans le chapitre 5 de sa thèse de doctorat
- La thèse de Fielding a expliqué les principes de REST auparavant connus comme le "modèle objet de HTTP" depuis 1994 et qui ont été utilisés dans l'élaboration des standards HTTP 1.1 et URI

Introduction IV

- Le terme est censé évoquer comment une application web bien conçue se comporte : c'est un réseau de ressources (une machine à états virtuelle) au sein duquel l'utilisateur évolue en sélectionnant des liens, tels que /utilisateur/tom, et des opérations telles que GET ou DELETE (des transitions d'état), provoquant le transfert de la ressource suivante (représentant le nouvel état de l'application) vers l'utilisateur pour être utilisée

Les opérations CRUD (Create, Retrieve, Update, Delete) sur les ressources du service sont matérialisées par les méthodes HTTP :

- Créer (create) : POST
- Rechercher (retrieve) : GET
- Mettre à jour (update) : PUT
- Supprimer (delete) : DELETE

Plan

Installation I

"Slim skeleton application" s'installe en utilisant `Composer` :

```
$ php composer.phar create-project slim/slim-skeleton monapp
```

Le répertoire `monapp` de l'application est créé qui contient des sous-répertoires :

- vendor : contient les paquetages dont dépend slim ;
- logs : les journaux générés par slim ;
- tests : les tests utilisant PHPUnit ;
- public : contenant un fichier `.htaccess` utilisant le module `mod_rewrite` qui permet la redirection des URLs (Attention à la directive de configuration globale d'Apache : `AllowOverride All`) , et un fichier `index.php` initial ;

Installation II

Puis, on lance la ligne de commande suivante afin de lancer un serveur http (interne à l'interprète php) qui va écouter sur le port 8080 et qui a comme racine (DocumentRoot) le répertoire monapp/public :

```
$ cd monapp
$ php composer.phar start
> php -S localhost:8080 -t public
```

Afin de vérifier le bon fonctionnement de monapp, il suffit de se rendre sur son navigateur à l'URL : http://localhost:8080/ pour voir afficher une page d'accueil de Slim qui est définie dans monapp/templates/index.phtml :

Installation III

```
<h1>Slim</h1>
  <div>a microframework for PHP</div>

  <?php if (isset($name)) : ?>
    <h2>Hello <?= htmlspecialchars($name); ?>!</h2>
  <?php else: ?>
    <p>Try <a href="http://www.slimframework.com">
      SlimFramework</a></p>
   <?php endif; ?>
```

Si l'on se rend à l'URL : http://localhost:8080/dupont, on verra alors s'afficher le message <h2>Hello dupont</h2> à la place du lien.
Si on utilise un serveur Apache sur le port 8888 (xAMP), on peut lancer à la ligne de commande :

Installation IV

```
$ curl http://localhost:8888/TestSlim/monapp/public/Michel
...
<h2>Hello Michel!</h2>
...
```

Remarquons que l'objectif d'une API REST n'est absolument pas d'afficher des messages mais d'effectuer des actions et de retourner des données souvent au format JSON. Etudions l'exemple fourni par défaut dans le projet Slim.

Plan

Le fichier public/index.php I

- Ce fichier est celui qui est lancé dans l'exemple précédent
- Il crèe une instance d'application Slim avec une configuration, puis il charge :
- les dépendances (rendu HTML avec template et journalisation),
- le middleware qui peut effectuer des traitements intermédiaires sur la requête (vérifier l'authentification ou décoder un objet JSON)
- et enfin les routes qui définissent les requêtes disponibles dans l'API Rest.

Le fichier public/index.php II

```php
 <?php            // index.php
require __DIR__ . '/../vendor/autoload.php';

session_start();

// Instantiate the app
$settings = require __DIR__ . '/../src/settings.php';
$app = new \Slim\App($settings);

// Set up dependencies
require __DIR__ . '/../src/dependencies.php';

// Register middleware
require __DIR__ . '/../src/middleware.php';

// Register routes
require __DIR__ . '/../src/routes.php'; // on y définit l'API
↪  REST
```


Le fichier public/index.php III

```php
// Run app
$app->run();
```


Plan

Les routes I

Visualisons le fichier `src/routes.php` puis nous l'analyserons afin de découvrir l'API de la plateforme Slim :

```php
<?php                   // routes.php
use Slim\Http\Request;
use Slim\Http\Response;

// Routes
$app->get('/[{name}]', function (Request $request,
  Response $response,
  array $args) {
    // Sample log message
    $this->logger->info("Slim-Skeleton '/' route");

    // Render index view
```


Les routes II

```
    return $this->renderer->render($response, 'index.phtml',
    ↪  $args);
});
```

Dans cet exemple simplissime, à la route `/Michel` on associe une fonction de `callback` qui retournera une réponse HTML créée par le rendu sur le modèle (template) `index.phtml` en utilisant les arguments (`name`) et la réponse éventuellement modifiée par les dépendances et le middleware.
/par
Suit le modèle de rendu ou la variable `$name` sera instancié par la fonction de callback précédente

Les routes III

```php
<?php              // template de rendu
<body>
  <h1>Slim</h1>
    <div>a microframework for PHP</div>
    <?php if (isset($name)) : ?>
            <h2>Hello <?= htmlspecialchars($name); ?>!</h2>
    <?php else: ?>
            <p>Try <a href="http://www.slimframework.com">
              SlimFramework</a></p>
    <?php endif; ?>
    </body>
```


Plan

Architecture de Slim I

C'est dans le fichier `src/routes.php` qu'on définit notre API à l'aide de :

- la classe `Http\Request` qui modélise une requête HTTP (instances immutables) possède :
  - une méthode avec des fonctions membres comme `getMethod()` ou `isPost()`. Les méthodes HTTP sont nombreuses : GET, POST, PUT, DELETE, HEAD, PATCH, OPTIONS.
  - une URI avec des méthodes `getHost()`, `getPort()`, `getPath()`, ...
  - des headers HTTP avec `getHeaders()`.
  - un body qui peut contenir du XML ou du JSON : on utilisera alors `getParsedBody()` afin de récupérer un `SimpleXMLElement` ou un simple tableau associatif ...
  - des Files chargés (upload) avec `getClientFilename()`, `getSize()`
  - des paramètres passés `getQueryParam('nom','défaut')`, `getServerParam()`, `getCookieParam()`

Architecture de Slim II

- la classe `Http\Response` qui modélise une réponse HTTP (instances immutables) possède :
  - un status code qui vaut 200 si OK, 404 pour une page non trouvée peut être positionné par la méthode `withStatus(302)` qui retourne une copie de l'instance avec un code en plus ;
  - des headers modifiables avec `withHeader('Content-type', 'application/json')` ;
  - un body qui est un objet de type flot (stream) et dans lequel on peut écrire à sa guise :
    `$body = $response->getBody();$body->write('Hello');` On peut retourner du JSON ainsi qu'un status code (200) ainsi :

```php
<?php
$data = array('name' => 'Bob', 'age' => 40);
$newResponse = $oldResponse->withJson($data);
```

    Le `Content-type` de la réponse sera automatiquement : `application/json;charset=utf-8`

Plan

Création de routes I

C'est le travail essentiel afin de construire notre API rest. Il suffit, comme dans l'exemple précédent, d'appliquer des méthodes PHP correspondant à chaque méthode HTTP sur l'instance de l'application Slim. La plupart de ces méthodes ont deux paramètres :

- le modèle de route avec des paramètres nommés éventuellement optionnels ;
- une fonction de callback qui a 3 paramètres :
  - la requête HTTP
  - le réponse HTTP par défaut qu'il peut enrichir avec les méthodes `with`
  - les valeurs des paramètres nommés stockés dans un tableau associatif dont les clés sont les noms de paramètres

  la fonction de callback retourne une réponse HTTP ou peut simplement faire des `echo`

Dans le code suivant, le modèle de route peut contenir de 0 à 2 paramètres nommés `year` et `month` :

Création de routes II

```php
<?php
$app->get('/news[/{year}[/{month:[0-9]+}]]',
          function ($request, $response, $args) {
    // reponds to `/news`, `/news/2016` and `/news/2016/03`
});
```

Remarquons que les crochets indiquent l'optionalité et que le deux-points impose un test avec une expression régulière.

Plan

Un exemple simple de bibliothèque I

Créons un formulaire HTML pour manipuler des livres qui seront stockés dans le tableau de session PHP (plutôt qu'une BD). Chaque livre sera composé de son indice entier (id) et d'un titre. On pourra :

- voir tous les livres grâce à : `get /books`
- ajouter un livre (titre uniquement) : `post /books` avec un paramètre titre
- supprimer un livre par son indice : `delete /book/12`
- mettre à jour un livre identifié par son indice auquel on affectera un nouveau titre : `put /book/13/Les misérables`

Un exemple simple de bibliothèque II

```html
<h2>add book</h2>
<form method="post" action="./books/">
  <input type="text" name="titreadd" placeholder="titre">
  <input type="submit" value="Add">
</form>
 <h2>delete book</h2>
<form method="post" id="formdel" action onclick='actdel();'>
  <input type="hidden" name="_METHOD" value="DELETE">
  <input type="number" id="numdel"  name="numdel"
    placeholder="numéro">
  <input type="submit" value="Delete">
</form>
<h2>put book</h2>
<form id="formput" method="post" action onclick='actput();'>
  <input type="hidden" name="_METHOD" value="PUT">
  <input type="number" id="numput"  name="numput"
    placeholder="numéro">
```

Un exemple simple de bibliothèque III

```
  <input type="text" id="titreput" name="titreput"
    placeholder="titre">
  <input type="submit" value="PUT">
</form>
```

Les 2 fonctions javascript sont nécessaires afin de positionner dynamiquement l'action du formulaire :

```
<script>
function actdel(){
  document.getElementById("formdel").action=
    "book/"+document.getElementById("numdel").value;
}
function actput(){
  document.getElementById("formput").action=
    "book/"+document.getElementById("numput").value+"/"+
    document.getElementById("titreput").value;
```


Un exemple simple de bibliothèque IV

```
}
</script>
```

Créons une nouvelle vue `template/books.phtml` qui affiche la liste des titres modifiés grâce au précédent formulaire :

```
<h1>Books</h1>
<div style="text-align:left;margin:30px;">
    <?php if (isset($_SESSION['books'])) :
                foreach($_SESSION['books'] as $key => $book): ?>
              <h2> <?= $key, ' ', htmlspecialchars($book)
↪   ?></h2>
            <?php endforeach;
                  else: ?>
              <h2>No book !</h2>
        <?php endif; ?>
</div>
```


Un exemple simple de bibliothèque V

Remarquons une syntaxe PHP des structures de contrôle qui évite les accolades et la balise <?= qui permet d'afficher le contenu d'une variable. Enfin, les deux routes permettant soit d'afficher le contenu de la bibliothèque (contenu dans la variable de session), soit d'ajouter un nouveau titre.

```
<?php
$app->get('/books/', function (Request $request,
          Response $response, array $args) {
  // Sample log message
  $this->logger->info("Slim-Skeleton '/' route");

  // Render index view
  return $this->renderer->render($response, 'books.phtml',
  ↪  $args);
});
```


Un exemple simple de bibliothèque VI

```
$app->post('/books/', function (Request $request,
        Response $response, $args) {
    // Sample log message
    $this->logger->info("Slim-Skeleton '/' route");
    if(isset($_SESSION['books'])){
      $_SESSION['books'][]=$request->getParam('titre'); // _POST
    }else{
      $_SESSION['books']=array();
    }
    // Render index view
    return $this->renderer->render($response, 'books.phtml',
    ↪  $args);
});
```

Un exemple simple de bibliothèque VII

Enfin, une route DELETE est ajoutée et un formulaire de suppression est créé. Notons que HTML ne permet que les méthodes GET et POST mais avec un subterfuge utilisant un champ caché, on peut écraser le nom de la méthode.

```
<h1>delete book</h1>
<form method="post" action="index.php/books/">
  <input type="hidden" name="_METHOD" value="DELETE">
  <input type="text" name="titre" placeholder="titre">
  <input type="submit" value="DELETE">
</form>
<?php
$app->delete('/books/', function (Request $request,
              Response $response, $args) {
    // Sample log message
    $this->logger->info("Slim-Skeleton '/' route");
    if(isset($_SESSION['books'])){
```


Un exemple simple de bibliothèque VIII

```
        $key=array_search($request->getParam('titre'),
                          $_SESSION['books']);
        if($key!==FALSE){
              unset($_SESSION['books'][$key]); // DELETE
        }
      } else {
        $_SESSION['books']=array();
      }
      // Render index view
      return $this->renderer->render($response,
                                'books.phtml', $args);
});
```

Bien entendu, d'autres méthodes PUT, GET books/id pourraient être utilisées afin d'étoffer notre API. De plus le stockage dans une base de données via PDO semble plus pertinent que dans une variable de session !

Plan

Une vraie API Rest I

Afin de réaliser une vraie API Rest, nous allons :

- fournir comme réponse, uniquement du JSON (page jbooks) et pas du HTML
- émettre les requêtes par l'intermédiaire de requêtes Ajax construites en JavaScript (JQuery)
-

Ajouter un titre I

La page HTML `jscrud.html` contient 3 formulaires permettant d'envoyer des requêtes HTTP à notre application.
Le premier permet d'émettre une requête POST qui va ajouter un titre à la bibliothèque :

```
<h2>add book</h2>
<form method="post" action="./jbooks/">
  <input type="text" name="titreadd" placeholder="titre">
  <input type="submit" value="Add (POST)">
</form>
...
```

regardons la route associée :

Ajouter un titre II

```
<?php // add par post et param
$app->post('/jbooks/', function (Request $request, Response
↪  $response, $args) {
  // Sample log message
  $this->logger->info("Slim-Skeleton '/' route");
  if(!isset($_SESSION['books'])){
    $_SESSION['books']=array();
  }
  $_SESSION['books'][]=$request->getParam('titreadd'); // POST
  ↪  par Param
  // Return en json
  end($_SESSION['books']); // pointeur interne sur le dernier
  ↪  ajout
  $newResponse =
  ↪  $response->withJson(array('num'=>key($_SESSION['books']),
  ↪  'titre'=>current($_SESSION['books']))); // tab PHP to json
```


Ajouter un titre III

```
  return $newResponse; // ret json du nouveau titre
});
```

- la méthode `withJson()` permet de transformer un tableau associatif PHP en objet JSON
- ce dernier livre ajouté va être renvoyé au navigateur qui l'affichera au format JSON
- on pourrait aussi renvoyer un code de réussite ou d'échec en cas d'insertion dans une BD ou les doublons de titre seraient interdits !

Supprimer un titre I

Le second formulaire de `jscrud.html` permet d'émettre une requête DELETE qui va supprimer un numéro de livre dans la bibliothèque :

```
<h2>delete book</h2>
<form id="formdel">
  <input type="number" id="numdel"  name="numdel"
  ↪  placeholder="numéro">
  <input type="button" value="DELETE" onclick='del();'>
</form>
```

La fonction JavaScript `del()` est appelée lorsqu'on valide ce formulaire. Grâce à JQuery, elle transmet une requête DELETE sur la route jbooks/ en lui fournissant le `num` du titre

Supprimer un titre II

```
function del(){
  $.ajax({
    type: "DELETE",
    url: 'jbooks/',
    contentType: 'application/json',
    context: this,
    dataType: "json",
    data: JSON.stringify({
      'num': $('#numdel').val()
          }
    ),
    success: function (res) {
      console.log("DELETE : ");
      console.log(res);
    },
    error: function (err) {
      console.log("ERREUR : ");
```


Supprimer un titre III

```
      console.dir(err);
    }
  });
  return 0;
 } // fin de del
```

regardons la route associée :

```
<?php /* delete par json */
$app->delete('/jbooks[/]', function (Request $request, Response
↪  $response, $args)  {
  // Sample log message
  $this->logger->info("Slim-Skeleton '/' route");
  $parsedBody = $request->getParsedBody(); // json to tableau
  ↪  asso PHP
  if(isset($_SESSION['books'])){
    if(array_key_exists($parsedBody['num'], $_SESSION['books'])){
      unset($_SESSION['books'][$parsedBody['num']]); // DELETE
```

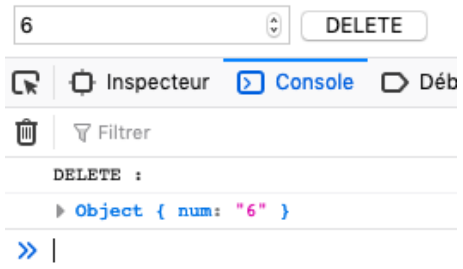

Supprimer un titre IV

```
      $newResponse = $response->withJson($parsedBody); // tab PHP
      ↪  to json
    } else {
      $newResponse = $response->withJson(array("erreur"=>"clé
      ↪  inexistante")); // tab PHP to json
    }
  } else {
    $newResponse = $response->withJson(array("erreur"=>"aucun
    ↪  ouvrage")); // tab PHP to json
  }
  // Render index view
  return $newResponse;
});
```

- la fonction de callback associée à cette route récupère le numéro de livre à supprimer dans le corps du message
  `$parsedBody = $request->getParsedBody();`

Supprimer un titre V

- elle tente de supprimer ce livre de la bibliothèque et retourne un objet JSON contenant une erreur ou le numéro de livre supprimé
- ce résultat sera affiché dans la console par la fonction js `del()`



Mettre à jour un titre I

Le dernier formulaire de `jscrud.html` permet d'émettre une requête PUT qui va modifierun titre de livre dans la bibliothèque :

```
<h2>put book</h2>
<form id="formput" >
  <input type="number" id="numput"  name="numput"
↪ placeholder="numéro">
  <input type="text" id="titreput" name="titreput"
↪ placeholder="titre">
  <input type="button" value="Update (PUT)" onclick='put();'>
</form>
```

La fonction JavaScript `put()` est appelée lorsqu'on valide ce formulaire. Grâce à JQuery, elle transmet une requête PUT sur la route jbooks/ en lui fournissant le `num` du livre et son **nouveau** titre :

Mettre à jour un titre II

```
function put(){
  $.ajax({
    type: "PUT",
    url: 'jbooks/',
    contentType: 'application/json',
    context: this,
    dataType: "json",
    data: JSON.stringify({
      'num': $('#numput').val(),
      'titre': $('#titreput').val()}
    ),
    success: function (res) {
      console.log("MISE A JOUR : ");
      console.log(res);
    },
    error: function (err) {
      console.log("ERREUR : ");
```


Mettre à jour un titre III

```
      console.dir(err);
    }
  });
  return 0;
} // fin de put
```

regardons la route associée :

```
<?php  // put 1 book en json
$app->put('/jbooks[/]', function (Request $request, Response
↪ $response, array $args) {
  // Sample log message
  $this->logger->info("Slim-Skeleton '/' route");
  if(!isset($_SESSION['books'])){
    $_SESSION['books']=array();
  }
  $parsedBody = $request->getParsedBody(); // json to tableau
  ↪ asso PHP
```


Mettre à jour un titre IV

```
  $_SESSION['books'][$parsedBody['num']]=$parsedBody['titre'];

  $newResponse = $response->withJson($parsedBody); // tab PHP to
  ↪ json
  // Return de l'objet mis à jour
  return $newResponse; // ret json
});
```

- la fonction de callback associée à cette route récupère le numéro et le titre de livre à mettre à jour dans le corps du message `$parsedBody = $request->getParsedBody();`
- elle change ou insère ce livre de la bibliothèque et retourne un objet JSON contenant le numéro et le titre de livre mis à jour
- ce résultat sera affiché dans la console par la fonction js `del()`

Mettre à jour un titre V

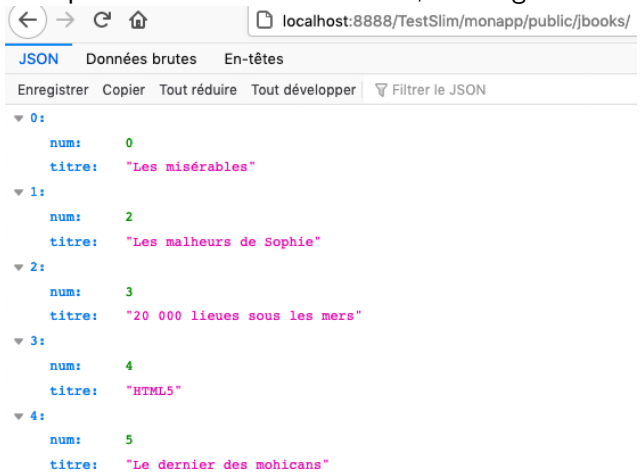
Une dernière route GET ! I

Pour compléter cet exemple, il suffit d'ajouter une route GET afin de récupérer le tableau JSON contenant tous les titres de notre bibliothèque au format JSON :

```php
<?php // get books en json
$app->get('/jbooks[/]', function (Request $request, Response
↪  $response, array $args) {
  // Sample log message
  $this->logger->info("Slim-Skeleton '/' route");
  $tabjson=array();
  foreach($_SESSION['books'] as $k=>$v){
    $tabjson[]=array('num'=>$k, 'titre'=>$v);
  }
  $newResponse = $response->withJson($tabjson);
  return $newResponse;// ret json
});
```


Une dernière route GET ! II

La réponse étant au format JSON, le navigateur l'affichera comme suit :



Plan

Conclusion I

Objectifs de ce cours :

- introduction aux technologies du web non exhaustive : quelques principes
- séparation de la forme (styles CSS) et du contenu HTML
- répartition de la logique de l'application entre script côté client et côté serveur
- double validation des données côté client et serveur
- structuration d'un projet complexe nécessitant de multiples pages : utilisation de frameworks (PHP : Laravel, Symfony, ... JS : Angular 2, React, Vue ...)

Manques :

- architecture logicielle à composants (Angular 2)
- Design Pattern (ORM, MVC, ...)

Conclusion II

- le routage, une notion primordiale
- la sécurité des sites Web (HTTPS, authentification, ...)
- templates PHP (twig, ...)

Plan

Environnement de développement Web I

- éditeur : préférer VSCode ou Sublime ou Eclipse à un éditeur de texte (auto-complétion)
- navigateur : utiliser un navigateur compatible avec tous sysèmes d'exploitation (Firefox ou Chrome)
- outils de développement du navigateur à utiliser intensivement (js)
- débogage PHP possible avec XDebug

Trucs et astuces

- balise `<script ... />` auto-fermante INTERDIT ;
- commentaires dans fichier json INTERDITS ; On peut en mettre `/*` à condition de les supprimer en production grâce à `http://www.httputility.net/` qui MINIFIE le json
- JSON : la moindre erreur syntaxique dans le fichier json interdit son parcours sans expliciter d'erreur dans la console ! Il faut donc vérifier ses json avec `http://jsonlint.com/`
- VIDER le cache nécessaire pour recharger le HTML modifié et les js et les json, css ... menu Historique, supprimer l'historique, détails
- en PHP, ne pas hésiter à exécuter le script en ligne de commande `php index.php` pour voir les erreurs de syntaxique

Bibliographie I

- [1] *Programmation HTML et Javascript*, de Chaléat, Charnay, Eyrolles (2000), "simple et rapide pour les bases HTML, CSS, JavaScript, 450 pages"
- [2] *Webmaster in a nutshell*, de S. Spainhour, R. Eckstein, O'Reilly (2003), "Manuel de référence rapide pour HTML, CSS, JavaScript, PHP, XML, 550 pages, en anglais"
- [3] *Spécification HTML*, `http://www.w3.org/TR/REC-html40/`, W3C (1999), "La spécification complète en anglais"
- [4] *Un site de documentation sur les technologies du web*, `http://www.w3schools.com/`, "Très complet et en anglais (XHTML, CSS, JavaScript) "
- [5] *Le manuel PHP en français*, `https://www.php.net/manual/fr/`, "La référence "

Bibliographie II

- [6] *Site WAMP*, `http://www.wampserver.com/`, "Le site Web de WAMP (Apache, MySQL, PHP pour Windows)"