R209
Initiation au développement Web

Manuel Munier

UPPA STEE - IUT des Pays de l'Adour - Département RT
manuel.munier@univ-pau.fr
https://munier.perso.univ-pau.fr/teaching/butrt-r209/

2023-2024

Plan du cours

Plan du cours

Ce que dit le PPN

- Contenus du module
  - Introduction au protocole HTTP
  - Mise en forme de pages Web
    - → balises HTML avancées
    - → structure d'une page avec son DOM
    - → CSS avancé ou Framework
    - → initiation au dynamisme côté client (JavaScript, bibliothèques comme jQuery)
  - Scripts côté serveur
  - Éléments d'interaction client-serveur (requête HTTP, URL, formulaire)
  - Interrogation d'un SGBD ou d'une API
  - Sensibilisation à la sécurisation de sites : failles XSS, XSS stockée, injections SQL

Ce que l'on va (essayer) de faire...

- Pré-requis
  - R107 : Fondamentaux de la programmation
  - R109 : Introduction aux technologies Web
  - R207 : Sources de données
  - R208 : Analyse et traitement de données structurées
  - \+ modules réseaux, architectures client-serveur, "culture" réseaux, etc.

⇒ `if (lacunes>0) then do_révisions([R107,R109,R207,R208,...]);`

Ce que l'on va (essayer) de faire...

- Ce que l'on va faire
  - ✓ PHP & SQL ⇝ génération dynamique de pages HTML
  - ✓ formulaires HTML ⇝ saisie d'informations
  - ✓ HTML & CSS ⇝ structure pages web & mise en forme
  - ✓ sessions PHP ⇝ authentification & "lien" entre pages
- **Ce que nous ne ferons pas ! ! !**
  - ✗ algorithmique : boucles, tests, structures de données, fonctions,...
  - ✗ ligne de commande : gestion fichiers & répertoires,...
  - ✗ bases de données & SQL
  - ✗ réseaux, fonctionnement des services, requêtes & co.

Pourquoi faire des sites web **"dynamiques"** ?

- **Sites internet**
  - "classique" : les pages HTML ne sont pas "stockées en dur" sur le disque dur, mais générées "à la volée" par des programmes (ex : PHP)
  - le contenu HTML généré peut dépendre des données présentes à cet instant dans une base de données (ex : MySQL, SQLite)
- **Applications web**
  - approche "applications lourdes"
    - → à déployer sur chaque poste de travail
    - ⇒ pbs d'archi système (compilation), de versions (màj), de synchro des données, ...
  - approche "applications web"
    - → centralisées sur un serveur
    - ⇒ accessibles depuis un simple serveur web $\rightsquigarrow$ postes "banalisés", smartphones, etc.

Déroulement

- Modalités de mise en œuvre
  - pédagogie par projet
    - travail en binôme
    - projet par étapes $\rightsquigarrow$ suivi de l'avancement (ou pas...)
    - quelques cours pour donner les "grands principes", mais aussi de l'**auto-formation** (aka "travail perso")
- Évaluation
  - évaluation du projet "au fil de l'eau"
    - ⚠ toute absence non justifiée sera sanctionnée sur la note !
    - ⚠ dans un binôme, les notes peuvent être individualisées
  - démonstration finale
  - documentation technique + documentation utilisateur

Déroulement

- Sujet = conception (technique) d'un site web dynamique
  - technique **côté serveur**
    - **L**inux ⇝ système d'exploitation
    - **A**pache ⇝ serveur web (avec support PHP)
    - **M**ySQL ⇝ serveur de base de données
    - **P**HP ⇝ langage de script (dans les pages web)
  - technique **côté client**
    - langage HTML + CSS ⇝ formulaires, couleurs, tableaux, images,...
    - éventuellement du JavaScript (script exécuté dans le navigateur web)
    - charte graphique... un peu, mais pas trop !

Principes

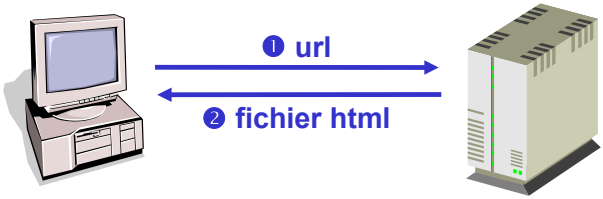
- URL ≡ Uniform Resource Locator (Wikipédia)
  - à la base, une URL identifie une ressource (ex : un fichier HTML) sur un serveur

ex `https://munier.perso.univ-pau.fr:443/teaching/butrt-r209/index.html`

- `https` ⇝ protocole (fixe également le port par défaut)
- `munier.perso.univ-pau.fr` ⇝ serveur (nom DNS ou adresse IP)
- `443` ⇝ n° du port (côté serveur)
- `teaching/butrt-r209/` ⇝ chemin d'accès (sur le serveur)
- `index.html` ⇝ ressource demandée

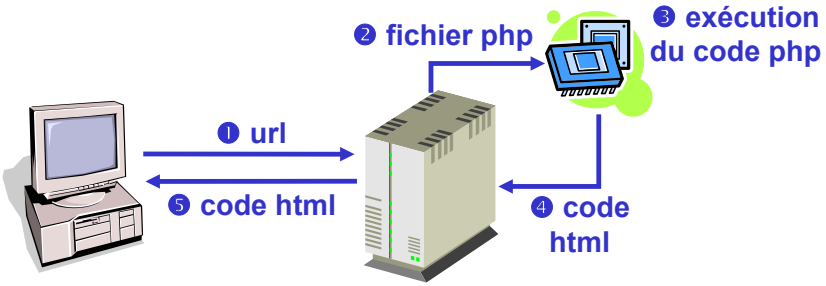
Principes : HTML

- Fichier HTML
  - stocké sur le disque dur (serveur) ⇒ **HTML statique**
  - envoyé en tant que fichier "texte" au navigateur (client)



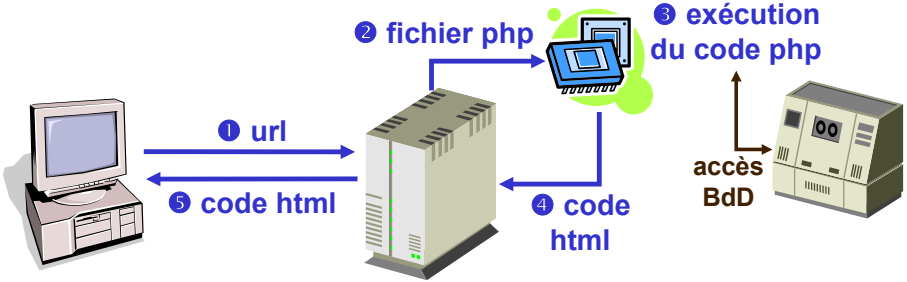
Principes : HTML + PHP

- L'URL indique soit un script PHP soit un fichier HTML contenant des balises PHP
- Fonctionnement
  - le serveur web exécute le script PHP
  - ce script écrit sur sa sortie standard du code HTML ⇒ **HTML dynamique**
  - ce code HTML est alors envoyé au navigateur web



Principes : HTML + PHP + MySQL

- Le script PHP se connecte à une base de données ($\rightsquigarrow$ consultation/màj des données)



Principes : formulaires HTML

- Comment envoyer des données du client vers le serveur ?

⇒ **HTML forms**

- ▷ différents widgets
- ▷ chaque widget fournit une variable ⇝ couple nom/valeur
- ▷ les variables sont encodées dans l'URL

⇒ Récupération des données du côté serveur

- ▷ sous forme de variables dans le script PHP

Principes : JavaScript ?

- JavaScript est également un langage de script (≠ Java !)
- Script encapsulé dans le code HTML pour envoi au client (entre 2 balises)
- Script exécuté côté client par le navigateur web (si JavaScript est autorisé)
- Ex : contrôle des valeurs saisies dans un formulaire, ...

Votre travail

- Vous familiariser avec l'environnement HTML/PHP/SQL
- Concevoir et réaliser votre site web
  - sujet = cahier des charges (définition des besoins)
  - définir la structure de votre site web dynamique
  - concevoir la base de données (et tester les requêtes)
  - écrire les pages HTML (avec formulaire, frames, etc.)
- Documenter votre projet
  - schéma de la BdD (entités/associations)
  - carte du site, enchaînement des pages, déclenchement des scripts
  - explications techniques

Votre travail

- Quelques liens pour débuter
  - https://www.php.net/
  - https://www.mysql.com/fr/
  - https://www.sqlite.org/index.html
  - https://www.w3schools.com/html/html_forms.asp

Plan du cours

Introduction : les principes du web

- Modèle client-serveur
- Le client envoie des requêtes au serveur
  - transfert de fichiers
  - exécution de programmes sur le serveur
  - mise à jour de fichiers
  - etc.
- Les objets manipulés sont repérés par leur URL
- Utilisation du protocole HTTP

Introduction : le protocole HTTP

- Définit le langage utilisé pour les échanges entre client et serveur web (Wikipédia)
  - version 0.9
    - débuts du World Wide Web...
    - simple protocole de transfert de données (GET et réponse)
  - **version 1.1**
    - RFC 2616 (juin 1999)
    - entête Host obligatoire, connexions *keep-alive*, meilleure gestion du cache,...
  - version 2
    - RFC publiée en mai 2015
    - nouveau protocole SPDY proposé par Google ☹
  - version 3
    - travaux en cours
    - remplacerait le procole TCP par QUIC... proposé par Google ☹

Introduction : le protocole HTTP

- Pas de session permanente entre client/serveur $\rightsquigarrow$ protocole sans état
- Déroulement d'une requête HTTP
  - demande de connexion
  - attente de la réponse du serveur
  - établissement de la connexion
  - envoi d'une requête (contenant l'URL)
  - réponse du serveur
  - affichage de la réponse (par le navigateur web)
  - fermeture de la connexion

Introduction : le protocole HTTP

exemple de transaction HTTP

```
munier@atlantis:~$ telnet localhost 80 ..........connexion au serveur web
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /index.html HTTP/1.1 ..........demande de transfert
Host: atlantis.univ-pau.fr ..........nom du serveur (virtual hosting)
From: munier@univ-pau.fr ..........adresse du demandeur (optionnelle)
                                   ligne blanche = fin de l'entête HTTP de la requête
HTTP/1.1 200 OK ..........réponse du serveur
Date: Mon, 21 Mar 2022 14:44:11 GMT
Server: Apache/2.4.41 (Ubuntu)
Last-Modified: Thu, 06 Sep 2018 08:46:35 GMT ..........informations sur la ressource
ETag: "2aa6-5752fedb19641"
Accept-Ranges: bytes
Content-Length: 10918 ..........taille la ressource
Vary: Accept-Encoding
Content-Type: text/html ..........type MIME
                                   ligne blanche = fin de l'entête HTTP de la réponse
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
...
</html>

Connection closed by foreign host. ..........fermeture de la connexion (protocole sans état)
munier@atlantis:~$
```

Introduction : les méthodes HTTP

- Demander une ressource
  - **GET** ⇝ télécharger une ressource
  - **POST** ⇝ envoyer des données et télécharger une ressource
- Obtenir des informations sur une ressource
  - **HEAD** ⇝ connaître ses caractéristiques
  - **OPTIONS** ⇝ connaître les options qui lui sont applicables
- Mettre à jour une ressource distante
  - **PUT** ⇝ créer ou remplacer son contenu
  - **DELETE** ⇝ la supprimer
- Déboguer
  - **TRACE** ⇝ tracer les mandataires (*proxies*)

Introduction : les codes réponse HTTP (➪)

- 10x ⇝ information
- 20x ⇝ succès
  - 200 ≡ OK
- 30x ⇝ redirection
  - 301 ≡ Moved Permanently
- 40x ⇝ erreur du client
  - 401 ≡ Unauthorized
  - 402 ≡ Payment Required
  - 403 ≡ Forbidden
  - 404 ≡ Not Found
- 50x ⇝ erreur du serveur
  - 500 ≡ Internal Server Error

Introduction : web dynamique ?

- Le serveur exécute, le client reçoit
  - CGI, PHP, ASP, JSP
  - indépendance vis-à-vis du client (navigateur)
  - interactivité limitée
  - sécurité (le code source reste sur le serveur, les contrôles sont faits sur le serveur)
- Le serveur envoie, le client exécute
  - JavaScript embarqué, applet Java
  - dépendance vis-à-vis du client (navigateur, OS,...)
  - plus d'interactivité
  - moins de sécurité (le client peut voir et/ou modifier le code source)

Web dynamique

- Quelles technos pour des pages dynamiques côté serveur ?
  - le protocole **CGI** (*Common Gateway Interface*) $\rightsquigarrow$ passage de paramètres client $\rightarrow$ serveur + exécution de scripts CGI
  - les langages de script $\rightsquigarrow$ code exécuté sur le serveur pour générer les pages web "à la volée"
    - Perl (*Practical Extraction and Report Language*)
    - **PHP** (*Php Hypertext Preprocessor*)
    - ASP (*Active Server Pages*)
    - JSP (*Java Server Pages*)

Web dynamique : CGI

- CGI ≡ Common Gateway Interface
- Standard pour l'interface entre applications et serveurs d'informations
- Permet de passer des paramètres aux requêtes
  - ▷ encodés dans l'URL avec la méthode GET
    ex : `http://serv.dom.org/cgi-bin/script?arg1=val1&arg2=val2`
  - ▷ en tant que données avec la méthode POST
- Exécution d'un programme sur le serveur
  - ▷ les informations renvoyées au client sont statiques (code HTML)
  - ▷ des requêtes successives permettent le "dynamisme" $\rightsquigarrow$ fonctionnement par pages

Web dynamique : CGI

- L'architecture CGI est très ouverte... (trop ?)
  - les scripts shell
    - sh, tcsh, bash,...
  - les langages compilés
    - C, C++, Pascal,...
  - Perl
  - Java ($\rightsquigarrow$ JSP, servlets)
  - PHP
  - Python
  - VBScript, JavaScript (ASP)

Web dynamique : Java

- Les servlets
  - programme Java ("lourd") exécuté côté serveur
  - transformation en byte-code avant exécution
  - servies par un serveur dédié (ex : Tomcat, Jetty, JBoss)
- JSP (*Java Server Pages*)
  - pages HTML avec Java embarqué
  - transformation en servlets

Web dynamique : PHP

- Pourquoi PHP ?
  - multi-plateformes
  - **le plus simple**
  - langage non généraliste (comparé à Java par ex.), mais néanmoins très riches ($\rightsquigarrow$ **nombreuses bibliothèques**)
  - produit libre
  - **communauté** très riche et très active

Web dynamique : PHP

- PHP c'est quoi ?
  - un langage
    - qui comprend les CGI nativement (paramètres CGI récupérés directement en tant que variables PHP)
    - complet
    - simple
    - hérité de Perl, C et sh
  - un module pour Apache
    - performant
    - existe aussi en *stand-alone*
  - une techno en pleine évolution
    - au départ pour les pages personnelles ☺
    - aujourd'hui pour les applications sur le web (ex : WordPress)

Web dynamique : PHP

- Un programme PHP, ça ressemble à quoi ? ⇝ **2 approches**
  - ▷ un programme "pur" PHP
    - uniquement du code PHP
    - tout le code HTML est généré par des instructions `echo`
  - ▷ un fichier HTML dans lequel on trouve des tags PHP

exemple de code PHP embarqué dans une page HTML

```
<html>
  <head><title>TEST</title></head>
  <body>
    <p>
      Il est <?php echo date("H:i"); ?>.
    </p>
  </body>
</html>
```

Web dynamique : PHP

- Dans la pratique (surtout pour les gros projets)
  - ▷ un programme PHP est un ensemble d'instructions PHP qui affichent
    - du code HTML
    - ou autre chose : ASCII, PostScript, PDF, GIF, JPEG, PNG, ...
  - ▷ l'approche «pages» est désormais supplantée par l'approche «composants»
    $\rightsquigarrow$ notion de *framework*
- Dans ce module R209
  - ▷ on va comprendre comment fonctionnent ces technos et comment on les assemble
  - ▷ plus tard vous apprendrez à vous organiser dans vos projets

Web dynamique : PHP

exemple de page HTML avec des instructions PHP

```
<html>
  <head>
    <title>
      Bonjour !
    </title>
  </head>
  <body>
    <p>
      Sur le serveur, il est exactement <?php echo date("H:i:s"); ?>
    </p>
  </body>
</html>
```

Web dynamique : PHP

exemple de programme PHP générant du code HTML

```
class maclasse {
  function maclasse($titre)        // constructeur
    { $this->titre = $titre ; }
  function debut()
    { echo "<html><head>$this->titre</head><body>" ; }
  function fin()
    { echo "</body></html>" ; }
}

$s = new maclasse("Bonjour !") ;
$s->debut() ;
echo "<p>Sur le serveur, il est exactement ".date("H:i:s")."</p>" ;
$s->fin() ;
```

Web dynamique : PHP

- C'est le module PHP pour Apache qui :
  1. "intercepte" le code PHP
  2. l'exécute
  3. le "remplace" par le résultat de sa sortie standard
  4. envoie le code HTML ainsi obtenu au client

⇒ Le code PHP **ne sort jamais** du serveur

- NB : Si c'est un fichier HTML avec des balises PHP, le client ne saura même jamais si le code HTML était écrit "en dur" ou s'il a été généré par un script

Formulaires HTML

- Il peut être intéressant que le programme (ex : PHP) adapte son exécution en fonction de données transmises par l'utilisateur (son navigateur web)

⇒ Utilisation de paramètres CGI
ex : `http://serv.dom.org/path/script.php?arg1=val1&arg2=val2`

- Les paramètres CGI seront accessibles directement sous forme de variables du même nom en PHP ; les valeurs sont des **chaînes de caractères**
  - `$arg1` ⇝ `"val1"`
  - `$arg2` ⇝ `"val2"`

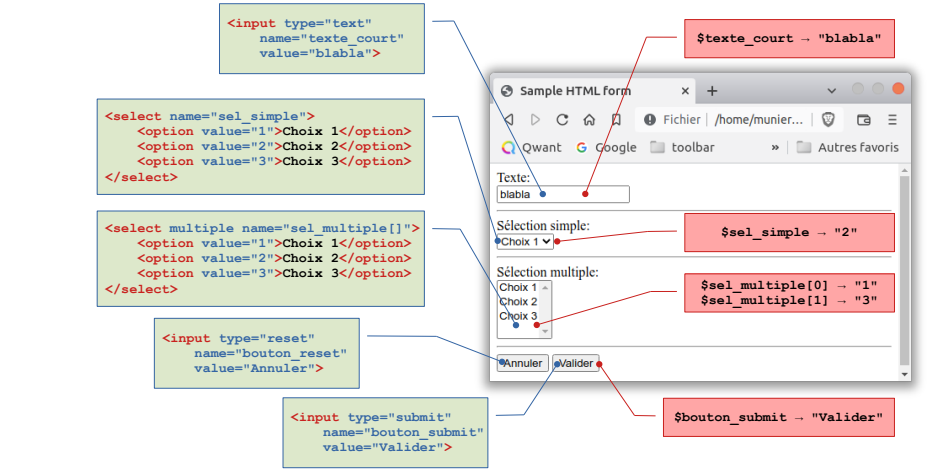
- NB : L'**encodage** et le **décodage** des caractères spéciaux dans l'URL se font automagiquement

Formulaires HTML

exemple de formulaire HTML

```
<form action="test_cgi.php" method="GET">
  Texte:<br>
  <input type="text" name="texte_court" value="blabla">
  <hr>
  Sélection simple:<br>
  <select name="sel_simple">
    <option value="1">Choix 1</option>
    <option value="2">Choix 2</option>
    <option value="3">Choix 3</option>
  </select>
  <hr>
  Sélection multiple:<br>
  <select multiple name="sel_multiple[]">
    <option value="1">Choix 1</option>
    <option value="2">Choix 2</option>
    <option value="3">Choix 3</option>
  </select>
  <hr>
  <input type="reset" name="bouton_reset" value="Annuler">
  <input type="submit" name="bouton_submit" value="Valider">
</form>
```

Formulaires HTML



Plan du cours

Introduction

- 1 site internet = 1 collection de scripts PHP
- Pour "partager" des données entre ces scripts $\rightsquigarrow$ base de données
  - certains scripts vont simplement consulter les données (`select`)
  - d'autres vont les modifier (`insert`, `delete`, `update`)
- PHP dispose d'API pour une multitude de bases de données
  - MySQL $\rightsquigarrow$ nécessite un serveur de BdD
  - SQLite $\rightsquigarrow$ 1 simple fichier dans le dossier des pages web $\Rightarrow$ **notre choix**
  - etc.

PHP + SQLite

- L'extension SQLite3 est supportée nativement depuis PHP 5.3.0
  (actuellement PHP 7.4 sous Ubuntu 20.04 LTS)
- Quelques liens web pour démarrer
  - ▷ SQLite - PHP
  - ▷ Utilisation de SQLite - PHP Facile!
- Même démarche que celle vue en R207 avec Python :
  1. création d'un connecteur à la BdD (localisation, login, mdp)
  2. préparation de la requête SQL $\rightsquigarrow$ une chaîne de caractères
  3. exécution de la requête SQL par le connecteur $\rightsquigarrow$ retourne un résultat
  4. boucle pour parcourir ce résultat $\rightsquigarrow$ traitement pour chaque tuple
  5. fermeture de la connexion à la BdD

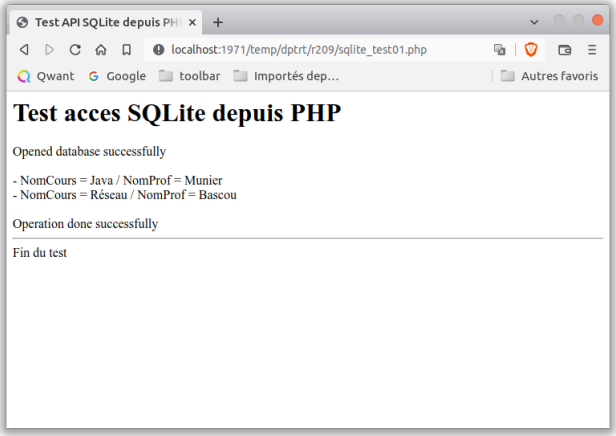
PHP + SQLite

fichier sqlite_test01.php

```
<html>
  <head>
    <title>Test API SQLite depuis PHP</title>
  </head>
  <body>
    <h1>Test acces SQLite depuis PHP</h1>

    <?php
      $db = new SQLite3('bdd_cours.sqlite');
      if(!$db) {
        echo $db->lastErrorMsg();
      } else {
        echo "Opened database successfully<br><br>";
      }

      $sql = "SELECT DISTINCT NomCours,NomProf FROM Cours,Suit WHERE (Suit.NumCours = Cours.NumCours)";
      $results = $db->query($sql);

      while ($row = $results->fetchArray()) {
        echo "- NomCours = {$row['nomcours']} / NomProf = {$row['nomprof']}<br>";
      }

      echo "<br>Operation done successfully<br>";
      $db->close();
    ?>

    <hr>
    Fin du test
  </body>
</html>
```

PHP + SQLite

fichier `sqlite_test01.php`



PHP + SQLite

fichier sqlite_test02.php

```
<html>
  <head>
    <title>Test API SQLite depuis PHP</title>
  </head>
  <body>
    <h1>Test acces SQLite depuis PHP</h1>

    <?php
      class MyDB extends SQLite3 {
        function __construct() {
          $this->open('bdd_cours.sqlite');
        }
      }

      $db = new MyDB();
      if(!$db) {
        echo $db->lastErrorMsg();
      } else {
        echo "Opened database successfully<br><br>";
      }

      $sql = "SELECT DISTINCT NomCours,NomProf FROM Cours,Suit WHERE (Suit.NumCours = Cours.NumCours)";
      $ret = $db->query($sql);

      while($row = $ret->fetchArray(SQLITE3_ASSOC) ) {
        echo "- NomCours = ". $row['nomcours'];
        echo "/ NomProf = ". $row['nomprof'] ."<br>";
      }

      echo "<br>Operation done successfully<br>";
      $db->close();
    ?>

    <hr>
    Fin du test
  </body>
</html>
```

PHP + SQLite

- Bien évidemment, l'objectif n'est pas de générer du **texte brut** comme dans la console, mais plutôt du **code HTML** qui sera affiché **proprement** par le moteur de rendu du navigateur web

- Par exemple en partant du code du fichier `sqlite_test01.php`
  1. balise début de table
     ```
     echo '<table border="1">';
     ```
  2. ligne des titres
     ```
     echo '<tr><th>NomCours</th><th>NomProf</th></tr>';
     ```
  3. 1 ligne pour chaque tuple du résultat
     ```
     echo '<tr><td>'.$row['nomcours'].'</td><td>'.$row['nomprof'].'</td></tr>';
     ```
  4. balise de fin de table
     ```
     echo '</table>';
     ```

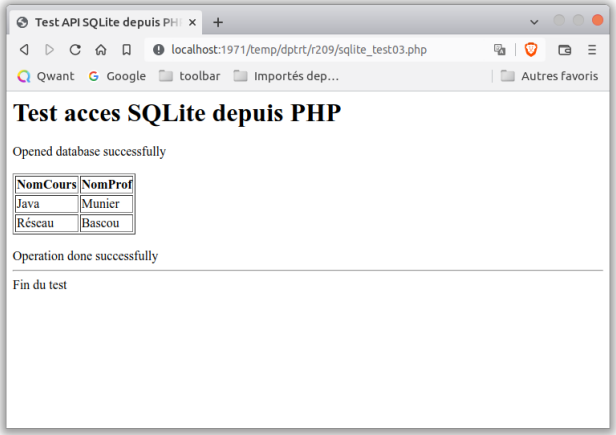
PHP + SQLite

fichier sqlite_test03.php

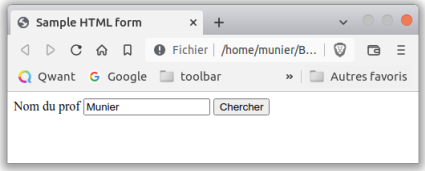
```
<html>
  <head>
    <title>Test API SQLite depuis PHP</title>
  </head>
  <body>
    <h1>Test acces SQLite depuis PHP</h1>

    <?php
      $db = new SQLite3('bdd_cours.sqlite');
      if(!$db) {
        echo $db->lastErrorMsg();
      } else {
        echo "Opened database successfully<br><br>";
      }

      $sql = "SELECT DISTINCT NomCours,NomProf FROM Cours,Suit WHERE (Suit.NumCours = Cours.NumCours)";
      $results = $db->query($sql);

      echo '<table border="1">'.'<tr><th>NomCours</th><th>NomProf</th></tr>';
      while ($row = $results->fetchArray()) {
        echo '<tr><td>'.$row['nomcours'].'</td><td>'.$row['nomprof'].'</td></tr>';
      }
      echo '</table>';

      echo "<br>Operation done successfully<br>";
      $db->close();
    ?>

    <hr>
    Fin du test
  </body>
</html>
```

PHP + SQLite

fichier `sqlite_test03.php`



Injection SQL



formulaire HTML

```
<form [...]>
  Nom du prof
  <input type=text" name="prof">
  <input type="submit" name="go" value="Chercher">
</form>
```

requête SQL (script PHP)

```
$sql = "SELECT * FROM Cours WHERE NomProf='{$prof}'";
```

- Quelqu'un de mal intentionné pourrait vouloir essayer de saisir la valeur suivante :

```
foo';
SELECT * FROM ...;
DROP DATABASE ...;
SELECT '1
```

- Après avoir injecté la valeur ci-dessus dans la chaîne de caractères ci-contre, la requête SQL devient la suivante :

```
SELECT * FROM Cours WHERE NomProf='foo';
SELECT * FROM ...;
DROP DATABASE ...;
SELECT '1'
```

- Qui reste une requête SQL valide... ☹

Injection SQL

⇒ Toutes les données qui proviennent de l'utilisateur doivent être considérées comme suspectes :

- paramètres CGI (depuis un formulaire HTML ou encodés à la main dans l'URL)
- fichiers téléchargés
- ...

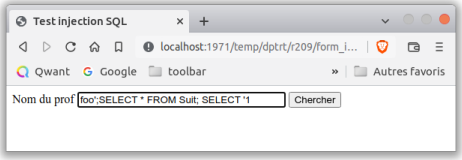
⇒ Il est judicieux de "nettoyer" les données reçues avant de les utiliser

- `public static SQLite3::escapeString(string $string): string`
- *Returns a string that has been properly escaped for safe inclusion in an SQL statement.*

Injection SQL

formulaire HTML

```
<html>
  <head>
    <title>
      Test injection SQL
    </title>
  </head>
  <body>
    <form action="sqlite_injection_sql_test01.php" method="GET">
      Nom du prof
      <input type="text" size ="40" name="prof" value="Munier">
      <input type="submit" name="go" value="Chercher">
    </form>
  </body>
</html>
```



script PHP "unsafe" (extrait)

```
$prof = $_GET['prof'];
$sql = "SELECT * FROM Cours WHERE NomProf='$prof'";
echo "sql = $sql<br><br>";
```

```
sql = SELECT * FROM Cours WHERE NomProf='foo';SELECT * FROM Suit; SELECT '1'
```

script PHP "safe" (extrait)

```
$prof = $_GET['prof'];
$prof = SQLite3::escapeString($prof);
$sql = "SELECT * FROM Cours WHERE NomProf='$prof'";
echo "sql = $sql<br><br>";
```

```
sql = SELECT * FROM Cours WHERE NomProf='foo'';SELECT * FROM Suit; SELECT ''1'
```

Plan du cours

Sessions

- Rappel : HTTP est un protocole sans état ⇒ aucun "lien" entre les requêtes
- Question : Comment "reconnaître" un utilisateur ?
  - nécessité que l'utilisateur se connecte avec un login/mdp
  - étapes successives d'une procédure
  - panier sur un site marchand
- Idée : placer une "info" pour "suivre" l'utilisateur au fil des requêtes

Sessions

- Différentes "technos" disponibles :
  - ajouter un champ "caché" dans les formulaires HTTP
  - utiliser des cookies
  - mettre en place des sessions PHP

- Chaque solution a ses avantages et ses inconvénients...

JavaScript

- Principe : JavaScript a été créé pour qu'un serveur web puisse envoyer un script au navigateur qui l'exécutera côté client

- Pourquoi ?
  - manipuler la structure de la page web $\rightsquigarrow$ le DOM (*Document Object Model*),
  - vérifier les valeurs de certains champs (mdp, adresse mail, etc.),
  - faire quelques animations, etc.

- Mais...
  - $\rightarrow$ problèmes de sécurité $\rightsquigarrow$ confiance dans le code reçu,
  - $\rightarrow$ le client peut refuser d'exécuter le code,
  - $\rightarrow$ le client peut modifier le code avant de l'exécuter, etc.

JavaScript

- Et maintenant...
  - langage de programmation à part entière
  - applications native $\rightsquigarrow$ NodeJS
  - des serveurs web sont écrits directement en NodeJS $\rightsquigarrow$ JavaScript côté serveur
  - mêmes modules / librairies côté serveur et côté client
  - injection de code dans des composants logiciels, etc.

API REST

- REST = REpresentational State Transfer (Wikipédia)
- C'est quoi ?
  - style d'architecture pour créer des services web RESTful
  - un client peut invoquer une fonction/méthode distante (i.e. sur un serveur)
  - tout passe par le protocole HTTP avec des GET, POST, DELETE, UPDATE, etc.
  - les paramètres sont transmis via les CGI encodés dans l'URL (ex : GET) ou dans le body (ex : POST)
  - le résultat est retourné par HTTP également $\rightsquigarrow$ types MIME (ex : texte, JSON, XML, fichier JAR, etc.)

API REST

- Avantages :
  - interopérabilité ⇝ le programme client et le programme serveur peuvent être écrits dans des langages de programmation différents
  - tout passe par le protocole HTTP ⇝ contrôle des ports plus facile sur un firewall
  - 1 appel de méthode = 1 URL
    - tous les langages proposent de faire des requêtes HTTP
    - un simple cURL peut même suffire...
    - il existe des portails web pour tester ces API (ex : Swagger) et ces UI peuvent être générées automagiquement à partir du code du serveur !
  - si besoin de "données structurées" en paramètre ou en résultat ⇝ une String contenant du JSON ☺