Applications Web I – 2024-2025

révision 2.25

Marc Schaefer

évolution et extension du cours de David Grunenwald & André Liechti;

merci à Stefano Carrino pour sa relecture et ses suggestions

28 août 2024

http://www.he-arc.ch/ingenierie

Préface

L'écriture d'applications classiques (non embarquées, sur un seul poste et environnement de travail, dans un seul langage solide, avec des structures de données bien définies) est un processus relativement bien connu depuis assez longtemps.

Cependant, les applications web sont différentes pour plusieurs raisons : les formats de données sont principalement des formats textes pas forcément bien différenciés ni structurés fortement ; les méthodologies existent (MVC), mais sont encore soit embryonnaires, soit leur application est difficile ou diffère via l'existence de (trop) nombreux environnements de développement (toolkit, **framework**) interprétant les méthodologies de manière particulière et nécessitant des apprentissages soit initiaux, soit permanents ; les langages à maîtriser sont multiples (HTML, CSS, Javascript, JSON, XML, SQL, PHP, Perl, Ruby, Python, SVG, ...), les technologies évoluent vite et la compatibilité et la sécurité sont un défi au sein d'un environnement d'exécution interconnecté et non contrôlé.

Le résultat est que bien souvent les applications web, même les plus simples, sont coûteuses à maintenir et à faire évoluer, et peuvent être exploitées par des attaquants (failles de sécurité).

Le but de ce cours est de faire un tour d'horizon des divers langages de base [1] classiques du web afin d'en acquérir les principes, les bases et la pratique, tout en donnant les informations nécessaires afin d'éviter les pièges inhérents à ces technologies, de manière à pouvoir créer une application simple, utilisable et solide, lors d'un projet en groupe au deuxième semestre.

Ce document fait un tour d'horizon des sujets traités au premier semestre. La plupart sont traités lors du cours et certains approfondis lors du travail personnel de l'étudiant (exercices, mini-projet du 1er semestre, projet du 2e semestre).

1. Le cours développement web, de 3e année, lui, se concentrera sur les frameworks **MVC** les plus populaires du langage **PHP** et introduira **Ruby on Rails**. Nous avons dû faire quelques choix : PHP plutôt que Perl ou Python, MySQL plutôt que PostgreSQL.

Sommaire

Sommaire iii

1 Le modèle web 1

2 Développement côté client 31

3 Développement côté serveur 61

4 Interfaçage de bases de données (SGBD) 91

5 Sécurité des applications web 105

Références et bibliographie 115

Lexique 117

Index des concepts 119

Table des figures 125

Table des matières 127

Chapitre 1

Le modèle web

Sommaire

1.1 Le modèle du WWW 1
1.1.1 Internet et WWW 1
1.1.2 Le modèle client-serveur 2
1.1.3 Les documents hypermédia 3
1.1.4 URL : Uniform Resource Locator 3
1.1.5 Passer de l'information 4
1.1.6 Le protocole HTTP 5
1.1.7 Vers le modèle PUSH 13
1.2 Modèles de conception et de déploiement logiciels 15
1.2.1 Le modèle de conception et de déploiement 3-tier 15
1.2.2 Le modèle de conception et de développement (pattern) MVC . . . 16
1.2.3 Autres modèles 16
1.3 Mise en production d'une application web 16
1.3.1 Approche non hébergée : exploitation en interne 16
1.3.2 Approche hébergée 17
1.3.3 L'hébergement 17
1.3.4 Les clouds 19
1.4 Les langages à balises 20
1.4.1 Introduction 20
1.4.2 Standard Generalized Markup Language (SGML) 21
1.4.3 XML 24

Le but de ce chapitre est d'expliquer le modèle de base d'une application Internet basée web, suivant les points de vues des métiers concernés (développeur, intégrateur, administrateur réseau ou système).

1.1 Le modèle du WWW

1.1.1 Internet et WWW

Internet et le World Wide Web sont deux concepts très liés et il est parfois difficile de bien séparer les deux choses pour un néophyte. Si le World Wide Web est le système d'information

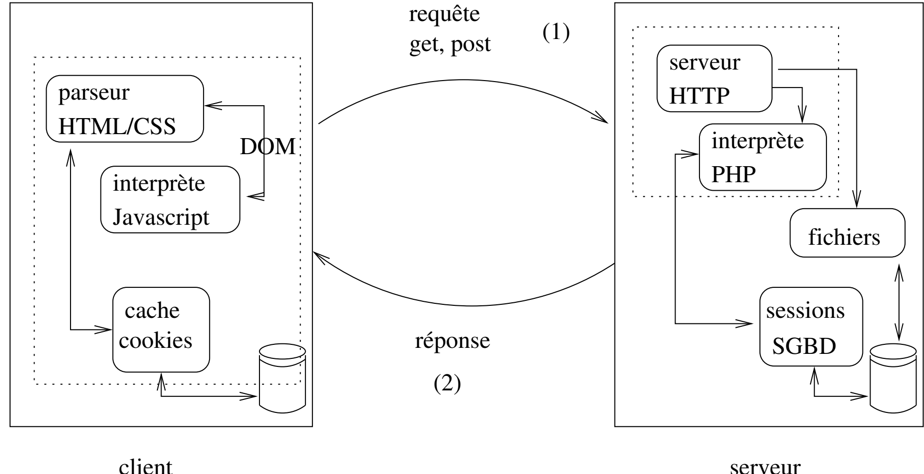


FIGURE 1.1 – Modèle client-serveur du web

distribué si populaire que l'on connaît, il doit son succès à son infrastructure, à savoir le réseau des réseaux qu'est Internet. Un internet est une fédération de réseaux interconnectés. Le réseau Internet est l'internet planétaire à protocole IP (version 4 ou 6). Cette multitude de réseaux interconnectés, ce maillage planétaire, permet la communication entre applications situées sur des terminaux (serveur web, client web), grâce à un protocole commun standardisé que l'on connaît sous le nom de TCP/IP et que l'on peut représenter au sein du modèle OSI (voir section 1.1.6.3 en page 6).

1.1.2 Le modèle client-serveur

Le World Wide Web fonctionne suivant le modèle classique client-serveur (voir la figure 1.1 en page 2). Deux acteurs sont mis en jeu : d'un côté le client qui effectue des requêtes en direction du serveur, de l'autre le serveur qui exécute ces requêtes et renvoie le résultat au client (document HTML, CSS, Javascript, multimédia...).

Le client et le serveur sont en pratique deux logiciels différents communiquant au moyen d'un protocole sur une même machine (p.ex. lors du développement et test local), à travers un réseau local (l'intranet de l'école par exemple) ou bien à travers un réseau étendu (une recherche sur Google).

Côté client, on utilise un client web (butineur, navigateur, *browser*) comme Mozilla Firefox, Google Chrome, Opera ou encore Microsoft Edge, ce qui originellement posait des problèmes de compatibilité. Au centre de ce client web, on trouve de quoi interpréter [1] l'HTML, les CSS et le Javascript. Aujourd'hui, la conformité aux normes du W3C s'améliore constamment, y compris pour du contenu utilisant du Javascript et des API HTML5. Cependant, la plupart des développeurs web utilisent des bibliothèques d'abstraction CSS et Javascript qui leur permettent d'être plus productifs et indépendants du client (notamment en complétant des implémentations manquantes grâce à des **polyfill**).

De nombreux équipements disposent aujourd'hui d'un client web, comme par exemple des télévisions récentes et bien sûr les innombrables mobiles. Ces clients embarqués diffèrent principalement par les extensions (**plugin**s) supportés : il y a rarement des problèmes avec l'HTML et le Javascript classiques, sinon pour la taille et la lourdeur d'interface que l'on peut résoudre en

1. une implémentation est **WebKit**, une bibliothèque logicielle libre, développée par APPLE sur la base d'un fork du moteur de rendu **KHTML** du projet KDE : elle est utilisée aujourd'hui notamment dans Safari et de nombreux navigateurs, notamment hors desktop

modifiant uniquement la partie présentation (visualisation) de l'application (voir section 1.2.2), voire en écrivant des applications natives locales utilisant les **Web services** pour l'interaction avec le serveur.

Le rôle d'un client web se bornait originellement à :

- traduire les ordres que lui donne l'utilisateur à travers l'interface graphique en messages conformes à un protocole d'échange avec un serveur (HTTP, URL, etc.)
- contacter le serveur adéquat et lui passer la requête
- attendre la réponse du serveur
- mettre en forme cette réponse et la présenter de façon convenable à l'utilisateur (*parsing* HTML/CSS, interprétation Javascript, etc)

Aujourd'hui, une partie du calcul et des fonctionnalités peut être déportée côté client, notamment avec les applications web riches faisant usage de **Web services** et des API HTML5 (géolocalisation, caméra ...).

Côté serveur, on utilise souvent le logiciel communautaire multiplateforme libre Apache, mais il existe d'autres produits comme le serveur Microsoft IIS (*Internet Information Services*) ou le logiciel libre Nginx[2].

Le serveur renvoie les informations demandées (document HTML, fichiers, etc) après avoir éventuellement analysé et interprété[3] le code présent dans les fichiers concernés. Le code ainsi interprété peut par exemple rechercher des informations dans une base de données et retourner une page de résultat au client.

1.1.3 Les documents hypermédia

On appelle document hypermédia ou document hypertexte un document auquel on a ajouté un mécanisme permettant d'établir des liens entre différentes parties. On préfèrera le terme hypermédia au terme hypertexte afin d'indiquer que le document ne contient pas que du texte mais également d'autres types de données comme du son, des images ou encore des films.

1.1.4 URL : Uniform Resource Locator

Afin de pouvoir établir facilement un lien vers d'autres documents, il a été nécessaire de définir une nomenclature standardisée pour identifier individuellement les différents documents accessibles à travers Internet. Cette nomenclature est connue sous le nom d'**URL** (Uniform Resource Locator), terme que l'on traduit parfois en français comme étant un *Localisateur Uniforme de Ressource* ou plus simplement une *adresse web*. Cette nomenclature est un sous-ensemble des **URI** (Uniform Resource Identifier).

Un URL se présente sous la forme suivante :

`protocole://[user[:password]@]serveur[:port]/[chemin/]fichier[#position]`

avec :

protocole le nom du protocole. Le plus souvent http, https ou ftp.

2. prononcer Engine-X
3. si l'interprète du langage de script côté serveur choisi y est intégré, par exemple par un module PHP ou Perl ; sinon, diverses interfaces standardisées permettent d'exécuter du code dans n'importe quel langage : CGI (*Common Gateway Interface*, Fast CGI, etc)

serveur le nom d'une machine reliée à Internet (par exemple www.he-arc.ch) ou son adresse IP (157.26.64.64). Nom et adresses sont d'ailleurs reliés par le protocole **DNS**[4].

port le numéro du port (TCP en général) sur lequel le serveur écoute. Une valeur par défaut existe (suivant le protocole utilisé). Par exemple 80 pour l'HTTP, 443 pour l'HTTPS, et 21 pour le FTP.

chemin le chemin (suite de répertoires séparés par des /) vers le document recherché.

fichier le nom du document recherché.

position un nom désignant une position (ancre, *anchor*) à l'intérieur du document. Facultatif.

user et password le client convertira ces informations en protocole d'authentification **auth/basic**, ou **auth/digest** (voir section 1.1.6.13 en page 12), géré par le serveur web et non pas l'application web elle-même : il n'y aura pas d'envoi de ces données dans l'URL.

Les documents hypermedia présents sur le World Wide Web sont écrits dans le langage **HTML** (HyperText Markup Language). Il s'agit de simple texte auquel on a ajouté des constructions spéciales, les tags ou étiquettes, qui permettent de définir la structure du document et les liens vers les autres documents au moyen des URLs.

HTML permet d'abréger les URLs à l'intérieur d'un document HTML, pour alléger l'écriture : on n'a pas besoin de spécifier le début de l'URL (protocole, port éventuel, serveur) et on se borne à préciser le chemin et le fichier ainsi que des paramètres éventuels. Cette simplification a deux formes, explicitées ci-dessous en prenant l'hypothèse que les URLs sont référencés depuis un document dont l'URL est `http://www.mon.serveur.ch/pub/doc.html` :

forme absolue on débutera par un / et l'URL fera référence à la racine du serveur web considéré : par exemple, dans le document ci-dessus, un URL de la forme `/priv/secret.html` référencera en fait l'URL complet `http://www.mon.serveur.ch/priv/secret.html`

forme relative dans le même document, un URL de la forme `misc/other.html`[5] correspondra en fait à l'URL complète `http://www.mon.serveur.ch/pub/misc/other.html`

On reconnaîtra ci-dessus les chemins relatifs et absolus du système d'exploitation **UNIX**. Sur d'autres systèmes d'exploitation, ou si le chemin réfère en fait à des méthodes d'une application, le comportement sera émulé.

Recommandation : on préférera en règle générale la forme relative pour pouvoir facilement déplacer l'application dans une arborescence, sauf si l'URL de base est configurable facilement (dans un fichier de configuration ou dynamiquement en fonction de l'environnement d'exécution).

1.1.5 Passer de l'information

Le client ne va pas seulement actionner certains liens et afficher les résultats, il va aussi passer de l'information (commandes, identification, données etc) qui pourra être exploitée côté serveur. Les deux méthodes pour passer de l'information explicitement (appelée ci-après **paramètres**) sont :

4. nom vers adresse IP via les champs **A**, l'opération inverse – non utile ici – via les champs **PTR**.
5. on trouve parfois la forme `./misc/other.html` qui n'apporte rien

1. par l'URL via des liens a : par convention, un URL contenant un ? peut passer des paramètres[6]. Ainsi, `http://www.mon.serveur.ch/map/draw.php?x=15&y=20` invoque le script `draw.php` se trouvant dans le répertoire `/map` à la racine **htdocs** sur le serveur `www.mon.serveur.ch` en lui passant comme paramètres `x=15` et `y=20`. La méthode HTTP correspondante (voir section 1.1.6.8 en page 9) est **get**.
2. par des formulaires (en général via la méthode **post** et alors dans les données envoyées par le client après la requête).

Le client passe aussi de l'information implicitement par de nombreux champs d'entête HTTP de la requête (par exemple : version du logiciel client, langues supportées, **cookies**, etc), comme nous le verrons ci-après.

1.1.6 Le protocole HTTP

1.1.6.1 Introduction

Un protocole est simplement le langage commun parlé par le client et le serveur pour se comprendre et mener à bien les requêtes de l'utilisateur. Le protocole de loin le plus utilisé sur le World Wide Web est le protocole HTTP (HyperText Transfer Protocol). C'est un protocole développé spécialement pour le transfert de documents typés **MIME**, par exemple hypermédia (HTML) ou CSS, image, son, etc.

HTTP est un protocole sans état : si votre application a besoin de maintenir un état, elle le fera côté client via des mécanismes de plus haut niveau comme les paramètres GET ou POST, des cookies ou du **DOM Storage**[7]. Le concept de **session** désignera d'ailleurs une suite dans le temps de connexions HTTP/TCP correspondant à une suite d'actions cohérentes dans l'application côté serveur, éventuellement précédée d'une connexion.

1.1.6.2 Fonctionnement client-serveur : PULL

Comme vu à la section 1.1.2 en page 2, HTTP est un protocole client-serveur. Seul le client HTTP envoie des requêtes au serveur et celui-ci lui répond. On appelle ce principe de fonctionnement **PULL** : autrement dit, c'est le client qui va toujours chercher de l'information. Ce principe de fonctionnement est fortement limitatif : il ne permet pas par exemple au serveur d'informer le client lorsqu'une information est disponible (notifications asynchrones).

Le grand avantage de ce principe de fonctionnement, en plus d'être extrêmement simple, est qu'il passe facilement un **firewall**.

Les applications web modernes (**web 2.0**) ont besoin de fonctionnalités de notifications (**PUSH**) : pas seulement pour les applications de discussion interactive, mais pour toute sortes d'applications de travail collaboratif, surveillance et gestion, aggrégation de contenu, synchronisation de données, etc.

---

6. notons que, même si le ? est une convention encore fort vivante qui est aussi interprétée comme supprimant tous caches, elle est gentiment remplacée par des URLs de la forme `http://www.mon.serveur.ch/map/draw.php/set/15/20`, notamment dans la philosophie **REST**.

7. contrairement aux cookies qui ont une capacité limitée et une structure plate, les applications **REST** ont besoin d'un stockage client plus signicatif : la spécification Web Applications 1.0, voir `https://developer.mozilla.org/fr/docs/DOM/Storage` permet d'implémenter ce nouveau type de stockage de données côté client) ; enfin certains plugins côté client peuvent proposer leurs propres API, citons par exemple le Flash Local Storage

Il est possible d'implémenter de telles applications **PUSH** sur du PULL, avec quelques astuces pour l'optimisation, dont la plus simple est l'interrogation régulière du serveur par le client (voir section 1.1.7 en page 13). La norme **HTML5** propose de nombreuses fonctionnalités dans ce domaine.

1.1.6.3 Modèle OSI à 7 couches

Le modèle à 7 couches OSI est le modèle standard utilisé en réseaux pour décrire des systèmes interconnectés communiquants. Ce modèle a cependant été conçu après de nombreux développements, notamment de TCP/IP. C'est pour cela que la correspondance n'est pas toujours parfaite dans ce contexte. On peut malgré cela et pour aider à la compréhension de systèmes complexes donner une représentation possible du web dans le modèle OSI [8] ci-dessous :

couche	nom	description	par exemple
7	application	tout ce qui n'est pas traité par les couches inférieures	HTTP
6	présentation	encodage, chiffrement	types **MIME**, **base64**, **charset**, compression (gzip) et chiffrement SSL/TLS
5	session	ouverture, fermeture, reprises de contextes	session, cookie...
4	transport	transmission sûre des données à travers un réseau	TCP
3	réseau	adressage et routage dans un réseau	IP
2	liaison	assurer la transmission de données au sein d'une liaison point à point ou d'un bus multipoint	Ethernet
1	physique	transmission physique de l'information	interface, câblage, signaux

FIGURE 1.2 – Modèle OSI à 7 couches

Où peut-on placer le protocole HTTP dans cette hiérarchie ? Son nom est trompeur (Hyper Text *Transport* Protocol) : en effet, si on le place en couche 4, il est lui-même transporté par TCP, et si du chiffrement intervient, lui sera en couche 6, le protocole HTTP étant alors transporté par lui et donc rejeté en couche 7. De plus, la notion de session en web s'implémente dans l'application elle-même (cookies, sessions...), même si les métadonnées correspondantes figurent dans les entêtes HTTP. Enfin, le protocole HTTP/2.0 (voir section 1.1.6.15.2 en page 13) propose une inversion des couches 5 et 6.

En bref, cette représentation à 7 couches est ici inadaptée et il est plus simple d'utiliser le modèle IP, qui fusionne les couches 5, 6 et 7 à la couche application (voir figure 1.3 en page 7). Ce modèle a l'avantage d'être simple et correct. Notre application web proprement dite deviendrait un utilisateur de cette couche 7 fusionnée.

Le plus important est bien sûr qu'HTTP repose sur **TCP**, un protocole fiable (les datagrammes manquants sont réémis automatiquement) à **connexion**. Cela signifie que chaque connexion HTTP correspond à une connexion TCP. Cela veut dire qu'un certain délai est à attendre à chaque nouvelle connexion (ouverture en 3 phase de TCP). Le **keep-alive** permet d'effectuer

8. notons que le modèle OSI est traité en détails lors du cours Réseaux

couche	nom	description	par exemple
7	application	toutes les fonctions du protocole HTTP (y compris la transmission de cookies, d'informations d'encodage, de compression et de chiffrement par les entêtes HTTP)	HTTP, HTTPS
4	transport	transmission sûre des données à travers un réseau	TCP
3	réseau	adressage et routage dans un réseau	IP
2	liaison	assurer la transmission de données au sein d'une liaison point à point ou d'un bus multipoint	Ethernet
1	physique	transmission physique de l'information	interface, câblage, signaux

FIGURE 1.3 – Modèle TCP/IP – Internet en 5 couches

plusieurs requêtes à la suite. Sans usage de SSL/TLS (protocole HTTPS), le flot de données est entièrement en clair (non chiffré).

1.1.6.4 Historique et versions

Le protocole HTTP est actuellement figé dans sa version HTTP/1.1 et est rétro-compatible. La version HTTP/1.1 permet notamment de gagner du temps en n'établissant qu'une seule connexion pour consulter plusieurs documents du même serveur (HTML, images référencées...) via le concept de **keep-alive**. La version HTTP/1.0 a amené le support des serveurs virtuels partageant la même adresse IP, ce qui a permis en pratique l'hébergement bon marché de masse. Le protocole HTTP/2.0 – déjà en production, voir section 1.1.6.15.2 en page 13 – propose plus de performance et de sécurité.

1.1.6.5 Phases

Le protocole HTTP est très simple, une transaction HTTP se décomposant en quatre phases : connexion, requête, réponse et fermeture[9].

L'activité du programme serveur se limite à attendre les requêtes de clients en restant perpétuellement à l'écoute, une fois reçue la requête, le serveur l'interprète et l'exécute : le plus souvent il renvoie en retour le *document* demandé par le client, souvent stocké sur le serveur sous forme d'un fichier, qui contient la plupart du temps du langage HTML (qui peut donc contenir des références à d'autres documents), des CSS, du code Javascript ou une image en format PNG, GIF ou JPEG, mais il peut contenir absolument n'importe quel autre type de données (texte brut, PDF, fichier son, archive, programme exécutable, etc...). Parfois, le client peut envoyer des données qui seront traitées par un programme ou script du côté serveur : c'est ce qui se passe par exemple lorsque, après avoir rempli les champs d'un formulaire de recherche, l'utilisateur appuie sur le bouton `Submit` du formulaire : les données du formulaire sont passées par le serveur à un script qui les utilise pour produire un résultat (par exemple HTML) qu'il renvoie au serveur qui le fait passer au client.

9. **keep-alive** (voir section précédente), **Websockets** et **WebRTC** exceptés

1.1.6.6 Formats et exemples

Voici un exemple de requête HTTP, telle qu'elle a été envoyée par un client. On peut capturer ces informations avec un outil de capture réseau comme **Wireshark**/**Ethereal**, ou avec les outils réseaux du développeur web de Firefox.

```
GET /wiki/CAPTCHA HTTP/1.1
Host: en.wikipedia.org
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.8)
    Gecko/20100723 Ubuntu/8.04 (hardy) Firefox/3.6.8
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset:  ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
```

Notez :

- uniquement la présence d'entêtes HTTP, sans corps (il n'y a ici pas de données en provenance du client) – s'il y en avait elles suivraient l'entête, après une ligne vide.
- la première ligne est la méthode (voir section 1.1.6.8) – aussi appelée le **verbe** HTTP – ici **get**, suivie du chemin du document désiré (pas l'URL entier), puis la version d'HTTP – si cette version n'est pas indiquée, le protocole est simplifié, aucun entête n'est attendu et la réponse du serveur sera également simplifiée.
- le serveur virtuel est indiqué par l'entête `Host:`
- d'autres paramètres sont indiqués au serveur

Et voici la réponse correspondante du serveur :

```
Content-Length: 21513
Age: 14637
Date: Tue, 07 Sep 2010 06:51:41 GMT
Content-Type: text/html; charset=UTF-8
Server: Apache
Cache-Control: private, s-maxage=0, max-age=0, must-revalidate
Content-Language: en
Vary: Accept-Encoding,Cookie
Last-Modified: Tue, 07 Sep 2010 06:50:48 GMT
Content-Encoding: gzip

<!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" lang="en" dir="ltr">
<head>

...
```

Notez :

- la première ligne de l'entête de la réponse est le status de la requête (voir section 1.1.6.7 en page 9).
- les données (corps de la réponse du serveur) suivent ici l'entête après une ligne vide.

— les divers entêtes particuliers (contrôle du cache du client ou d'éventuels proxies, l'encodage, la compression éventuelle, la date de modification, etc).

Ces informations sont transmises en clair sur le port 80/TCP, sauf si un chiffrement est actif (protocole `https`, port 443/TCP). Les données proprement dites (provenant de l'utilisateur : p.ex. paramètres d'un formulaire en méthode `post` (voir section 1.1.6.8 en page 9) se trouveraient après les entêtes client et seraient encodées pour le transfert (voir section 1.1.6.11 en page 11).

1.1.6.7 Status et codes d'erreurs

La première ligne de l'entête de la réponse du serveur est toujours [10] le status (ou code d'erreur) de la requête, sous la forme :

`version-http status texte`

Le code de status est un code à trois chiffres dont le premier chiffre indique le type du résultat. Ces codes sont similaires à ceux utilisés par d'autres protocoles Internet plus anciens, comme p.ex. **SMTP** ou **FTP**.

code	**signification**	**par exemple**
1xx	information (dès HTTP/1.1)	**WebDAV**
2xx	aucun problème	données du document suivent la ligne vide après l'entête
3xx	redirection	nom ambigu (300), redirection vers une autre URL (permanente : 301, 303 : temporaire, 302 : abusé comme 303, 307 : dès HTTP/1.1 : redirection avec la même méthode)
4xx	erreur du côté du client	document inexistant (404), authentification requise (401), interdit (403), mauvaise méthode, etc...
5xx	erreur du côté du serveur	erreur interne (500)...

Le texte, quant à lui, est une version humainement compréhensible du code d'erreur.

1.1.6.8 Méthodes (ou verbes HTTP)

A chaque requête HTTP, la méthode est spécifiée. Elle peut être, par exemple :

get obtenir un document ; des paramètres peuvent être spécifiés ; l'opération doit être **idempotente** – sans effet de bord – et donc les données peuvent être mises en cache [11]

post transmettre des données ou effectuer une opération qui ne doit généralement pas être mise en cache

head obtenir juste les entêtes

delete conçu originellement pour l'effacement d'une donnée, aujourd'hui se fait plutôt par get ou post, sauf en REST ou la sémantique d'effacement est préservée

10. si aucune version d'HTTP n'est spécifiée dans la requête du client, il n'y aura pas d'entête de réponse !
11. sauf en cas de présence de directives d'entêtes **Cache-control** comme `no-cache`, ou d'autres indications comme la présence du caractère ? dans l'URL

put conçu originellement pour le transfert de fichiers, qui se fait aujourd'hui plutôt comme expliqué à la section 3.2.15.5 en page 86 ; en REST permet de remplacer un objet.

patch en REST, permet de modifier des propriétés d'un objet

La programmation web classique (non **RESTful**, voir section 1.1.6.10.2 en page 10) utilise principalement **get** et **post**. La sémantique du **post** est intéressante car elle n'autorise pas la mise en cache et empêche (par une demande de confirmation à l'utilisateur) le réenvoi d'une opération (p.ex. commande).

1.1.6.9 Redirections sémantiques

Il est d'usage d'utiliser des redirections HTTP, par exemple dans le but d'améliorer l'expérience utilisateur : la soumission d'un formulaire en **post** [12] aboutit à une une redirection **get** ; ou pour assurer l'idempotence : un effacement d'un objet est traitée par **get** pour éviter la lourdeur d'un **post** mais aboutit à une redirection pour éviter à l'utilisateur des problèmes de navigation ultérieurs. On implémentera ces redirections grâce à l'entête HTTP `Location:`.

1.1.6.10 Méthodes étendues

1.1.6.10.1 WebDAV Un système de fichiers complet peut être implémenté par web. Il nécessite quelques méthodes supplémentaires (lister un répertoire p.ex.). Le support existe pour de nombreuses plateformes. Les calendriers partagés et **GIT**, par exemple, peuvent utiliser le **WebDAV** comme couche de transport de leurs données. Plus d'informations dans [15] et [16].

1.1.6.10.2 REST A la différence de la programmation web classique, et notamment pour les interfaces riches côté client et pour les **Web services** légers, **REST** [13] propose d'utiliser une sémantique précise pour toutes les méthodes HTTP (**delete**, **put** et **patch** en plus des usuelles **get** et **post**). Ces méthodes agissent sur des ressources nommées et permettent l'échange de données typées (voir figure 1.4 en page 10).

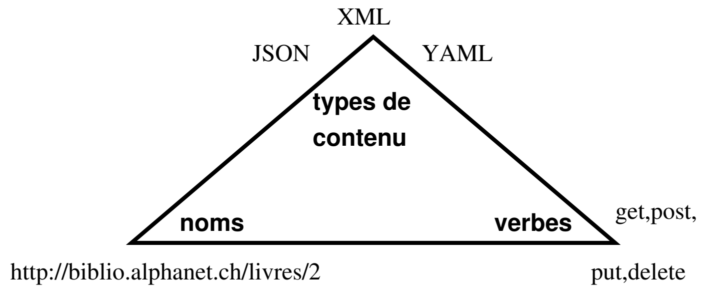


FIGURE 1.4 – Le triangle REST

Chaque méthode ou **verbe**, suivant sa sémantique de base, sert à une opération particulière du modèle **CRUD** [14] : cela évite, notamment, que chaque développeur ait ses propres conventions de nommage de paramètres pour la création, la lecture, la modification, ou la suppression (voir figure 1.5 en page 11).

12. de toute façon meilleure qu'un **get** qui surchargera l'URL et qui ne serait pas sémantiquement correcte car pas idempotente
13. REpresentational State Transfer
14. Create (**post**), Read (**get**), Update (**patch**, ou **put** pour remplacer l'objet), Delete (**delete**)

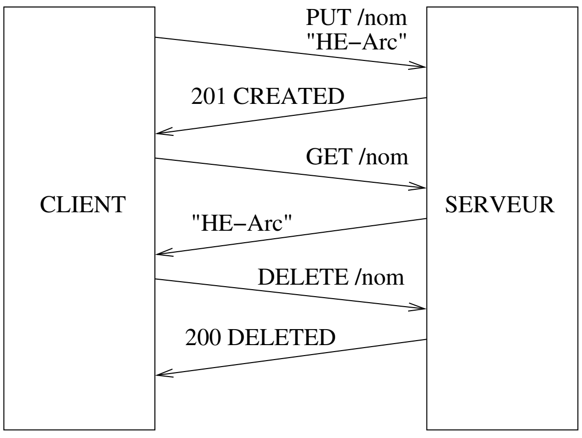


FIGURE 1.5 – Le protocole REST

On peut parler de philosophie d'interaction plutôt que de modèle ou de framework. En pratique, un des avantages de REST est de permettre d'appeler les URLs sans besoin de contexte lourd (on peut facilement sauver des signets ou accéder directement des objets).

1.1.6.11 Types MIME, jeux de caractères et encodages de transfert

Originellement prévus pour les parties jointes du courrier électronique via les *Multi-purpose Internet Mail Extensions*, les types **MIME** sont aujourd'hui incontournables à chaque fois qu'il s'agit de déterminer le type ou l'application associée à une donnée particulière.

Tout document envoyé par HTTP doit être typé MIME (texte, image, document HTML, etc). Les textes (texte pur, HTML, CSS, Javascript) doivent être encodés dans un jeu de caractères spécifique, par exemple **UTF-8**. L'envoi proprement dit peut se faire sans encodage de transfert spécifique (binaire) ou protégé par **base64** [15] ou **quoted-printable** [16]. Le comportement exact se déclare dans les entêtes HTTP.

Les données provenant du client (voir section 1.1.5 en page 4) sont également typées et encodées.

Voici un extrait des entêtes HTTP pertinents dans une requête du client et une réponse du serveur :

```
requête:
   Accept text/html,application/xhtml+xml,
             application/xml;q=0.9,*/*;q=0.8
   Accept-Language en-us,en;q=0.5
   Accept-Encoding gzip,deflate
   Accept-Charset ISO-8859-1,utf-8;q=0.7,*;q=0.7

réponse:
   Content-Type text/html; charset=ISO-8859-1
```

Des séparateurs peuvent être utilisés pour certains types de données, comme p.ex. le type MIME `multipart` : à chaque partie, un nouvel entête `Content-type` et d'encodage est émis.

15. conversion de suite de 8 bits en suites de 6 bits, donc 64 caractères ASCII imprimables, avec report des bits supplémentaires augmentant la taille

16. encodage des octets non ASCII avec une séquence d'échappement `=XY`, avec `XY` le code hexadécimal de l'octet

Vous trouverez dans [18] une liste des types MIME officiels. La problématique des jeux de caractères et des encodages est traitée, quant à elle, dans [25] au chapitre couche 6.

1.1.6.12 Formats de données du web

L'ensemble des types MIME (voir ci-dessus) peuvent être transférés par HTTP, y compris des données binaires non définies à priori. De plus, les applications interactives **AJAX** et les APIs, par exemple REST (voir section 1.1.6.10.2 en page 10), vont échanger des données structurées, originellement en **XML** (voir section 1.4.3 en page 24) mais la plupart du temps aujourd'hui en **JSON**.

Le format ouvert standardisé[17] **JSON** (*JavaScript Object Notation*) permet d'échanger des données structurées qui sont sérialisées (comme des tableaux associatifs Javascript) et qui peuvent être typées, permettant une certaine validation. Il est supporté par de nombreux langages côté client et côté serveur et est en général encodé en UTF-8. Des échappements (\) sont utilisés pour les cas particuliers.

1.1.6.13 Authentification par le serveur web

Une application va en général gérer elle-même[18] l'aspect et le fonctionnement de son authentification via un formulaire de connexion et de la persistence (**cookies** ou **session**). Toutefois, on peut, à la place, ou en plus, effectuer une authentification directement par le serveur web (l'identité de l'utilisateur authentifié peut ensuite être utilisée par l'application via des variables d'environnement). Si ce mécanisme peut être plus compliqué pour l'utilisateur, l'avantage est l'uniformité et l'assurance qu'en cas de bug dans l'application, celui-ci ne peut être exploité que par des utilisateurs authentifiés. Enfin, il fonctionne même si les cookies sont désactivées.

A l'accès d'une ressource qui nécessite authentification, le serveur web retournera une erreur HTTP 401 en indiquant le protocole d'authentification (Basic, Digest...) et l'identification du domaine d'authentification dans un entête serveur. Le client web présentera un dialogue à l'utilisateur lui demandant son login et son mot de passe (un cache d'authentification est utilisé durant la session) et réitérera l'accès à la ressource, en ajoutant un entête client d'authentification.

En Basic, les données d'authentification sont transmises *en clair*, HTTPS (voir section suivante) est nécessaire. En Digest, vu qu'un hachage cryptographique est utilisé, en combinant les informations d'authentification avec un **nonce**[19], l'envoi en clair du hachage n'est pas aussi risqué qu'en Basic, mais HTTPS reste fortement recommandé (notamment en raison du risque d'attaque MitM).

1.1.6.14 HTTPS : Chiffrement SSL/TLS

Il est aujourd'hui quasi obligatoire d'exploiter ses applications en HTTPS et de forcer l'utilisation d'HTTPS (voir section 5.3.8 en page 112). HTTPS devenu nécessaire pour être bien référencé par les moteurs de recherche. De plus, le projet LET'S ENCRYPT automatise la création et la gestion des clés et certificats reconnus.

Le principe d'HTTPS, préfixé `https://`, est d'ajouter une couche de chiffrement (SSL, ou plus récemment TLS, *Transport Layer Security*), reposant sur l'authentification du serveur par

17. ECMA-404 et RFC-8259
18. consulter la section 5.3.9 en page 112 pour d'autres options
19. number-once : nombre garanti unique, transmis par le serveur, évite les attaques de rejeu

certificats **X.509** qui lient une identité numérique (nom de domaine(s)) à une clé de signature ou de chiffrement, dont l'association peut être vérifiée via une **chaîne de confiance** aboutissant à une des autorités (**CA**, *Certificate Autority*) préinstallées dans les navigateurs.

1.1.6.15 Avenir d'HTTP

1.1.6.15.1 Evolution Le protocole HTTP est suffisamment puissant et très largement supporté, notamment par les firewalls et proxies, ce qui fait que pendant longtemps seules des évolutions rétro-compatibles semblaient possibles. Toutefois, deux révolutions, touchant soit le protocole HTTP, soit la couche transport (TCP) sont déjà en test aujourd'hui.

1.1.6.15.2 HTTP/2.0 – Google SPDY Le protocole HTTP/2.0, proposé initialement par Google, permet plus de parallélisme dans les échanges HTTP : l'idée est que le client peut émettre plusieurs requêtes à la fois afin éviter l'attente inutile pendant le traitement d'une requête par le serveur ou le délai de propagation de celle-ci et de la réponse. Ajouté à cette technique d'optimisation connue dans les protocoles[20], en plus, HTTP/2.0 laisse le serveur réordonnancer[21] les requêtes en fonction de la disponibilité des données.

Par exemple, si le client a besoin d'un fichier HTML (généré par un script du côté serveur, prenant un peu de temps), puis d'un CSS et fin d'une image statique, le serveur réordonnancera ces 3 requêtes de manière plus optimale.

Cette nouvelle spécification est déjà productive, mais nécessite des serveurs et clients HTTP/2.0[22] : sinon, un protocole plus ancien est utilisé.

1.1.6.15.3 HTTP/3.0 – Google SPDY over QUIC Et la version ultérieure est déjà là : si HTTP/2.0 était une évolution de SPDY sur TCP (voir paragraphe précédent), **HTTP/3.0** c'est HTTP/2.0 utilisant en couche 4 le protocole **QUIC**[23] – ce dernier protocole de transport propose plusieurs améliorations de performance et fiabilité par rapport à TCP, notamment l'optimisation du débit en fonction des délais mesurés, liés aux tailles de queue des routeurs.

1.1.7 Vers le modèle PUSH

1.1.7.1 Besoins

Les applications **Web 2.0** nécessitent une grande interactivité. Elles interagissent avec l'utilisateur et avec des sites distants, synthétisent (agrègent) du contenu de sources multiples (p.ex. fils **RSS** ou **ATOM**). Tout ceci nécessite l'échange d'informations, et le support d'une méthode permettant de notifier les changements intervenus : prenons par exemple une application de discussion en-ligne (chat). Dès qu'un message est disponible pour l'utilisateur sur le serveur, on aimerait qu'il soit affiché sur le navigateur, sans délai.

Techniquement et pour d'autres protocoles qu'HTTP, une notification est généralement implémentée par la mise en place d'un service d'écoute (listener) sur le client, suivie de l'enregistre-

20. voir les protocoles à fenêtres du cours Protocoles et Réseaux.
21. la réflexion est la même que pour l'introduction du Tagged Command Queuing dans SCSI ou du NCQ dans Serial ATA (SATA).
22. en août 2019, 40% des échanges HTTP se faisaient déjà en HTTP/2.0, selon `https://w3techs.com/technologies/details/ce-http2/all/all`
23. les implémentations actuelles ne sont pas dans les kernels mais sont implémentés dans le client ou le serveur web, par-dessus UDP en couche 4

ment auprès d'un **annuaire**. La notification est initiée par un autre client, qui consulte l'annuaire pour déterminer comment contacter le client (sur quel port se trouve son listener). C'est ainsi que par exemple le protocole **SIP**, utilisé notamment en téléphonie sur IP, procède.

Or, nous avons vu que le protocole HTTP est strictement initié par le client (**PULL**) et jamais par le serveur. Il n'est pas immédiatement possible, dans le cadre du protocole HTTP, d'implémenter des notifications strictes.

1.1.7.2 Implémentations actuelles

Les principes utilisées dans l'implémentation de mécanismes de notification en **Web 2.0** reposent sur plusieurs idées :

- l'envoi de requêtes HTTP asynchrones du client vers le serveur, ces requêtes ayant pour origine non pas une action de l'utilisateur, mais l'expiration d'un délai ou la réponse à une requête précédente de la part du serveur
- le traitement asynchrone des réponses du serveur, en général par du Javascript qui agira ensuite sur la structure du document telle que représentée par le client web (modèle **DOM**) pour modifier l'affichage en fonction des nouvelles informations
- des techniques du côté serveur visant à limiter la charge de celui-ci et le trafic réseau, comme par exemple des requêtes longues, bloquantes pendant un temps ne risquant pas de créer une erreur sur le client, tant qu'un événement n'est pas survenu (**long query**, **long polling**, ou d'autres astuces comme les transferts de données partiels (**Comet**) ou encore **BOSH**)
- le protocole **Websockets** (RFC-6455) permet la communication bidirectionnelle et donc les notifications et le PUSH, mais nécessite le support de fonctionnalités nouvelles sur le client, le serveur et les éventuels proxies au sein du protocole HTTP
- l'API **HTML5 WebRTC** permet les connexions TCP ou les flux UDP directement entre clients, par exemple pour des applications de voix-sur-IP

La technologie **AJAX**[24] – et notamment l'API classique **XMLHttpRequest** ou la moderne **HTML5 Fetch API**, ou en plus haut niveau des **frameworks** comme **JQuery** – permettent d'implémenter ces mécanismes de manière interopérable entre implémentations de clients (et de langages **Javascript**) et de serveurs (simples requêtes HTTP, **XML**, **SOAP**, etc).

1.1.7.3 Evolutions

HTML5 et ses APIs vont dans la direction d'une implémentation plus interactive du web. En effet, HTML5 a du support pour les protocoles peer-to-peer (**P2P**, client à client, en direct, via l'API **HTML5 WebRTC**), et les **Websockets** permettent l'échange facilité de données entre client et serveur, notamment via le support de notifications et de serveur **PUSH** (HTML5 **Server-Sent Events**).

Ces évolutions mettront un certain temps à se standardiser puis à se mettre en place. De plus, la présence de **firewall** ou de **proxy** peut rendre difficile l'exploitation globale de ces nouvelles fonctionnalités, en particulier pour le peer-to-peer. Il existe toutefois déjà des implémentations complètes pouvant utiliser chacune des méthodes en fonction de l'environnement (voir section 2.3 en page 58). L'utilisation de frameworks cachant cette complexité est recommandée.

24. Asynchronous Javascript and XML – même si aujourd'hui on transfère plutôt du **JSON**, voir section 1.1.6.12 en page 12

1.2 Modèles de conception et de déploiement logiciels

1.2.1 Le modèle de conception et de déploiement 3-tier

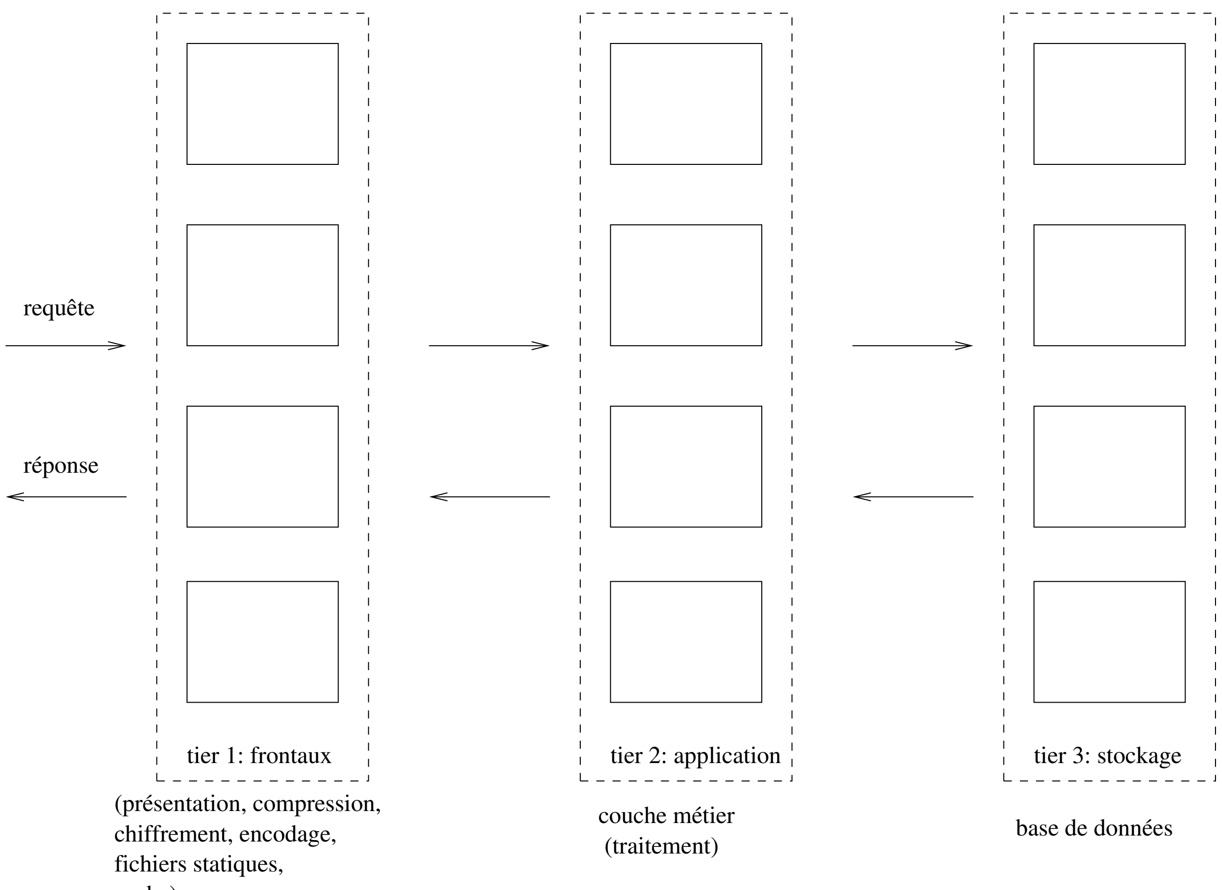


FIGURE 1.6 – Modèle 3-tier classique pour la répartition de charge

Une application web peut rapidement devenir très populaire. La montée en charge [25] de celle-ci est souvent un paramètre critique de l'application.

L'approche relativement classique d'ajouter plus de serveurs pour traiter les requêtes – *throw more hardware at the problem* – peut être affinée, du point de vue de l'administrateur système et réseau, par un découpage logique de l'application en plusieurs partenaires collaborants, appelés aussi chacun un **tier** [26]. Il faut bien sûr concevoir l'application, au départ, avec ce modèle ! Nous verrons que le modèle de conception **MVC** (voir section 1.2.2 en page 16) peut, dans une certaine mesure, permettre au développeur de s'adapter au modèle 3-tier.

Le nombre de serveurs dans chaque tier est à moduler en fonction des besoins.

Sur la figure 1.6 en page 15, le répartiteur de charge (**load-balancer**) n'est pas représenté. Il peut s'agir soit d'une astuce utilisant le DNS et qui peut suffire dans une certaine limite – l'utilisation de plusieurs enregistrements DNS de type **A** (adresse) qui seront alors traités en tourniquet (**round-robin**) – ou d'un logiciel spécifique installé sur un serveur frontal (load-balancing proxy), ou d'un composant réseau comme un **firewall** à suivi de connexion (**connection tracking**) comme le sous-système Linux **VLS**. Le rôle du **load-balancer** est de répartir la charge sur les frontaux, en fonction de certains critères.

25. en anglais : **scalability**.
26. un tier est un partenaire : ici, en présence d'un **load-balancer**, il y a 4 tier

1.2.2 Le modèle de conception et de développement (pattern) MVC

Le but du modèle MVC (Modèle, Vue, Contrôleur) est de séparer la présentation (View), le flot de l'application (Controller), la logique métier et le stockage des données (Model) en trois différentes parties. Les notions de **routeur MVC** (traduisant les routes – les URL – en appels de méthodes du contrôleur) et de redirections sémantiques (voir section 1.1.6.9 en page 10) sont ici essentielles.

Dans le développement web, on distingue entre **light controller** et **fat controller**, suivant si la logique métier est complètement intégrée au modèle ; et on aura, dans la version étudiée au cours, aucune [27] interaction entre la View et le Model qui n'est pas gérée par le contrôleur.

1.2.3 Autres modèles

1.2.3.1 Architecture logicielle de type gestion

1. couche présentation (images, templates, CSS : couche View du MVC)
2. couche coordination (contrôleurs MVC, logique du flux de l'application)
3. couche service (Web services de SOA p.ex., transactionnel)
4. couche métier (logique de l'application)
5. couche persistance (SGBD : cohérence, transactions, etc)

Dans le cas d'un modèle MVC où toute la logique métier est regroupée dans le Modèle (**light controller**), les couches métiers et service du modèle ci-dessus y sont intégrées. La couche persistance peut être vue comme l'interface entre le SGBD utilisé et le Modèle persistant.

1.2.3.2 L'architecture Web service à 5 couches

Cette architecture, exposée dans [13], a pour avantage de décrire tous les partenaires fonctionnels de l'application web (client – serveur – stockage) de manière plus claire.

1. couche client (navigateur et/ou application locale et interaction utilisateur)
2. couche présentation (AJAX, Web 2.0, HTML/XML, Javascript, templates)
3. couche métier (flot de l'application, objets métiers SOAP)
4. couche technique (middleware, transactions, persistance)
5. couche SGBD (SQL, requêtes, stockage)

1.3 Mise en production d'une application web

1.3.1 Approche non hébergée : exploitation en interne

Une approche classique est d'exploiter soi-même le(s) serveur(s) en interne dans l'entreprise et d'offrir les accès sécurisés depuis l'extérieur en fonction des besoins réels. Cette approche est de plus en plus complexe et coûteuse par rapport à une solution hébergée et relève d'une réflexion sur les points suivants :

27. il existe des modèles comme **MVVM** où une interaction bidirectionnelle entre View et Model existe

— le personnel interne ou externe (mandataire) est-il en mesure de gérer le matériel, le système d'exploitation de base, les bases de données, serveur web, les logiciels de déploiement d'application sur la plateforme retenue et le logiciel-lui-même, et à quel coût ?
— l'application n'est-elle utilisée qu'en interne et aucune extension externe n'est prévue dans un futur proche ?
— la plateforme retenue n'est pas disponible facilement en hébergement externe ?
— la confidentialité des données ou des réglementations imposent un serveur local ?
— en cas d'accès extérieur (collaborateurs mobiles, B2B, voire B2C), quelle solution sécurisée d'accès est proposée (**DMZ**, **VPN**, etc)

1.3.2 Approche hébergée

Les applications web sont en règle générale exploitées sur un serveur extérieur à l'entreprise, de manière à simplifier l'accès aux collaborateurs en déplacement (**road warrior**), aux clients (**B2C**) et aux fournisseurs (**B2B**). Dans ce cas, un hébergeur externe, choisi sur le marché ouvert très concurrentiel des hébergeurs sera utilisé et en fonction des besoins.

Dans tous les cas, une évaluation de la solution, *avant* son implémentation (système d'exploitation, base de données, plateforme de déploiement, logiciel, versions et dépendances) est à effectuer, sous la forme de questions fondamentales, liant technique, administration et budget, voire des choix liés au type d'entreprise[28] :

— qui gère le matériel ?
— qui gère le logiciel système, la plateforme et les bases de données ?
— la charge attendue sera-t-elle supportable et les systèmes informatiques sont-ils extensibles ?
— qui accède au système ?
— quel est le niveau de sécurité attendu ? (haute fiabilité, redondance des accès Internet, sauvegarde hors-site des données, etc)
— où sont les données ?

En particulier pour de petites structures ne disposant pas des connaissances d'administration (maintenance, sauvegardes, mises à jour et sécurité), l'exploitation hébergée d'applications Internet est en général une bonne solution.

1.3.3 L'hébergement

1.3.3.1 L'hébergeur

Un hébergeur est une entreprise disposant de locaux (**data center**) et d'une connexion Internet, qui fournit un service d'hébergement (de divers types et respectant diverses conditions) à ses clients.

locaux les locaux peuvent-être sécurisés (accès par clé ou carte, présence de personnel, accès autonome 24h/24 ou non), situés à proximité ou non de l'entreprise, disposant de systèmes de sécurité (alarmes, surveillance, alimentation permanente (UPS), groupes électrogènes, etc.)

28. L'hébergement de données sensibles (médicales p.ex.) peut être difficile, vu que l'hébergeur doit garantir les mêmes conditions de sécurité qui p.ex. règneraient dans un cabinet médical.

connexion Internet la connexion Internet peut être résistante aux pannes (p.ex. via **multi-homing**), rapide/à délai faibles ou non, etc, bien "localisée" par rapport aux utilisateurs (nombre de "hops"), la présence de firewall, de caches, de systèmes de répartition de charge, etc

services de minimaux à très étendus et touchant de nombreux domaines (sauvegardes, support technique, intervention, etc)

1.3.3.2 Types d'hébergement

La terminologie proposée ci-après n'est pas universelle, mais les explications ci-après devraient pouvoir permettre de se faire une représentation des divers types :

hébergement de machine(s) vous achetez et exploitez vos propres serveurs comme en interne, ils sont simplement localisés chez l'hébergeur : il s'agit en règle générale du contrat le plus cher, mais aussi du plus flexible pour vous

machine réelle (dédiée) l'hébergeur gère le matériel, vous exploitez le système : installations, mises à jour, sauvegardes, etc : *hardware as a service* (**HaaS**).

machine virtuelle similaire à la machine réelle, quoique probablement un peu moins performant ; peut permettre de moduler plus facilement la performance du système et réagir rapidement aux pannes : *infrastructure as a service* (**IaaS**).

domaine virtuel de masse vous n'avez qu'à gérer votre application, sa base de données et éventuellement la plateforme. Les autres logiciels (système, serveurs web, interprète PHP, etc), sont gérés par l'hébergeur. La performance, la flexibilité, et la sécurité seront moins bonnes, le prix par contre sera très bas. Dans ce cas, en règle générale, une seule adresse IP désigne plusieurs clients, et la différence est faite par le nom de domaine (hébergement HTTP/1.1), voire par des numéros de ports différents dans le cas de services particuliers. Cela signifie aussi qu'un seul site virtuel envoyant du **SPAM** va empêcher tous les autres de délivrer directement les messages (black-list). Egalement, par conséquence, une préconfiguration des logiciels peut ne pas correspondre aux besoins (p.ex. pour le PHP : `register_globals`, `magic_quotes`, ou la version de l'interprète) ; on peut parler de *platform as a service* (**PaaS**) en particulier si le langage et son environnement sont imposés.

application hébergée si l'application est entièrement gérée par l'opérateur – on parle souvent de *software as a service* (**SaaS**) – il s'agit d'un service pur non extensible (sinon par un choix de modules et/ou une configuration, voire l'adaptation de **templates**)

1.3.3.3 Contrats de service

Un contrat de service (en anglais : service level agreement ou **SLA**) est un accord entre un hébergeur (fournisseur de prestation) et un client. Il précise les conditions (redondance, performance, volume de données, qualité de service (**QoS**), etc.) de l'utilisation et les garanties fournies par l'hébergeur.

Divers contrats de service existent :

— choix de débit (improprement : *bande passante*) ou de volume mensuel, voire qualité de service complète (garanties de délai et de débit)
— gestion des adresses IP (**Autonomous System**, adresses IP de l'hébergeur, adresses virtuelles, multi-homing, etc)
— distribution du stockage (**Amazon S3**)

— caches de données et protections contre le **DDoS** (**CloudFlare**, **Akamai** p.ex.), notamment pour le contenu volumineux (multimédia) – on parle de *content delivery network* (**CDN**)
— et bien sûr sur les services additionnels (firewall, surveillance, adresses e-mail, volume de stockage, redondance, sauvegarde, qualité de service, etc.)

1.3.4 Les clouds

Un concept relativement récent est celui du cloud ou nuage en français. L'idée est d'utiliser pour les applications des ressources offertes par un service d'hébergement distribué public. La facturation est très directement liée aux ressources effectivement utilisées, tout en externalisant les problèmes liés à la gestion globale de ces resources et en particulier les problèmes matériels : *hardware as a service*, *infrastructure as a service* : par exemple, l'Amazon Elastic Compute Cloud (EC2).

Un autre type de cloud est le *platform as a service*, qui propose des logiciels classiques (base de données, environnement d'exécution de langages, etc). Un troisième est le *software as a service* où un logiciel donné est exploité tel quel sans possibilité de modifications (p.ex. ERP).

Un cloud peut également être entièrement interne (privé). Les logiciels de gestion de ces environnements progressent régulièrement, car ils sont basés sur des technologies (**virtualisation**, **system management industrialisation**) qui ont déjà plusieurs dizaines d'années.

1.4 Les langages à balises

1.4.1 Introduction

En informatique, on rencontre souvent le besoin de structurer des informations, de leur donner un **format**. Que cela soit pour des documents de traitement de texte, des formats d'image, des enregistrements de base de données ou les documents hypertextes du web, il est nécessaire de définir un format spécifique, ou d'en réutiliser un existant.

De plus en plus souvent, les formats utilisés sont des formats textes dans lesquels sont ajoutés des **balises** (du **markup** en anglais) qui identifient et qualifient les données.

Certains formats sont dits *binaires*, cela n'empêche pas de spécifier leur format dans un **langage descriptif**, qu'il soit informel (une spécification, un **RFC**) ou formel comme p.ex. la grammaire **EBNF**, dans une structure du langage de programmation C, ou encore en **XML** si ces formats binaires sont encapsulés dans un fichier XML [29].

Quelques exemples de formats :

format	description	langage de description
SVG	format d'image vectoriel	SVG (XML)
ELF	format d'exécutable GNU/Linux	binaire, structure C, documentation EBNF, ...
JPEG	format d'image compressée avec perte	binaire, spécification, structure C
SNMP	format d'interrogation et de réponses d'équipements réseau	binaire structuré, ASN.1 / BER
dump SQL	sauvegarde ou échange de données SGBD	format texte défini dans la norme SQL.99, avec variantes
entête IPv4	format de l'entête d'un datagramme IP version 4	binaire, documentation dans le RFC-791
SOAP	format des messages et méthodes pour appel de procédures distances	WSDL (XML)

Dès les années 1970, de nombreux formats textes comprenant des commandes intégrées ont été développés : citons par exemple LaTeX (ce document est rédigé en LaTeX, voir figure 1.7 en page 21 pour un très court exemple de document), *roff (manuels structurés UNIX), etc. Le concept même d'**échappement** [30] est central à ces formats textes : une séquence de caractère (p.ex. un caractère d'échappement : antislash en LaTeX, point en *roff) est utilisée pour différencier [31] les directives (le contrôle) des données.

Les formats textes à balises sont de plus en plus aujourd'hui décrits dans un **langage à balises** (un cas particulier de **langage descriptif** ou *markup language*). Un langage est une sorte de grammaire qui permet de décrire un document d'un format particulier et définit les règles qui permettent de le *valider* [32]

Ces langages à balises ont conduit ensuite à l'émergence d'un standard universel (un méta-langage) permettant de décrire des langages à balises servant à la description de documents : **SGML** (ISO 8879 :1986).

29. c'est le cas par exemple du format Microsoft OO-XML qui contient des BLOBs binaires plus ou moins spécifiés dans son format XML.

30. séparer contrôle (p.ex. ceci est un titre) des données (le titre proprement dit).

31. ce problème très général est traité à la section 5.3.6 en page 111.

32. vérifier sa bienfacture, voire sa conformité à une grammaire particulière.

```
\documentclass[12pt,a4paper]{article}
\begin{document}
\section{Une section}
\subsection{Une sous-section}
Ceci est un paragraphe.

Ceci est un \emph{autre} paragraph.

\section{Une autre section}
...
\end{document}
```

FIGURE 1.7 – Exemple de fichier au format LaTeX

1.4.2 Standard Generalized Markup Language (SGML)

1.4.2.1 Introduction

SGML, est utilisé notamment pour la définition de langages à balises servant aux mondes de l'édition et de la gestion documentaire (notamment via le format **DocBook** [21]), est composé de trois parties :

1. la structure : décrite dans une **DTD** [33]
2. les données proprement dites : dans des instances d'éléments, validés par la **DTD**, selon un format particulier (assurant l'**échappement** et le support de jeux de caractères, les entités)
3. la façon dont ces données doivent être représentées sur les divers médias de sortie (la présentation) : sous forme de feuilles de style

Un fichier texte balisé validable en SGML (une **instance**) commence toujours par une référence à la **DTD** (qui nomme les éléments utilisables et les relations valides entre ces éléments pour toute instance valide) correspondante.

1.4.2.2 DTD

On définit à la fois les éléments proprement dits (leur contenu, leurs attributs) mais aussi l'organisation logique du document, p.ex. en HTML, l'élément `html` contient forcément un `head` et un `body`, et dans cet ordre là (la parenthèse définit une **séquence**) :

```
<!ELEMENT html (head, body)>
```

Un élément peut aussi ne rien contenir (**EMPTY**), n'importe quoi (**ANY**), ou une expression. Dans ces expressions, des quantificateurs et une syntaxe voisine des **expressions régulières** (voir figure 1.8 en page 22) sont utilisés, comme par exemple :

```
<!ELEMENT TABLE (CAPTION?, (COL*|COLGROUP*), THEAD?, TFOOT?, TBODY+)>
```

Les attributs d'éléments sont définis comme par exemple ci-dessous :

33. Document Type Definition

symbole	rôle
|	alternative entre expressions possibles
,	séquence d'expressions (ordonnées)
()	groupement d'expressions
?	l'expression qui précède est optionnelle
*	répétition : l'expression qui précède est présente de 0 à N fois
+	répétition : l'expression qui précède est présente de 1 à N fois

FIGURE 1.8 – Expressions régulières dans une DTD

```
<!ELEMENT META - O EMPTY               -- generic metainformation -->
<!ATTLIST META
  %i18n;                               -- lang, dir, for use with content --
  http-equiv  NAME           #IMPLIED  -- HTTP response header name  --
  name        NAME           #IMPLIED  -- metainformation name --
  content     CDATA          #REQUIRED -- associated information --
  scheme      CDATA          #IMPLIED  -- select form of content --
  >
```

Une DTD peut être encapsulée dans un document XML de la manière suivante, par exemple pour de l'XHTML :

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" [
<!-- définitions éventuelles supplémentaires -->
]>
<!-- début de l'XHTML -->
<html>
   ...
</html>
```

Dans l'exemple ci-dessus, on fait référence à une DTD externe (système, prédéfinie). Il serait aussi possible de définir une DTD interne additionnelle à l'endroit des commentaires.

1.4.2.3 Exemple d'une application de SGML : l'HTML

Même si **HTML** n'est pas un très bon exemple de langage strict [34] (les instances n'étant pas toujours conformes : manque de fermeture de certaines balises, mélange entre structure et style, etc), c'est une bonne illustration d'une application de SGML. On reconnaît dans la **DTD HTML 4.01 strict** [35] des définitions de types, mais aussi de structure et de relations entre éléments, comme p.ex. la construction de l'**element body** et les notions d'éléments de types **block** et **inline** (voir figure 1.9 en page 23).

D'ailleurs, un document HTML 4.01 strict valide – comme toute classe de documents dérivées de SGML – doit commencer par une référence à une DTD, comme suit (ici sans définitions supplémentaires) :

34. la tentative XHTML ayant échoué
35. http://www.w3.org/TR/html4/sgml/dtd.html

```
<!ENTITY % HTMLlat1 PUBLIC
   "-//W3C//ENTITIES Latin1//EN//HTML"
   "HTMLlat1.ent">
%HTMLlat1;

[ ... ]

<!ENTITY % block
     "P | %heading; | %list; | %preformatted; | DL | DIV | NOSCRIPT |
      BLOCKQUOTE | FORM | HR | TABLE | FIELDSET | ADDRESS">

[ ... ]

<!--=================== Document Body ====================================-->

<!ELEMENT BODY O O (%block;|SCRIPT)+ +(INS|DEL) -- document body -->
<!ATTLIST BODY
  %attrs;                              -- %coreattrs, %i18n, %events --
  onload          %Script;   #IMPLIED  -- the document has been loaded --
  onunload        %Script;   #IMPLIED  -- the document has been removed --
  >
```

FIGURE 1.9 – Extraits de la DTD SGML pour l'HTML4.01 strict

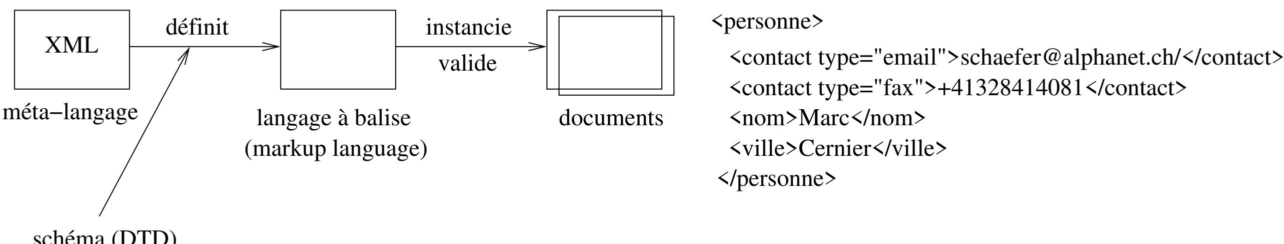
```
 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
            "http://www.w3.org/TR/html4/strict.dtd">
    <html>
    ...
    ...
    </html>
```

C'est parce que l'HTML (basé sur SGML) n'était pas assez strict que la norme XHTML (basée sur XML) a été mise en œuvre. Elle permet alors une chaîne de traitement XML pure, aboutissant à une sortie XHTML pour le client web. Cette façon de faire est encore populaire en grandes entreprises, mais plus du tout sur le web (voir section 2.1.2 en page 32), où le langage HTML5, bien plus simple à écrire car moins strict, mais plus compliqué à *parser*, car les ambiguités doivent être résolues, est désormais utilisé.

1.4.3 XML

1.4.3.1 Introduction

Le but de conception d'XML était de créer un métalangage aussi générique et universel pour la description de documents que le HTML l'est au web. En effet, XML dérive de la norme SGML, mais a le but d'être plus simple à mettre en œuvre tout en étant programmatiquement plus aisé à valider et à traiter (*parsing*). Le monde XML est un univers à lui seul, qui a (re)donné vie à de nombreux concepts [22].



1.4.3.2 Schémas

XML est un métalangage formel qui définit des langages à balises (markup languages) structurés en arbres, ceux-ci permettant de créer des instances de **document**, comme par exemple un document écrit dans le sous-ensemble d'XHTML suivant :

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE html PUBLIC "demo" "demo.dtd">
<!-- <html xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:noNamespaceSchemaLocation="demo.xsd">  -->
<html>
   <head>
      <title>le titre</title>
   </head>
   <body>
      <h1>Premier niveau</h1>
      <p>Ceci est un paragraphe</p>
      <h2>Deuxième niveau</h2>
      <h3 type="a">24</h3>
      <h2>Deuxième niveau bis</h2>
      <h3 type="b">42</h3>
   </body>
</html>
```

Cependant, ce document ne devrait pas exister seul : dans la logique XML, il devrait être validable suivant un certain nombre de règles, par exemple :

- p, h1, h2 et h3 peuvent être présents dans body de 0 à N fois (cardinalité 0..N)
- un élément h3 doit contenir un nombre, et l'attribut type qui peut valoir a ou b
- les éléments html, head, title et body sont obligatoires, et html contient head puis body, et head contient title
- aucun autre élément n'est autorisé

Pour permettre une validation de ce langage à balises particulier, il nous faut une définition d'une **grammaire** et d'un **vocabulaire**. La grammaire précise les règles de **syntaxe** du langage à définir (les relations, la cardinalité (le nombre), la structure – ordre, imbrication, ainsi que les types et format de données littérales autorisées) et le vocabulaire définit les noms d'éléments et d'attributs.

Un document XML peut utiliser des éléments différents de même identificateur provenant de plusieurs vocabulaires : pour les distinguer, XML propose le concept de **namespace**.

Il y a deux manières de définir un langage à balises XML : soit par une **DTD** (comme vu précédemment avec SGML, voir section 1.4.2.2), soit par un schéma XML via un fichier au format **XML XSD**. C'est en général cette dernière forme qui est préférée car un tel schéma, étant aussi une instance de document XML, est *aussi* validable !

En reprenant l'exemple du sous-ensemble spécial du langage XHTML ci-dessus, une manière de décrire ce langage pourrait être la DTD ci-dessous (sans définition d'attributs ni de validation des données) :

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Version simplifiee (sans validation) des contraintes -->
   <!ELEMENT html (head, body)>
   <!ELEMENT head (title)>
   <!ELEMENT body (p*, h1*, h2*, h3*)*>
   <!ELEMENT h1 (#PCDATA)>
   <!ELEMENT h2 (#PCDATA)>
   <!ELEMENT h3 (#PCDATA)>
   <!ELEMENT title (#PCDATA)>
   <!ELEMENT p (#PCDATA)>
   <!ATTLIST h3
      type CDATA #REQUIRED>
   <!ENTITY statement "This is well-formed XML">
```

Enfin, la méthode plus classique de définition, cette fois-ci avec contraintes sur les attributs et les données serait le schéma XML (XSD) `demo.xsd` ci-après :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="html">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="head" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="body" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="head">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="title" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
```

```
    </xs:complexType>
  </xs:element>

  <xs:element name="body">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded" minOccurs="0">
        <xs:element ref="p"/>
        <xs:element ref="h1"/>
        <xs:element ref="h2"/>
        <xs:element ref="h3"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:element name="p" type="xs:string"/>
  <xs:element name="h1" type="xs:string"/>
  <xs:element name="h2" type="xs:string"/>
  <xs:element name="title" type="xs:string"/>

  <xs:simpleType name="h3ValueType">
    <xs:restriction base="xs:string">
      <!-- voir http://www.w3.org/TR/xmlschema-2/#dt-regex  -->
      <xs:pattern value="[0-9]+"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="h3">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="h3ValueType">
          <xs:attribute name="type">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="a"/>
                <xs:enumeration value="b"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Les éléments d'un schéma XML sont par exemple les suivants (au sein du **namespace** `xs`, comme défini dans la déclaration XML du schéma) :

élément	**description**
element	définition d'un élément : l'attribut `name` définit le nom de l'élément et l'attribut `type` optionnel peut spécifier un type scalaire (p.ex. `string`)
complexType	définition d'un type complexe
sequence	type complexe de séquence ordonnée (alternatives : **choice**)

Des cardinalités sont spécifiées dans les attributs de la définition de l'élément comme suit, avec `unbounded` pour ne pas spécifier de limite.

```
<xs:element ref="head" minOccurs="1" maxOccurs="1"/>
```

Une ressource très complète concernant la définition de schémas XML se trouve à [24].

1.4.3.3 Jeu de caractères et entités

Pour pouvoir différencier [36] entre balises (métadonnées) et données littérales, il est nécessaire d'échapper certains caractères. Par exemple, le début de balise, s'il doit être affiché plutôt qu'interprété, doit être écrit comme `<`. De plus, il est d'usage d'écrire les caractères accentués ou spéciaux à l'aide de leurs entités. Cela est utile en particulier vu que beaucoup d'outils de traitement XML ne supportent que l'Unicode (qui est souvent représenté dans le jeu **UTF-8**). Mais attention, contrairement à HTML qui propose un grand nombre d'entités prédéfinies (voir section 2.1.7), XML n'en prédéfinit que 5 (voir figure 1.10 en page 27). Il est possible d'en définir d'autres dans le schéma, ou d'utiliser la forme hexadécimale pour des caractères **Unicode** quelconques.

Caractère	**en XML**	**Unicode**
`"`	`"`	U+0022 (34)
`&`	`&`	U+0026 (38)
`'`	`'`	U+0027 (39)
`<`	`<`	U+003C (60)
`>`	`>`	U+003E (62)

FIGURE 1.10 – Entités prédéfinies en XML

1.4.3.4 Cas d'utilisation

Le cas le plus classique d'utilisation d'XML vise l'échange de données entre applications. Pour ce faire, on spécifie dans un schéma le format des données et les données sont converties du format spécifique interne [37] dans le format d'échange [38] et symétriquement, aux bords du système ainsi défini. Cela est en particulier populaire dans les appels de procédures distantes ou les **Web services**, comme par exemple pour **XMLRPC** ou **SOAP** (langage **WSDL**).

36. à nouveau, voir la section 5.3.6 en page 111.
37. **syntaxe locale**, p.ex. une base de données SQL, une structure de données dépendant du langage en mémoire, etc
38. **syntaxe de transfert**

Cependant, certaines applications stockent en mémoire, voire sur disque, de manière relativement efficace le XML et l'utilisent directement pour les traitements. Cela a du sens en particulier pour des données sous forme d'arbres plutôt que relationnelle[39]. Citons par exemple la base de données XML **eXist**, en Java.

Enfin, les formats d'archivage standardisés ISO sont aujourd'hui en général des formats XML : que cela soit pour les documents de traitement de texte (format XML ODF), ou pour des données plus complexes (indexes de documents, archives officielles, etc).

1.4.3.5 Traitements

Les opérations que l'on peut effectuer sur un document XML sont les suivants :

validation vérifier que le document est conforme au schéma XML du langage spécifique

parsing extraire des informations à la volée (**SaX**) et/ou construire un modèle en mémoire du document (**DOM**).

recherches de données en XML : **XPath**, **XSearch**

stockage de documents de grande taille (livres p.ex.) ou d'informations XML dans des bases de données spécifiques, et langage d'interrogation **XQuery**.

transformation et présentation avec **XSLT**.

1.4.3.6 SaX et DOM

SaX (*Simple API for XML*) est une interface programmatique qui permet de parser un document XML via un modèle événement. Des méthodes sont appelées au moment où les éléments syntaxiques du langage sont rencontrés.

Au contraire, **DOM** (*Document Object Model*), consiste à analyser l'entier du document en une fois en créant un modèle du document *en mémoire*, en général sous forme d'arbre.

Ces deux approches (événements SaX ou arbre en mémoire DOM) ont leurs avantages et inconvénients : DOM étant une API générique idéale pour accéder aux informations depuis un langage de programmation[40], SaX nécessitant plus de programmation spécifique, mais pouvant supporter des documents de taille illimitée.

1.4.3.7 Transformation avec XSLT

XSLT permet, via un processeur XSLT, d'appliquer une feuille de style (sous différentes formes, statiques ou dynamiques) à une instance de document, ce qui permet par exemple de produire une sortie texte, HTML, LaTeX, PDF, CSV, RTF ... ou bien sûr XML, à partir des mêmes données. XSLT est donc un candidat potentiel dans la partie **Vue** de votre application (voir la section 1.2.2). Le livre [23] est une bonne introduction à la transformation de documents et à la présentation en XSLT.

Un traitement particulier est la transformation d'un format XML dans un autre, ou dans tout format nécessaire (HTML, LaTeX, texte, CSV, etc.).

Par exemple, la feuille de style ci-dessous peut être appliquée à l'instance de document dont nous nous occupons depuis le début de la section, de manière à générer un texte simple.

39. il est facile d'émuler des arbres en relationnel-SQL, il est plus difficile de faire l'inverse.

40. citons par exemple le Javascript, qui accède à un modèle objet de document (DOM) représentant p.ex. les données HTML, CSS et d'autres informations, au sein du client web, voir section 2.3.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

   <xsl:output method="text" encoding="iso-8859-1"/>

   <xsl:template match="html">
      <xsl:text>Title: </xsl:text>
      <xsl:value-of select="head/title"/>
      <xsl:apply-templates select="body"/>
   </xsl:template>

   <xsl:template match="body">
      <xsl:apply-templates/>
   </xsl:template>

   <xsl:template match="h1|h2">
      <xsl:text>Head: </xsl:text>
      <xsl:value-of select="."/>
   </xsl:template>

   <xsl:template match="h3">
      <xsl:text>Head 3: </xsl:text>
      <xsl:value-of select="."/>
      <xsl:text> (type </xsl:text>
      <xsl:value-of select="@type"/>
      <xsl:text>)</xsl:text>
   </xsl:template>

   <xsl:template match="p">
      <xsl:text>Content: </xsl:text>
      <xsl:value-of select="."/>
   </xsl:template>
</xsl:stylesheet>
```

1.4.3.8 Outils

Les outils liés aux XML sont nombreux : on peut débuter par un éditeur de texte classique, et valider, parser, chercher et transformer son XML par des méthodes de son langage de programmation préféré : notamment XML est intégré au langage de programmation Java (et est supporté par quasi tous les langages du web, côté client ou serveur).

Il peut être cependant recommandé d'utiliser des outils spécifiquement développés, comme par exemple le logiciel libre **xmlcopyeditor**. Au minimum, une validation (bienfacture et, optionnellement, conformité au schéma) peut être effectuée en ligne par exemple via http://www.xmlvalidation.com/.

Chapitre 2

Développement côté client

Sommaire

2.1 HTML **32**
- 2.1.1 Introduction 32
- 2.1.2 HTML, XHTML, HTML5 32
- 2.1.3 Conformité et compatibilité 33
- 2.1.4 Séparation entre contenu et mise en forme 33
- 2.1.5 HTML en pratique 33
- 2.1.6 Les étiquettes (tags) et les éléments 33
- 2.1.7 Les entités 34
- 2.1.8 Les commentaires 35
- 2.1.9 Création d'un document, déclaration XML et DTD 35
- 2.1.10 Eléments, balises, attributs 36
- 2.1.11 Titre du document et entête 36
- 2.1.12 Corps du document, titres et paragraphes 37
- 2.1.13 Types d'éléments du corps HTML 37
- 2.1.14 Faire des liens 38
- 2.1.15 Insérer des images dans vos documents 39
- 2.1.16 Les listes 40
- 2.1.17 Tableaux 40
- 2.1.18 Sauts de ligne et espace insécable 42
- 2.1.19 Les formulaires 43

2.2 Feuilles de style (CSS) **52**
- 2.2.1 Introduction 52
- 2.2.2 Association de feuilles de style à un document 52
- 2.2.3 Appliquer un style à des éléments : les sélecteurs 54
- 2.2.4 Priorité des spécifications CSS : spécificité, héritage et cascade . . . 56
- 2.2.5 Indépendance du média et du terminal 56

2.3 Javascript **58**

2.4 Autres langages côté client **59**

Le but de ce chapitre est de présenter les grandes lignes des langages côté client, en particulier les langages HTML, CSS et Javascript.

2.1 HTML

2.1.1 Introduction

La grande majorité des documents accessibles sur le World Wide Web et qui en font sa spécificité par rapport aux autres facettes d'Internet, est écrite au moyen du langage HTML (*HyperText Markup Language*). Ce langage a plusieurs incarnations (HTML original jusqu'à HTML 4.01, XHTML 1.0 et 1.1, HTML5) et le but de ce chapitre est aussi de montrer les différences entre ces incarnations, sachant que le monde du web se standardise autour de l'HTML5.

2.1.2 HTML, XHTML, HTML5

Le langage utilisé originellement sur le web était l'HTML, un dérivé de **SGML** (voir section 1.4.2.3 en page 22). Sa syntaxe était assez libre et posait des problèmes d'interprétation par les clients web plus ou moins permissifs. Elle était également incompatible avec une chaîne de traitement XML (1.4.2.3 en page 23).

Pour cette raison, un langage basé **XML** a été ensuite proposé pour décrire la structure des documents : le langage **XHTML**. Toutefois, à part pour les applications spécialisées XML vers XHTML, ce langage ne s'est jamais complètement imposé. Le parser interne du navigateur ne travaillait d'ailleurs en pur mode XHTML que si le bon type **MIME** XHTML était transmis et dans ce cas il ne tolérait aucune erreur :

- chaque balise ouverte doit être refermée
- les balises et leurs attributs doivent être écrits en minuscules
- une balise simple en HTML comme <BR> s'écrit <br /> en XHTML
- les attributs ne sont que de la forme attr="valeur" (valeurs à placer entre guillemets) et ne peuvent pas être abrégés
- les éléments doivent être imbriqués correctement

Exemples :

<p>Bonjour vous</p> Invalide : mauvaise imbrication des éléments. Version correcte : <p>Bonjour vous</p>

 Invalide : les attributs d'une balise sont toujours placés entre guillemets. Version correcte :

<option value="bleu" selected></option> Invalide : les attributs ne peuvent plus exister tels quels. Version correcte :
<option value="bleu" selected="selected"></option>

<h1 name="titre">...</h1> Invalide : l'attribut "name" est remplacé par l'attribut "id" (mais on peut assurer la compatibilité). Version correcte :
<h1 name="titre" id="titre">...</h1>

Un autre problème était que ces langages s'occupaient également de présentation (styles physiques), parfois de manière très manuelle (mise en page en tableaux).

Aujourd'hui, le langage standard du web est l'**HTML5**. Derrière ce nom se cache en fait également le support des CSS pour la présentation ainsi que des interfaces programmatiques Javascript adaptées au monde d'aujourd'hui (applications web et mobiles enrichies).

Sa spécification a été finalisée une première fois fin 2014, pour être régulièrement révisée. Contrairement au XHTML, l'HTML5 ne *nécessite pas* une syntaxe XML totalement stricte, mais

cela peut être utile de la conserver en présence d'une chaîne de traitement XML. HTML5, toutefois, comme ses prédécesseurs, ne tolère pas certaines erreurs d'imbrication ou de délimiteurs d'attributs.

2.1.3 Conformité et compatibilité

Le contrôle de la validité d'un document HTML ou XHTML peut être réalisé en ligne à l'adresse `http://validator.w3.org/`. Il existe par ailleurs des plug-ins pour les clients web. Attention : si vous faites `Sauver la page ...`, il se peut que le code ainsi sauvegardé soit différent du code original. Il est recommandé d'utiliser plutôt un outil comme `curl` ou `wget`, ou de lancer par exemple l'interprète PHP en ligne de commande et de récupérer sa sortie.

Le navigateur détecte la version du langage grâce aux déclarations situées au début du document (voir section 2.1.9 en page 35) ou devine, souvent de manière incorrecte. Cela vaut également pour le jeu de caractère (voir page 35).

Un document dont le type est bien déclaré et valide a plus de chance d'être compatible entre navigateurs : une fois le code HTML et/ou CSS validés, on peut se concentrer sur la vérification de compatibilité de l'aspect sur les implémentations réelles de clients web, par exemple via des outils de tests automatiques sur de nombreuses versions de navigateur, avant d'effectuer des tests plus poussés de fonctionnalité. Il existe des outils comme **Selenium** pour tester automatiquement des logiciels web.

Si une fonctionnalité, par exemple d'API HTML5 ou un attribut spécifique n'est pas encore implémenté dans le navigateur, il est possible d'utiliser un complément Javascript pour implémenter ce qui manque (**polyfill**).

2.1.4 Séparation entre contenu et mise en forme

Aujourd'hui, on décrit la structure d'un document plus que sa présentation, ceci afin d'assurer une indépendance maximale par rapport au dispositif de visualisation. Les caractéristiques de mise en page sont décrites dans une syntaxe particulière (intégrée dans l'HTML ou séparée) : les feuilles de style (**CSS**, ou **style sheets** en anglais, voir section 2.2 en page 52).

2.1.5 HTML en pratique

Très concrètement, un document HTML, lors de son écriture, est un simple fichier texte dont certaines parties portent une signification particulière, ce qu'on appelle le balisage (markup). Ce balisage permet de structurer le document.

Plus formellement, un document est une suite imbriquée d'éléments, respectant la syntaxe (noms des éléments et des attributs de ces éléments) et la grammaire du langage HTML (règles d'imbrications). Chaque élément représente une unité structurelle du document : titre, liste, paragraphe, texte mis en évidence, citation, etc.

2.1.6 Les étiquettes (tags) et les éléments

On appelle étiquettes (tags) les délimiteurs concrets des éléments HTML. Un élément se concrétise la plupart du temps par une étiquette initiale (start-tag) de la forme `<tag>` (pouvant contenir des attributs et leurs valeurs), puis un contenu interne (pouvant contenir d'autres éléments) et enfin une étiquette finale (end-tag) de la forme `</tag>` :

```
<div>contenu interne</div>
```

Certains éléments, que l'on appelle autofermés, comme br (saut de ligne) ont une étiquette à la fois initiale et finale et s'écrivent ainsi en XHTML : <br />. En HTML5, on peut ne pas les fermer et donc les écrire <br>.

Les éléments s'imbriquent les uns à l'intérieur des autres suivant des règles définies par le langage, qui ne peuvent être violées. Il y a notamment le type (bloc ou en-ligne, voir section 2.1.13 en page 37) qui détermine les éléments que l'on peut placer à l'intérieur d'autres éléments.

Les étiquettes peuvent contenir des attributs valués : par exemple, l'élément servant aux liens hypertextes est formé de l'étiquette initiale <a> dans laquelle on trouve l'attribut href qui porte la valeur http://www.he-arc.ch/.

```
<a href="http://www.he-arc.ch/">Serveur de la HE-Arc</a>.
```

Pour chaque élément, seuls certains attributs sont autorisés[1]. La valeur donnée aux éléments doit figurer entre guillemets[2] et ne doit pas contenir de guillemets ou signes > qui doivent, le cas échéant, être remplacés par les entités XHTML équivalentes (voir section 2.1.7 en page 34). En XHTML, les étiquettes et attributs sont *toujours* écrites en minuscules. En HTML5 c'est recommandé.

2.1.7 Les entités

Le fichier HTML suivant est ambigu : on ne sait pas s'il faut afficher ou s'il faut interpréter la chaîne <test> comme l'élément test.

```
<html>
   <head><title>test</title></head>
   <body>
   <h1>Un <test></h1>
   </body>
</html>
```

Pour différencier les caractères devant être affichés des caractères devant être interprétés, en particulier le fameux caractère <, mais pas seulement, une séquence d'échappement (voir section 5.3.3 en page 109) est utilisée pour éviter l'interprétation. Ainsi, le fichier HTML suivant est correct :

```
<html>
   <head><title>test</title></head>
   <body>
   <h1>Un &lt;test&gt;</h1>
   </body>
</html>
```

1. il y a toutefois des attributs utilisables partout, comme id, class ou title et des attributs que l'on peut définir soi-même

2. l'apostrophe est toléré, mais cela a une conséquence sur la sécurité, voir section 5.3.3 en page 110

La séquence d'échappement commence par &, se poursuit par le nom de l'**entité** correspondante (p.ex. `lt` pour less-than, plus petit que ; `amp` pour le et commercial, etc) et se termine obligatoirement par un `;`.

Il ne faut pas confondre encodage des entités comme vu ci-dessus avec l'encodage encore plus strict nécessaire pour le contenu d'URLs (voir section 5.3.3 en page 109).

Notons que XHTML exige, contrairement à HTML, que les & figurant dans des URLs soient encodés comme entités. En HTML5 c'est recommandé pour éviter des ambiguités.

2.1.8 Les commentaires

Un document HTML peut contenir des commentaires qui seront ignorés par l'analyseur-parseur lors de la visualisation :

```
<!-- Ceci est un commentaire HTML -->
```

Un commentaire valide ne doit pas contenir la séquence -- ou le caractère >. Il peut s'étaler sur plusieurs lignes (les retours à la ligne sont autorisés à l'intérieur d'un commentaire).

2.1.9 Création d'un document, déclaration XML et DTD

Un document XHTML étant un document XML, il commence par une déclaration XML indiquant la version et le jeu de caractère utilisé. Ensuite vous devrez spécifier quelle **DTD** vous comptez utiliser avec votre document. Une DTD est un document dans lequel est décrite la manière d'utiliser les différentes balises[3] (une grammaire pour le langage considéré).

Voici donc à quoi ressemble un document XHTML vide :

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
</html>
```

Et en HTML5, c'est un peu plus simple, avec la balise `meta` située dans les entêtes *HTML* – et non HTTP – `head` pour spécifier le jeu de caractères du document :

```
<!DOCTYPE html>
<html>
   <head><meta charset="utf-8"></head>
</html>
```

Le contenu de votre document se trouvera alors entre les balises `<html>` et `</html>`.

On utilise ici le jeu de caractère **UTF-8** qui est assez courant en HTML5. On aurait pu à la place déclarer **ISO-8859-1** (**ISO-latin-1**) qui correspond au jeu ouest européen, ou le jeu **ISO-8859-15**, qui contient le symbole Euro.

3. mais pas seulement : par exemple, l'XML seul ne permet pas d'utiliser les entitités (sauf celles de base comme `&`), c'est la DTD XHTML qui ajoute p.ex. `é`

Il faut simplement que votre chaîne complète de traitement soit configurée de manière cohérente, ce qui inclut : l'éditeur de texte, le frontend et le backend de la base de données, mais aussi [4] la configuration du serveur web spécifiant le charset (qui se retrouvera dans les entêtes HTTP), le charset spécifié sur le document, etc : le plus simple est donc de travailler partout en UTF-8.

2.1.10 Eléments, balises, attributs

Dans ce document nous allons parler d'éléments, de balises (ou tags, étiquettes) et d'attributs. Ces notions ont leur sens propre en HTML, sens que nous allons expliciter :

élément notion abstraite

balise forme concrète d'un élément : on parlera ainsi de balise ouvrante et de balise fermante. Par exemple `<html>` est la balise ouvrante de l'élément html.

attribut propriété d'un élément. Il permet de préciser le rôle ou certaines propriétés d'une balise dans le document. Par exemple l'adresse pointée par un lien :
`<a href="lien.html">lien</lien>`

Les balise et les attributs suivent quelques règles simples :

- les noms des balises doivent être écrits en minuscules
- toute balise ouverte est fermée (par exemple `<html>...</html>` ou `<br />`)
- les noms des attributs doivent être écrits en minuscules
- les valeurs des attributs devraient de préférence être entre guillemets doubles (") pour des raisons de sécurité (voir section 5.3.3 en page 110).

En HTML5, ces règles sont un peu plus permissives : on peut ne pas fermer les balises auto-fermées (`<br>`) et la casse n'est pas importante.

2.1.11 Titre du document et entête

La première chose à faire lorsque l'on crée un document quel qu'il soit est de lui donner un titre. En HTML on donne le titre du document dans une partie appelée entête. L'entête se situe au début du document (après la balise `<html>`) ; elle est délimitée par les balises `<head>` et `</head>`. Elle contient des informations qu'un navigateur n'affichera généralement pas dans son espace d'affichage mais qui pourront être utilisées à des fins diverses. Le titre est défini à l'aide de l'élément `title` comme dans l'exemple suivant :

```
<head> <title>Titre de mon document</title> </head>
```

Pour tester ce que nous venons d'expliquer, créez un fichier `test.html`, écrit en HTML5 dans lequel vous copierez le code suivant :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Titre de mon document</title>
  </head>
</html>
```

Les navigateurs web affichent généralement le titre dans la barre de titre de la fenêtre ou l'utilisent pour diverses fonctions (description de signets p.ex.).

4. il y a des règles de priorité en cas de configuration incohérente, mais ...

2.1.12 Corps du document, titres et paragraphes

Le corps du document est la zone que l'on va trouver après l'entête. Il est défini par l'élément `body`. Les titres sont au nombre de 6, chacun ayant un niveau d'importance. Le plus haut niveau d'importance est le titre `h1`, suivit de `h2` et ainsi de suite jusqu'à `h6`. Ces niveaux sont largement suffisants pour couvrir l'ensemble de vos besoins les plus courants. Un paragraphe est défini par l'élément `p`. Il sera affiché avec un espacement avant et après, permettant ainsi une bonne séparation des paragraphes entre eux et avec le reste des éléments du document. Voici un exemple de titres et paragraphes, vous pouvez le copier dans votre document précédemment créé, juste en-dessous de la balise `</head>` :

```
<body>
   <h1>Titre très important</h1>
   <p>Paragraphe.</p>
   <h2>Titre moins important</h2>
   <p>Second paragraphe un peu plus long que le premier.</p>
</body>
```

Les retours à la ligne sont considérés comme des espaces. Plusieurs espaces ou retours à la ligne consécutifs ne donneront pour résultat qu'une seule et unique espace [5] (faites l'essai).

Il arrive que certains contenus (mots) aient *besoin* d'être mis en évidence. En HTML5, on dispose pour cela des éléments suivants, à utiliser de manière sémantiquement correcte :

- `strong` contenu à mettre en valeur car il est important, sans changer le sens
- `em` emphase sur le contenu, qui change le sens de la phrase
- `mark` texte à surligner ou marquer comme provenant d'un autre contexte

Si ces éléments ont une valeur sémantique (un sens), par exemple pour les moteurs de recherche, ils ont aussi une action concrète par défaut sur la mise en page : par exemple pour `em`, généralement, cela aura pour effet de mettre le texte en question en italique. Sur un navigateur textuel comme Lynx, cela soulignera le mot. Toutefois, le style par défaut de ces éléments peut être modifié comme celui de tous les éléments grâce CSS.

N'utilisez donc pas les styles physiques comme `b` (bold) ou `i` (italique) qui n'aident pas les systèmes sémantiques.

2.1.13 Types d'éléments du corps HTML

2.1.13.1 Introduction

Les éléments HTML qui peuvent figurer à l'intérieur de l'élément `body` (corps, contenu) peuvent être de deux types : **block** ou **inline**.

Même s'il y a aussi une différence syntaxique, affaiblie en HTML5, et une différence sémantique, la différence principale entre ces deux types est liée à la position et la place prise dans la présentation du document. Pour cette raison, en HTML5, on traite les éléments de présentation en CSS Flow Layout, même si des règles sémantiques d'imbrication existent parfois encore.

5. Un espace est le caractère informatique. Une espace est un blanc en typographie.

2.1.13.2 Eléments de type bloc

Les éléments de type bloc peuvent être affichés en rupture du parcours normal du document, par exemple via un **CSS** qui spécifierait la position d'un bloc par rapport à l'autre. De plus, les éléments-blocs occupent par défaut tout l'espace horizontal et commencent toujours par une ligne nouvelle [6].

De plus, ils ne peuvent être imbriqués les uns les autres sans restrictions : la règle générale est qu'un élément de type bloc ne peut pas être inclus dans un élément qui n'attend que des éléments en-ligne (inline). Par exemple, un paragraphe `p` ne peut pas contenir un entête `h1` ou un `div`. Par contre, une cellule d'une table ou une entrée de liste peuvent contenir des paragraphes.

Quelques exemples d'éléments bloc : `div`, `h1` à `h6`, `p`, `table`, `pre`, `ul`, `form`, `fieldset`, ...

La division `div` permet de grouper du contenu sémantiquement proche. Elle ne peut que contenir d'autres éléments blocs. Cet élément est très utilisé pour la mise en page via les CSS.

2.1.13.3 Eléments de type en-ligne (inline)

Les éléments en-ligne ne peuvent qu'agir sur du contenu directement placé à l'intérieur d'eux et n'ont pas de restrictions particulières de placement, sinon qu'ils doivent être à l'intérieur d'un bloc. Il ne prennent par défaut que la place horizontale nécessaire.

Quelques exemples d'éléments inline : `a`, `b`, `br`, `button`, `img`, `input`, `label`, `map`, `strong`, `script`, ...

En HTML5, il n'est pas nécessaire de mettre un élément inline dans un élément bloc et il est permis de disposer des éléments de type bloc dans un élément inline tant qu'il n'y a pas de risque de confusion [7] sémantique ou sur les styles par défaut CSS, par exemple ce code est valide en HTML5 :

```
<a href="url"><h1>Titre</h1></a>
```

En effet, l'élément bloc contenant l'élément `a` est implicite, et il n'y a pas de problème sémantique ni de présentation CSS à mettre un `h1` dans un `a`.

Par contre, mettre un `div` dans un `span` serait inutile, donc interdit.

2.1.14 Faire des liens

L'aspect le plus intéressant du web est cette formidable capacité à créer des liens de pages en pages, de sites en sites. Pour créer un lien on utilise l'élément `a` (de l'anglais *anchor* pour ancre).

Pour créer un lien vers un fichier `test2.html` se trouvant dans le même répertoire que votre fichier `test.html` il suffira de faire comme dans l'exemple suivant :
`<a href="test2.html">Second fichier test</a>`. Le navigateur affichera par défaut le texte en bleu et le soulignera. A la place d'un texte, on peut aussi imbriquer d'autres éléments de type inline, par exemple une image avec l'élément `img`.

---

6. Il est cependant possible d'influencer sur ce comportement par des CSS, notamment via la propriété `display`.
7. https://html.spec.whatwg.org/#restrictions-on-content-models-and-on-attribute-values

Pour faire un lien vers un autre site web on écrira l'URL comme valeur de l'attribut `href`, par exemple avec le code suivant : `<a href="http://www.w3.org/">Le W3C</a>`. Il est bien sûr possible de faire des liens vers autre chose qu'une page HTML. Vous pouvez faire des liens vers des images, des fichiers audio, des documents textes, etc.

Il est souvent utile de faire des liens à l'intérieur d'un même document (p.ex. pour une table des matières) : pour référencer le lien utiliser la forme `<a href="#nom">...</a>`, avec `nom` l'identificateur unique de l'emplacement lié, qui se définit par exemple avec `<a id="nom"></a>` [8].

2.1.15 Insérer des images dans vos documents

Afin de rendre vos documents plus attrayants, vous pouvez y insérer des images à l'aide de l'élément `img`. Admettons que vous possédiez une photo de votre ami Tristan en train de s'amuser avec un Lézard en plastique (simple hypothèse), le fichier de cette image s'appelant `images/tristan.jpg`, voici le code à écrire pour insérer l'image dans votre page (en notant que les attributs `width` et `height` sont facultatifs mais optimisent [9] le rendu initial) :

`<img src="images/tristan.jpg" width="180" height="130">`.

Cet élément est intéressant à plus d'un titre ; il vous montre comment insérer une image dans un document et la syntaxe correcte en HTML5 pour un élément ne possédant pas de balise fermante. L'élément `img` ne contient, en effet aucun texte, il est donc faux d'écrire une balise fermante, et le `/>` n'est pas nécessaire en HTML5.

L'exemple précédent n'est pas correct car il manque une information capitale. Dans le cas précédent, si votre visiteur ne peut pas lire les images car il ne possède qu'un navigateur textuel ou vocal (ou bien car un serveur **proxy** très intelligent filtre les contenus tendancieux de personnes pratiquant les jeux avec des lézards), il est indispensable de lui fournir un texte décrivant votre image. C'est aussi utile pour les moteurs de recherche qui n'effectuent en général pas de reconnaissance d'images. Voici donc la manœuvre avec l'attribut `alt` (pour les formes alternatives de navigateurs ne pouvant représenter l'image) et `title` (pour présenter un texte d'information complémentaire lorsque la souris est placée au-dessus de l'image, **tooltip**).

```
<img src="tristan.jpg" width="180" height="130"
     alt="Tristan jouant avec un lézard en plastique"
     title="Tristan et un lézard">
```

NB : l'attribut title, en HTML5, peut être utilisé sur n'importe quel élément HTML.

Avec cet exemple, vos visiteurs savent de quoi il s'agit, ce qui est bien mieux, surtout s'ils ne peuvent afficher les images. Si jamais vous aviez besoin d'en dire beaucoup plus sur votre image, si par exemple, vous présentez un graphique avec courbe, camembert (ou bien si vous en savez plus long sur l'affaire du lézard) vous pouvez intégrer votre image dans un lien hypertexte, comme par exemple pour le logo de votre organisation permettant de revenir à la page d'accueil :

```
<a href="/"><img src="logo.gif"
                 width="180" height="130"
                 alt="Logo de l'organisation"></a>
```

8. avant XHTML 1.0, on utilisait l'attribut `name` dont la valeur n'était pas forcément unique : `id` a une valeur qui est forcément unique dans un document valide HTML ; il est toutefois possible de préciser ces deux attributs pour compatibilité.

9. dans le but d'une réservation initiale de l'espace que prendra l'image ; si le but est une mise à l'échelle, il suffit de spécifier une seule des dimensions et il y aura respect des proportions et réservation de l'espace automatique après lecture de l'image ; on peut aussi utiliser un pourcentage

2.1.16 Les listes

Les listes sont des outils bien utiles pour présenter des informations. HTML en offre 3 types différents. Le premier type de liste est la liste non ordonnée, autrement appelée liste à puces. En voici un exemple :

```
<ul>
  <li>1er élément</li>
  <li>2ème élément</li>
  <li>3ème élément</li>
</ul>
```

NB : en HTML5, il n'est pas nécessaire de fermer le `li`.

Ce type de liste sera présenté avec des petites puces avant le texte de chaque élément. Le second type de liste est la liste ordonnée :

```
<ol>
  <li>1er élément</li>
  <li>2ème élément</li>
  <li>3ème élément</li>
</ol>
```

Cette liste ne sait pas mieux ranger ou faire le ménage que l'autre, elle doit uniquement son nom au fait qu'elle affiche un numéro au lieu d'une puce avant le texte de chaque élément. Enfin, le dernier type de liste est la liste de définition, permettant de créer des listes possédant un terme et sa définition, jugez plutôt :

```
<dl>
  <dt>Premier terme</dt>
  <dd>Définition du premier terme.</dd>
  <dt>Second terme</dt>
  <dd>Définition du second terme.</dd>
</dl>
```

Ce dernier type de liste est généralement affiché avec la définition en retrait du terme.

2.1.17 Tableaux

Longtemps, les tableaux ont été utilisés pour effectuer la mise en page des éléments d'une page web. Cela est aujourd'hui évité pour des raisons sémantiques. Cependant, les tableaux sont toujours utiles pour présenter des informations de manière plus claire que dans une liste ou des paragraphes.

2.1.17.1 Tableaux simples

Un tableau HTML est découpé en lignes contenant des cellules. Le nombre de cellules dans chaque ligne doit être le même, ou alors il est nécessaire de spécifier des options de recouvrement. Voici un exemple de tableau très simple :

```
<table>
  <tr>
    <th>ligne 1, colonne 1 (entête)</th>
    <th>ligne 1, colonne 2 (entête)</th>
    <th>ligne 1, colonne 3 (entête)</th>
  </tr>
  <tr>
    <td>ligne 2, colonne 1</td>
    <td>ligne 2, colonne 2</td>
    <td>ligne 2, colonne 3</td>
  </tr>
  <tr>
    <td>ligne 3, colonne 1</td>
    <td>ligne 3, colonne 2</td>
    <td>ligne 3, colonne 3</td>
  </tr>
</table>
```

Les balises `<tr>` et `</tr>` délimitent les lignes du tableau. Dans la première ligne, la balise `<th>` désigne des cellules d'un type particulier : elles ne contiennent pas de données, mais l'entête (ou titre) des colonnes correspondantes. Dans les lignes suivantes, `<td>` spécifie les cellules de données.

2.1.17.2 Tableaux avec cellules recouvrantes

Comme nous l'avons dit, des cellules peuvent en recouvrir d'autres, que ce soit en largeur ou en hauteur. Pour qu'une cellule occupe n cellules vers la fin de la ligne, on utilise l'attribut `colspan="n"`. Pour qu'une cellule occupe n cellules vers la fin de la colonne, on utilise l'attribut `rowspan="n"`.

Voici un exemple où les cellules supprimées sont laissées en blanc :

```
<table>
  <tr>
    <td rowspan="2">ligne 1 et 2, colonne 1</td>
    <td>ligne 1, colonne 2</td>
    <td>ligne 1, colonne 3</td>
  </tr>
  <tr>
    <td>ligne 2, colonne 2</td>
    <td>ligne 2, colonne 3</td>
  </tr>
  <tr>
    <td>ligne 3, colonne 1</td>
    <td colspan="2">ligne 3, colonne 2 et 3</td>
  </tr>
</table>
```

2.1.17.3 Titre de tableau

Vous pouvez donner un titre à votre tableau grâce à l'élément `caption`. Cet élément va placer le titre, généralement au dessus du tableau. Voici un exemple d'usage de l'élément caption :

```
<table>
  <caption>Mon premier tableau</caption>
  <tr>
    <td>ligne 1, colonne 1</td>
    <td>ligne 1, colonne 2</td>
  </tr>
  <tr>
    <td>ligne 2, colonne 1</td>
    <td>ligne 2, colonne 2</td>
  </tr>
</table>
```

2.1.18 Sauts de ligne et espace insécable

Il est commode, pour diverses raisons, de sauter une ligne dans un texte, sans pour autant marquer un saut de paragraphe (en poésie par exemple). Ceci est permis par l'élément `br`. Cet élément est autofermé.

Voici un exemple :

```
<p>
  vidi lecta diu et multo spectata labore<br>
  degenerare tamen, ni vis humana quotannis<br>
  maxima quaeque manu legeret. sic omnia fatis<br>
  in peius ruere ac retro sublapsa referri,<br>
  non aliter quam qui adverso vix flumine lembum<br>
  remigiis subigit, si bracchia forte remisit,<br>
  atque illum in praeceps prono rapit alveus amni.
</p>
<p>Texte de Virgile.</p>
```

La langue française impose d'utiliser des espaces avant certaines marques de ponctuation telles que le point d'interrogation, le point d'exclamation, le point virgule ou encore les deux points. Ceci est facile à faire, il suffit de mettre un espace. Le problème est que votre marque de ponctuation, avec un espace normal, peut passer à la ligne, ce qui sera des plus disgracieux. Heureusement, il existe un caractère particulier dit espace insécable. Un mot suivi d'un espace insécable et d'une marque de ponctuation ne sera jamais coupé. L'espace insécable est un caractère particulier codé par l'**entité** ` `. Voici un exemple :

```
<p>
  Ceci est un paragraphe, veuillez le lire
  attentivement :
</p>
<p>
  Mon paragraphe HTML vous plaît-il ?
</p>
```

Lorsque l'encodage du document est ISO-8859-1 ou UTF-8, on peut écrire l'espace insécable avec le code suivant : ` ` ou ` ` (code **ISO-8859-1** : 160 décimal, 0xA0 hexadécimal).

2.1.19 Les formulaires

HTML permet, via des éléments `form`, d'interagir avec l'utilisateur au-delà des simples liens de navigation. Il est possible de faire faire des choix à l'utilisateur ou de lui demander de remplir des informations. Une fois le formulaire rempli, il est en général envoyé par le navigateur suite à une action de l'utilisateur (p.ex. bouton `submit`) et analysé par un programme sur le serveur web (voir section 3.2.15.1 en page 82). Parmi leurs utilisations courantes on peut noter :

— récupérer des informations sur l'utilisateur (par exemple nom/prénom)
— procéder à des authentifications
— permettre à l'utilisateur de contribuer à un site (via des forums par exemple)
— opérer des recherches ou sélections sur le site
— télécharger un fichier vers le serveur (upload)

2.1.19.1 Définir un formulaire simple

Un formulaire est délimité par la balise `form`. La délimitation permet d'en mettre plusieurs indépendants sur une même page et d'en définir les paramètres généraux d'envoi. Si plusieurs formulaires peuvent exister sur une page, il est interdit de les imbriquer : une même zone ne peut être contenue que par un et un seul formulaire. La balise contient un attribut obligatoire nommé `action`, qui donne l'URL du programme (script) sur le serveur qui traitera ces paramètres de formulaire, et un attribut optionnel `method` qui définit la méthode (**post** (le plus courant), pour les insertions, modifications, ou effacements ; et **get** (par défaut) pour les consultations sans effet de bord, et lorsque cela n'est pas gênant de voir les valeurs transmises dans l'URL).

2.1.19.2 Structuration de la page

La balise de formulaire n'est qu'un délimiteur, elle est destinée à recevoir des éléments de niveau bloc et non du contenu dit "en-ligne" (voir la section 2.1.13 en page 37) . Elle est donc elle-même une balise de type bloc, et à ce titre ne peut pas être incluse dans un paragraphe (ou autre élément qui n'accepte que le contenu en-ligne).

```
<div>
  <form action="..." method="...">
    <p>texte</p>
  </form>
</div>
```

NB : en HTML5, il n'est pas nécessaire de disposer l'élément `form` ni les éléments d'entrée ci-après dans un élément de type bloc explicite, mais cela peut être pratique.

2.1.19.3 Champs d'entrées de données

Commençons d'abord par demander le nom du visiteur. Pour qu'il puisse l'inscrire nous allons créer un champ (balise `input`) de type texte (attribut `type="text"`). À l'affichage le visiteur aura une zone de saisie simple où il pourra écrire du texte.

```
<input type="text" value="exemple" /> <!-- valide HTML5 ou XHTML -->
<input type="text" value="exemple"> <!-- valide HTML5 -->
```

Attention : comme `input` est autofermé, la forme `</input>` n'existe pas.

2.1.19.4 Description du champ

Afin que l'utilisateur puisse savoir à quoi sert ce champ de texte il est nécessaire de lui donner une légende ou un titre. Les descriptions de champs sont faites à l'aide de la balise `label`.

Cette balise est souvent oubliée mais est indispensable à la compréhension de votre formulaire pour les nombreuses personnes employant des méthodes de navigation alternatives. Sans elle, ces personnes seront dans l'impossibilité de comprendre le rôle de chaque champ du formulaire, et donc de le remplir. Un champ correctement symbolisé facilitera d'ailleurs l'accès d'un formulaire à tout le monde puisque cliquer sur la légende activera le champ concerné. Pour un champ texte il s'agit d'y déplacer le **focus clavier**, pour une case à cocher cliquer sur la légende cochera la case. Ce sont ces petits plus qui améliorent l'**ergonomie** pour l'utilisateur final.

Pour indiquer une description de champ, on inclut celle-ci entre les balises de label et on indique, par l'attribut `for`, l'`id` du champ auquel la description est rattachée, comme ceci :

```
<form action="...">
  <p>
    <label for="nom">Nom: </label>
    <input type="text" id="nom">
  </p>
</form>
```

Il existe une autre façon d'utiliser la balise `<label>` : par imbrication : on inclut le champ de saisie entre les balises de label, en même temps que la description :

```
<form action="...">
  <p>
    <label>
      Nom: <input type="text">
    </label>
  </p>
</form>
```

Dans ce cas, il est possible d'omettre les attributs `for` et `id`.

2.1.19.5 Désignation des champs (paramètres)

Le contenu du formulaire est normalement destiné à être envoyé à un programme sur le serveur web. À la réception, le programme doit pouvoir connaître le contenu de chaque élément, donc identifier les différentes données. Cette identification est faite en attribuant un nom à chaque champ du formulaire, via l'attribut `name`. Cet attribut n'est pas nécessairement unique [10]. Vous pouvez par exemple avoir plusieurs formulaires dans une même page, certains ayant des champs

10. et il peut même être multivalué, si votre langage de script côté serveur le supporte, consulter la section 3.2.15.1 en page 83

nommés de la même façon. Le programme serveur qui interprète le formulaire n'aura alors pas à savoir sur quel formulaire l'utilisateur a cliqué, il interprétera les paramètres reçus uniquement à partir de leur nom.

Attention à ne pas confondre le nom (attribut `name`) avec l'identifiant (attribut `id`) : le premier sert à identifier les données qui sont soumises, le second à identifier des éléments dans la page en cours de visualisation (par exemple pour y accéder facilement via le modèle **DOM** en Javascript via `getElementById()`). Seul l'identifiant est unique, deux éléments peuvent envoyer une donnée sous le même nom. Utiliser les mêmes valeurs pour les deux attributs vous épargnera toutefois beaucoup d'erreurs dans vos développements.

```
<input type="text" name="nom" id="identifiant">
```

2.1.19.6 Grouper les champs

Dans un formulaire avec plusieurs zones de saisies il est généralement de bon ton de regrouper les champs par catégories. On peut par exemple séparer les quelques champs qui vous demandent des informations sur votre situation familiale des champs qui demandent vos coordonnées bancaires. L'utilisateur en retirera une meilleure compréhension du formulaire et de son utilisation. L'élément qui remplit ce rôle est `fieldset`. Il suffit de faire délimiter les données appartenant à un même groupe par cette balise. Toutefois, cet élément seul ne permet que de grouper certaines parties du formulaire, il ne renseigne pas sur l'utilité de la partie en question. Vous pouvez donc rajouter un élément `legend` à l'intérieur, son contenu servira de description au groupe.

```
<form action="...">
<fieldset>
  <legend>Adhérent</legend>
  <p><label>Nom: <input type="text"></label></p>
  <p><label>Prénom: <input type="text"></label></p>
</fieldset>
<fieldset>
  <legend>Conjoint(e)</legend>
  <p><label>Nom: <input type="text"></label></p>
  <p><label>Prénom: <input type="text"></label></p>
</fieldset>
</form>
```

2.1.19.7 Les autres types de champs de formulaires

La zone de saisie créée plus haut est intéressante mais est loin de couvrir tous les besoins d'une application web. Cette section détaille quelques champs supplémentaires.

Deux attributs particuliers communs à la plupart des champs de formulaires sont à signaler : `readonly` permet d'empêcher le changement du contenu par l'utilisateur et `disabled` agit comme si le champ n'existait pas (affiché mais non éditable, pas de focus et pas transmis à l'envoi du formulaire). Dans un document XHTML les attributs doivent avoir une valeur, il faut alors leur donner le nom de l'attribut comme valeur : `<input type="text" disabled="disabled">`

Il ne devrait pas être nécessaire de souligner que cette sécurité est du côté du client, votre script exécuté du côté serveur ne devrait pas en dépendre.

2.1.19.7.1 Zone de texte courte La zone de texte courte est l'élément qui a été pris en exemple plus haut. Il s'agit d'une balise `<input>` avec l'attribut `type="text"`. Si l'attribut `value` est présent, il donne la valeur par défaut de la zone de saisie
`<input type="text" value="valeur par défaut">`

Vous pouvez définir la taille à l'affichage en spécifiant un nombre de caractères dans l'attribut `size`. Si l'utilisateur rentre une donnée plus longue il ne la verra pas en entier.

Plus intéressant, vous pouvez aussi limiter la taille de la donnée grâce à l'attribut `maxlength` : `<input type="text" size="10" maxlength="20">`. Faites tout de même attention au fait que cette limitation est faite pour empêcher l'utilisateur honnête de se tromper et de dépasser. Spécifier cet attribut n'empêchera pas l'utilisateur malhonnête d'envoyer une donnée trop longue volontairement.

2.1.19.7.2 Saisie de mots de passe Il existe un élément spécifique à la saisie de mot de passe. En spécifiant `type="password"` à la place de `type="text"` on obtient une zone qui affiche des étoiles à la place des caractères tapés. Cette saisie est appréciée pour les mots de passe, de façon à ce que l'assistance ne puisse pas lire à l'écran.

Attention toutefois : si vous spécifiez une valeur par défaut, même si elle apparaîtra sous forme d'étoiles dans la plupart des navigateurs, la valeur reste en clair dans le code source. Cela veut dire qu'il suffit de demander au navigateur d'afficher le code source pour connaître le mot de passe par défaut. De même, quelqu'un ayant accès au cache du navigateur, ou écoutant sur le réseau, pourra connaître ce mot de passe (`https` excepté). La procédure ne protège donc que de ceux qui regardent l'écran en même temps que l'utilisateur, mais de rien ni de personne d'autre.

2.1.19.7.3 Séries d'options à cocher Pour des séries d'options, il est possible de définir des cases à cocher (`type="checkbox"`). Ces cases ont un fonctionnement spécial : cochées elles envoient la valeur définie par l'attribut `value`, non cochées elles n'envoient rien (comme si elles n'étaient pas là). Il est possible de définir une case comme cochée par défaut en lui ajoutant un attribut vide nommé `checked` (`checked="checked"` en XHTML). Il est fréquent de fournir une liste de cases à cocher pour poser des questions assez proches. N'oubliez alors pas de grouper ces différentes cases grâce à `fieldset` pour en faciliter la compréhension.

```
<fieldset>
 <legend>Conditions générales</legend>

  <p><label>
    <input type="checkbox" value="OK" name="condgen">
    J'ai lu les conditions générales du service
  </label></p>

  <p><label>
    <!-- opt-out -->
    <input type="checkbox" value="OK" name="spam" checked="checked">
    J'autorise l'utilisation de mes coordonnées par la société
  </label></p>
</fieldset>
```

Dans une série de cases à cocher, on aime souvent que l'utilisation d'une case désactive les autres pour sélectionner une option parmi plusieurs. Ces zones d'options (aussi appelées boutons

radio) sont gérées par le type `radio`. Le fonctionnement est similaire aux cases à cocher mais si plusieurs cases ont le même nom (attribut `name`) dans le formulaire, une seule pourra être activée à la fois (l'activation d'une case désactivera les autres). Si le groupement des options lors de leur utilisation se fait par le nom, il est toujours conseillé de grouper aussi par `fieldset` pour permettre à l'utilisateur de prévoir ce fonctionnement et de le comprendre.

```
<fieldset>
  <legend>Civilité</legend>
  <ul>
    <li><label>
      <input type="radio" value="monsieur" name="civilite">
      Monsieur
    </label></li>
    <li><label>
      <input type="radio" value="madame" name="civilite"
                          checked="checked">
      Madame
    </label></li>
    <li><label>
      <input type="radio" value="mademoiselle" name="civilite">
      Mademoiselle
    </label></li>
  </ul>
</fieldset>
```

2.1.19.7.4 Listes de sélection Les listes de sélection (balise `select`) sont une autre manière de choisir entre plusieurs options. Quand l'attribut `size` n'est pas défini il s'agit d'une liste déroulante, sinon la zone est une liste de taille fixe (l'attribut définissant le nombre de choix à afficher à la fois). Si l'attribut `multiple` est défini il est possible pour l'utilisateur de choisir plusieurs options dans la liste (généralement avec les touches `control` et `shift` en plus de la méthode de sélection habituelle). Les différents choix possibles sont définis par les balises `option` contenues dans le `select`. Chaque balise a un attribut `value` qui définit la valeur de l'option et contient sa description. Une option présélectionnée a de plus l'attribut `selected` défini.

Si vous pouvez facilement faire plusieurs groupes distincts dans les options, il vous est recommandé de les grouper grâce à l'élément `optgroup` afin d'aider l'utilisateur à s'y retrouver. Il est similaire à `fieldset` mais au lieu d'avoir un sous-élément `legend` il contient un attribut `label`. Ces groupes ne peuvent pas être imbriqués. Les clients WWW peuvent interpréter des groupes comme des sous menus ou comme des titres.

```
<select name="pays">
  <optgroup label="Europe">
    <option value="fr" selected="selected">France</option>
    <option value="it">Italie</option>
  </optgroup>
  <optgroup label="Asie">
    <option value="ch">Chine</option>
  </optgroup>
</select>
```

2.1.19.7.5 Zones de saisie étendue La zone de saisie `input type="text"` vue préalablement n'est adaptée que pour des courtes chaînes et ne permet pas d'éditer simplement des textes sur plusieurs lignes. Pour cela il existe l'élément `textarea`. L'affichage sera un cadre de texte éditable, sa taille peut être définie par les attributs `cols` (nombre de caractère par ligne) et `rows` (nombre de lignes). Le texte défini par défaut est celui contenu entre la balise de début et la balise de fin. N'oubliez pas que ce texte ne remplace pas la balise `label`.

```
<textarea cols="80" rows="4" name="article">
  Tapez ici votre article
</textarea>
```

2.1.19.7.6 Envoi de fichiers Il vous est possible d'envoyer des fichiers via les formulaires. L'élément `<input type="file">` permettra à l'utilisateur de sélectionner un fichier local à envoyer. Si théoriquement l'attribut `value` peut définir l'emplacement par défaut du fichier, les navigateurs désactivent généralement cette possibilité pour des raisons de sécurité.

Vous pouvez restreindre le choix à une liste de types de contenu autorisés séparés par des virgules, en la spécifiant dans l'attribut `accept`. Les types de contenu sont les mêmes que ceux des entêtes **HTTP Content-type** et **Accept**. On y trouve en vrac **text/plain** pour les textes, **text/html** pour le HTML, **application/xml** pour le XML, **image/png** pour les PNG, ou plus généralement image/* pour tous les types d'images. Le support par les navigateurs de cette propriété n'est malheureusement pas toujours très bon, il ne s'agit au plus que d'une aide pour l'utilisateur. Vous pouvez toutefois trouver une liste détaillée des types de contenu standardisés via la page de l'IANA [18] sur les types **MIME**.

```
<form action="..." accept="text/plain,text/html"><p>
  <label>Fichier à envoyer <input type="file"></label>
</p></form>
```

2.1.19.7.7 Champs cachés Un dernier type de champ est possible pour transférer des données : des champs invisibles. Ces champs sont faits pour pouvoir définir sur la page des valeurs qui ne changeront pas ou qui n'ont pas besoin d'interactions directes avec l'utilisateur. Ils seront envoyés avec le formulaire au même titre que les autres champs. Il vous suffit de définir un élément `<input type="hidden">` avec une valeur et un nom en attributs. Attention toutefois, ce n'est pas parce que le champ n'est pas visible dans le rendu que les valeurs ne peuvent pas être lues ou modifiées par le visiteur. Ne mettez rien de confidentiel ou ayant trait à la sécurité (à part évt. une valeur aléatoire unique).

2.1.19.8 Cible et méthode d'envoi du formulaire

Un formulaire est habituellement destiné à être traité par un programme sur le serveur. L'attribut `action` permet de définir l'URI où sera envoyé le formulaire pour traitement. Cet attribut est le seul obligatoire.

Il existe deux manières d'envoyer le contenu d'un formulaire (les **paramètres**), gérées avec l'attribut **method** : **post** et **get**. Cette dernière se voit habituellement car les différentes valeurs du formulaire sont visibles dans l'URL de la page résultante : c'est la méthode utilisée par défaut. L'autre méthode (post) fait passer les informations à envoyer dans le corps de la requête HTTP, et permet d'envoyer des informations plus conséquentes ou nécessitant un encodage spécifique (p.ex. l'envoi de fichiers). Il vous est recommandé d'utiliser la méthode get quand

aucune information envoyée n'est exagérément secrète (pas de mot de passe), que la quantité d'information est limitée et que deux soumissions avec les mêmes paramètres renvoient toujours la même information et n'a pas d'effets de bord (**idempotente**). Si un de ces trois critères n'est pas vérifié, vous devriez envisager la méthode post.

```
<form action="http://www.domaine.ext/rep/script" method="post">
   <!-- contenu du formulaire -->
</form>
```

Note : une erreur très courante est d'écrire les valeurs de l'attribut `method` en majuscule. En fait, elles doivent être en minuscule : `method="get"` ou `method="post"`.

2.1.19.9 Envoi et traitement du formulaire

Jusqu'à présent nous avons rempli notre formulaire mais nous n'avons toujours aucun élément permettant de déclencher l'envoi. En ajoutant un champ de type `submit`, vous obtiendrez un bouton qui déclenche la soumission des données. Le texte par défaut du bouton est déterminé par le navigateur, vous pouvez le changer grâce à l'attribut `value`. Si ce bouton de validation a un nom (attribut `name`) et une valeur, cette valeur sera transmise lors de l'envoi : une telle fonctionnalité permet d'utiliser plusieurs boutons et de déterminer lequel a été utilisé.

```
<input type="submit" value="Envoyer" name="e1">
```

Il est aussi possible d'utiliser une image comme bouton d'envoi (`<input type="image">`). Dans ce cas l'adresse de l'image devra être spécifiée via l'attribut "src" et un texte alternatif devra être fourni dans l'attribut "alt", comme une image classique. Lorsque l'utilisateur envoie le formulaire en cliquant sur l'image, la position du clic dans l'image est envoyée en plus des données normales. Dans ce cas, la donnée `monimage.x` est la position horizontale en pixel et la donnée `monimage.y` est la position verticale, où `monimage` est le nom du champ de formulaire contenant l'image. Il est aussi possible de définir les zones réactives de l'image grâce aux attributs `ismap` et `usemap`, ces propriétés fonctionnent alors comme pour les images cliquables classiques.

2.1.19.10 Encodage de transfert

Lorsqu'un formulaire est envoyé, les données sont protégées avec un encodage de transfert (voir section 1.1.6.11 en page 11), de façon à ne pas perturber la requête HTTP.

L'attribut `enctype` de l'élément `form` définit la façon d'encoder les données. Par défaut la valeur est **application/x-www-form-urlencoded**. Vous devrez utiliser **multipart/form-data** si votre formulaire contient des fichiers à envoyer tels quels. Pour finir, l'attribut `accept-charset` définit une liste de jeux de caractères utilisables pour l'envoi, séparés par des virgules ou espaces. Cette dernière propriété supporte des valeurs comme **UTF-8** ou **ISO-8859-1** mais pour compatibilité avec les navigateurs qui ne s'en servent pas, il est préférable de savoir gérer n'importe quel codage et de faire des conversions à la volée en se basant sur les entêtes HTTP.

2.1.19.11 Notions d'accessibilité et d'utilisabilité

Comprendre un formulaire est souvent complexe pour des visiteurs sur navigateurs non graphiques. Pour ceux-ci une description complète du formulaire est une nécessité. Cela passe par l'utilisation de la balise `label` pour tous les champs, l'exploitation des groupes `fieldset`

```
<div>
<form action="script-traitement.php" method="post">
  <p>
    <label for="nom">Nom: </label>
    <input type="text" name="nom" id="nom">
  </p>

  <p>
    <label for="prenom">Prénom: </label>
    <input type="text" name="prenom" id="prenom">
  </p>

  <fieldset><legend>Sexe:</legend>
    <ul><li>
      <input type="radio" value="h" name="sexe" id="s_h">
      <label for="s_h">Homme</label>
    </li><li>
      <input type="radio" value="f" name="sexe" id="s_f">
      <label for="s_f">Femme</label>
    </li></ul>
  </fieldset>

  <p>
    <label for="pays">Pays: </label>
    <select name="pays">
      <optgroup label="Europe">
        <option value="fr">France</option>
        <option value="it">Italie</option>
      </optgroup>
      <optgroup label="Asie">
        <option value="jp">Japon</option>
      </optgroup>
    </select>
  </p>
  <p><input type="submit" value="Envoyer"></p>
</form>
</div>
```

Figure 2.1 – Exemple d'un formulaire plus complet

et optgroup dès que possible, mais aussi par la définition des zones dans une image cliquable grâce à l'attribut usemap et l'élément map (se reposer uniquement sur la position du clic rend la fonctionnalité inaccessible pour les navigateurs non graphiques ou avec un pointage alternatif).

Malheureusement tous les clients ne mettent pas à profit ces balisages. Il est donc important de vérifier la compréhension du formulaire lors d'une lecture linéaire : faire en sorte que la description suive ou précède toujours directement le champ décrit, et éviter de disperser les éléments du formulaire dans la page. Vous pouvez par exemple tester le rendu avec le navigateur Lynx ou w3m pour vérifier que le formulaire, et le rôle de chaque élément, y sont compréhensibles.

Suivant les applications et environnements d'exécution, on ne peut pas toujours garantir l'exécution du **Javascript** : il est donc nécessaire de permettre l'accès au formulaire sans scripts. L'ajout d'un bouton de soumission (`<input type="submit" value="envoyer">`) et une destination correctement remplie pour le formulaire sont indispensables. Les scripts pouvant, quant à eux, prendre la main en complément des éléments normaux lorsque le navigateur les supporte.

Dans le même esprit, si une validation des données par Javascript (ou en **HTML5**) peut être faite pour aider l'utilisateur à remplir les cases, elle ne remplace pas une validation par le programme sur le serveur car rien ne garantit que le visiteur utilise Javascript et n'envoie pas les données directement.

Inversement, lorsque le Javascript est supporté, il peut aussi limiter l'accessibilité du formulaire. C'est particulièrement le cas des scripts qui déclenchent automatiquement l'envoi après sélection d'une entrée (par exemple dans une liste déroulante) : certains logiciels ne permettent pas de naviguer dans une liste sans sélectionner un à un les éléments, ce qui activerait le Javascript dès le premier élément et déclencherait l'action. Ces scripts sont donc à éviter sauf réflexion approfondie d'utilisabilité, d'autant plus qu'HTML5 permet pas mal de retours à l'utilisateur sur les problèmes des données entrées.

La norme **WCAG** (*Web Content Accessibility Guidelines*) propose des bonnes pratiques pour améliorer l'accessibilité, quelque soit le terminal.

2.1.19.12 Accès rapide par raccourcis clavier et ordre de parcours

La structure est une chose, mais tel quel le formulaire n'est pas encore parfait pour un accès avec autre chose qu'une souris. Pour une meilleure accessibilité, il peut être intéressant de rajouter des combinaisons d'accès rapide : avec un attribut accesskey sur quelques éléments spécifiques vous permettrez l'accès direct à un élément du formulaire (ou tout autre élément d'ailleurs). C'est particulièrement utile pour des formulaires qui servent à la navigation, comme un formulaire de recherche.

Attention : les expériences en terme d'accesskeys montrent que ce système a des limites et défauts, notamment le fait que chaque navigateur ou OS propose une autre séquence de touche [11] et que les actions des touches elle-mêmes ne sont pas standardisées [12].

Il existait un moyen de spécifier un ordre de parcours dans les champs de formulaire via l'attribut tabindex. C'est aujourd'hui considérée obsolète, sauf pour la valeur 0 qui permet de signaler qu'un élément HTML par défaut non accessible (par exemple un div) est accessible au clavier : pour le reste, le mieux est que l'ordre de parcours soit logique par rapport à la présentation du document HTML

11. https://www.w3schools.com/tags/att_global_accesskey.asp
12. http://reference.sitepoint.com/html/a/accesskey

2.1.19.13 Recommandations d'utilisabilité (UX)

L'accès technique au formulaire étant fait, il reste une dernière touche importante : aider l'utilisateur à se servir du formulaire.

Indiquez toujours si un champ est obligatoire par une étoile ou par une simple ligne qui nomme les valeurs nécessaires au traitement si vous pensez qu'il serait trop lourd de le signaler à chaque fois.

Lorsqu'une erreur survient dans le traitement et qu'elle n'a pas pu être détectée par la validation du formulaire côté client en HTML5, voire Javascript, indiquez précisément ce qui doit être fait comme correction et pointez la donnée manquante ou erronée. Renvoyez le message d'erreur sur la même page que le formulaire, lui-même devant reproduire toutes les données entrées précédemment, ou conserver les données valides, et ne redemander que les manquantes ou erronées. L'utilisateur ne doit pas avoir à saisir à nouveau les données correctement introduites, ni à chercher où est son erreur.

Pour le traitement d'erreur simple, une validation locale du formulaire en HTML5 (via le typage et les **regex**), ou pour des cas plus complexes du code Javascript peut améliorer largement l'expérience utilisateur. Nous verrons toutefois qu'une validation doit toujours être faite au minimum côté serveur pour des raisons de sécurité.

Inversement, si aucune erreur n'apparaît, et à défaut d'une réponse spécifique, informez l'utilisateur que les données ont été reçues et traitées avec succès. Ne redirigez pas directement l'utilisateur vers la page d'accueil ou un nouveau formulaire vierge sans lui donner de confirmation sur le traitement du précédent.

2.2 Feuilles de style (CSS)

2.2.1 Introduction

Les feuilles de style sont le seul et unique mécanisme préconisé pour contrôler l'aspect visuel d'un document aujourd'hui. La plupart des éléments et attributs qui permettaient d'exercer un tel contrôle dans les versions antérieures du langage (HTML 3.x et inférieures) sont en effet désormais déconseillés. Une feuille de style regroupe des règles de présentation que l'on peut appliquer à des éléments (X)HTML dans un document. Ces règles sont exprimées dans un langage indépendant de (X)HTML. En HTML5, c'est le langage **CSS3** (Cascading Style Sheet) qui tend à s'imposer : de plus en plus, toutefois, le CSS3 est généré par des *translateurs* : par exemple le langage de plus haut niveau **SCSS** est compilé en CSS par **Sass**.

Cette introduction n'a pas d'objectifs très ambitieux et est complétée par la présentation du cours, le sommaire de référence polycopié [12] et de nombreux liens[6][7][8][9][10][11] et exercices.

2.2.2 Association de feuilles de style à un document

Les feuilles de style peuvent être associées de différentes manières :

2.2.2.1 Feuille de style externe dans un fichier séparé

C'est la forme recommandée pour les feuilles de style, notamment pour avoir un format homogène sur toutes les pages d'un site.

```
<html>
  <head>
    <title>Titre</title>
    <link rel="stylesheet" type="text/css" href="formats.css">
  </head>
  <body>
  </body>
</html>
```

ou encore

```
<html>
  <head>
    <title>Titre</title>
    <style type="text/css" title="MonStyle">
      @import "formats.css";
    </style>
  </head>
  <body>
  </body>
</html>
```

2.2.2.2 Feuille de style incluse dans le document

Ceci est rarement utilisé, mais cela peut être pratique quand on envoie un document HTML par mail (on peut ainsi y intégrer directement la feuille de style à l'intérieur).

```
<html>
  <head>
    <title>Titre du fichier</title>
    <style type="text/css">
    <!--
      /*  ... ici sont définis les formats ... */
    -->
    </style>
  </head>
  <body>
  </body>
</html>
```

Comme les très anciens clients WWW pourraient être tentés d'afficher le contenu de l'élément `style` ci-dessus, une habitude malheureuse – voir ci-dessus – est de jouer sur le fait que les commentaires HTML et CSS sont différents et ignorés de part et d'autre.

2.2.2.3 Style propre à un élément seulement

Dans certains cas qui devraient rester rare, il est possible de définir des formats propres à un seul élément.

```
<html>
  <head>
    <title>Titre du fichier</title>
  </head>
  <body>
    <h1 style="color:#b0b0b0;background-color:#ffa0a0;">Titre spécial</h1>
  </body>
</html>
```

2.2.3 Appliquer un style à des éléments : les sélecteurs

2.2.3.1 Sélection d'éléments

Que la feuille de style soit dans un fichier séparé ou incluse dans le document, la définition des formats reste identique. Ainsi, le fichier `format.css` pourrait par exemple contenir, pour définir le style de l'élément `body` et des éléments `h1` d'un document :

```
body {
   background-color: #ffffcc;
   margin-left: 100px;
}

h1 {
   font-size:48pt;
   color: #ff0000;
   font-style: italic;
   border-bottom: solid thin black;
}
```

On remarque d'ailleurs ci-dessus l'encodage hexadécimal **RGB** (rouge-vert-bleu) des couleurs.

2.2.3.2 Sélections de classes

On utilise la notation pointée pour associer un style à une classe. L'avantage des classes est qu'elles peuvent s'appliquer à plusieurs types d'éléments, tant que vous ne limitez pas à un élément particulier comme ci-dessous :

```
<!DOCTYPE html>
<html>
   <head>
      <title>Exemples CSS</title>
      <style type="text/css">
         /* pour tout élément de classe error, p.ex. p class="error" */
         .error {
            color: red;
            font-style: italic;
         }

         /* on précise ou redéfinit les éléments pour un élément de type h1 */
```

```
         h1.error {
            background-color: yellow;
            color: black; /* si pas spécifié, h1.error
                              hériterait .error aussi
                            */
         }
      </style>
   </head>
   <body>
      <h1>Normal</h1>
      <p>normal</p>
      <p class="error">Erreur</p>
      <h1 class="error">Erreur</h1>
      <p>normal</p>
      <p class="error">Erreur</p>
      <div class="error">
         <h1>Normal</h1>
         <p>normal</p>
         <p class="error">Erreur</p>
         <h1 class="error">Erreur</h1>
         <p>normal</p>
         <p class="error">Erreur</p>
      </div>
   </body>
</html>
```

Ce qui précède soulève la question de la portée des CSS : comme on peut le voir ci-dessus avec l'exemple du `div`, un style appliqué à un élément s'applique à tous les éléments qui le contiennent : c'est la propriété d'**héritage** des CSS : à l'exception de `margin`, `border` et de `padding`, toutes les définitions CSS non contradictoires sont héritées automatiquement.

De plus, les définitions sont cumulatives si elles ne sont pas contradictoires : par exemple ici, le style italique de la classe `.error` s'applique au `h1` de classe `error` même s'il n'est pas spécifiquement défini dans `h1.error`.

Par contre, lorsqu'une définition contradictoire (différente) existe dans un sélecteur plus spécifique, comme par exemple `color` ci-dessus, les règles de priorité de la **spécificité** s'appliquent (voir 2.2.4 en page 56) : ci-dessus, la couleur noire s'applique dans tout `h1` de classe erreur malgré la définition précédente.

2.2.3.3 Sélection d'un élément unique identifié

Un élément marqué par un attribut (X)HTML `id` d'une valeur particulière – unique sur le document (X)HTML – peut être référencé par la notation `#valeur`, par exemple :

```
   /* pour l'élément dont l'attribut id vaut error */
   #error {
      font-weight: bold;
   }
```

2.2.3.4 Autres sélecteurs

En plus de la combinaison de sélecteurs par des virgules (ce qui correspond à un OU logique), de nombreux autres sélecteurs conditionnels sont apparus dans CSS2 et CSS3, qui permettent par exemple des hiérarchies d'éléments ou même de sélectionner p.ex. une ligne sur deux dans une table, ce qui en fait un mécanisme très puissant pour simplifier le code côté serveur que l'on utilisait originellement pour ce genre d'opérations.

2.2.4 Priorité des spécifications CSS : spécificité, héritage et cascade

Les Cascading Style Sheets s'appliquent avec les propriétés suivantes :

spécificité plus une sélection d'élément est précise, plus elle est prioritaire ; p.ex. `h1.error` ci-dessus est plus spécifique que `.error` ; la spécificité se calcule précisément avec un algorithme défini dans les spécifications CSS, influençable par le qualificateur `!important` – à ne pas utiliser à tort et à travers [13].

héritage un élément hérite des propriétés CSS de son parent ; avec quelques exceptions comme p.ex. les marges et les remplissages (padding) sauf si vous les déclarez avec : `margin: inherit`

cascade CSS décide l'application des déclarations en fonction des sélecteurs, d'un premier tri lié à où la déclaration a été faite et l'importance de la déclaration, puis calcule la spécificité. En cas d'égalité, c'est la dernière règle définie qui est appliquée.

2.2.5 Indépendance du média et du terminal

2.2.5.1 Media Types

Un des buts fondamentaux d'un design est de pouvoir présenter l'information de manière adéquate que l'on visualise (screen) ou que l'on imprime (print) un document.

Pour ce faire, CSS2 propose la notion de **media types** :

```
<html>
   <head>
      <title>Exemple Media Types</title>
      <style type="text/css">
         @media screen {
            p.adapt { font-size: 18pt; }
         }
         @media print {
            p.adapt { font-size: 8pt; }
         }
      </style>
   </head>
   <body>
      <p class="adapt">Ceci est un texte qui s'adaptera au media</p>
      <p>Ceci est un texte</p>
   </body>
</html>
```

13. http://css-tricks.com/when-using-important-is-the-right-choice/

En plus des deux types vus ci-dessus, notons par exemple *braille*, *handheld* ou *projection*.

2.2.5.2 Responsive Web Design

L'approche *Responsive Web Design* a pour but de rendre l'expérience utilisateur bonne, quelque soit le terminal utilisé (ordinateur à plusieurs résolutions, tablette, téléphone portable, imprimante, ...). Pour ce faire, il y a bien sûr quelques contraintes, comme l'utilisation de dimensions relatives, et l'introduction des **media queries** en CSS3 :

```
<html>
   <head>
      <title>Exemple Media Queries</title>
      <style type="text/css">
         /* small */
         @media screen and (max-width: 500px) {
            body {
               font-size: 60%;
            }
            div#droite {
               background: red;
            }
         }
         /* big */
         @media screen and (min-width: 800px) {
            body {
               font-size: 140%;
            }
            div#droite {
               background: blue;
            }
         }
         /* average */
         @media screen and (min-width: 501px) and (max-width: 799px) {
            body {
               font-size: 100%;
            }
            div#droite {
               background: green;
            }
         }
         div#gauche, div#milieu, div#droite {
            float: left;
         }
         div#milieu {
            background: grey;
         }
         div#dessous {
            clear: both;
            background: yellow;
         }
```

```
    </style>
  </head>
  <body>
    <div id="gauche">
      <p>Un contenu, gauche</p>
    </div>
    <div id="milieu">
      <p>Un contenu, milieu</p>
    </div>
    <div id="droite">
      <p>Un contenu, droite</p>
    </div>
    <div id="dessous">
      <p>Un contenu, dessous</p>
    </div>
  </body>
</html>
```

Cet exemple illustre aussi le positionnement flottant utilisé pour remplacer la mise en page par colonnes fixes ou par tableau.

Il ne faut pas négliger non plus l'impact de la taille des données transférées par le client – cela peut complètement dégrader l'interactivité d'une application.

2.3 Javascript

Javascript, après une création au sein de Netscape et une variante dans Microsoft Internet Explorer (JScript) est aujourd'hui un langage standardisé (Javascript 2.0, sur-ensemble de ECMA-262-4) qui est disponible – dans divers dialectes du langage et surtout de l'environnement du navigateur en **DOM** – sur la quasi totalité des clients WWW.

Javascript est orienté objet (à prototypes : création de nouveaux objets par clonage), non typé et interprété et exécuté sur le client, au sein d'une **sandbox** confinée (hors extensions du langage).

Javascript permet de réagir à des événements précédemment associés à des actions et des éléments (**programmation événementielle**), de contrôler des actions et transférer des données (p.ex. via des requêtes **Web services**) et d'agir sur la représentation interne du client WWW (modèle **DOM**, Document Object Model).

Le développeur Javascript doit constamment faire attention à la compatibilité des diverses implémentations : des bibliothèques ou **framework**s comme **JQuery** permettent de régler la plupart de ces problèmes.

Une excellente introduction à Javascript se trouve dans [14].

Notons que Javascript peut également s'utiliser pour le développement *côté serveur* (p.ex. **NodeJS**), par exemple si l'on veut éviter de devoir supporter plusieurs langages ou pour des fonctionnalités avancées comme PUSH (via NodeJS.SocketIO).

Javascript est également un langage fondamental de certains frameworks mobiles multiplateformes (p.ex. PhoneGap).

2.4 Autres langages côté client

A une certaine époque, les langages suivants côté client étaient populaires : applets Java et **Webstart**, le **Flash** ...

Aujourd'hui, Javascript est devenu le langage de base, supportant tout ce que l'on attend d'un navigateur grâce aux APIs HTML5 (lecture vidéo, accès aux périphériques des smartphones comme le GPS ou l'accéléromètre, graphisme 2D et 3D haute performance avec WebGL, la télé- et visiophonie IP, ...), mais l'on trouve aussi quelques langages descriptifs spécialisés comme **SVG**, basé XML.

Même dans le domaine des applications mobiles, le besoin d'applications natives (Android Java ou Kotlin, iOS Swift) n'est plus toujours justifié en particulier lorsque la performance de calcul n'est pas critique et si les API nécessaires existent. Les applications mobiles web apportent le multiplateforme et l'intégration web.

Chapitre 3

Développement côté serveur

Sommaire

3.1 Introduction **61**
3.1.1 Avantages du développement côté serveur 61
3.1.2 Principes de fonctionnement 62
3.1.3 Langages, environnements et frameworks côté serveur 62
3.2 Le langage PHP **63**
3.2.1 PHP en bref 63
3.2.2 Principe de fonctionnement 64
3.2.3 Environnement de travail 66
3.2.4 Syntaxe de PHP 67
3.2.5 Opérateurs 67
3.2.6 Erreurs 67
3.2.7 Structures de contrôle 67
3.2.8 Variables 71
3.2.9 Constantes 74
3.2.10 Chaînes de caractères 75
3.2.11 Tableaux et tableaux associatifs 77
3.2.12 Fonctions 80
3.2.13 Namespaces 81
3.2.14 Orientation objet 81
3.2.15 Exemples courants du web 82
3.2.16 Identificateurs réservés 88
3.2.17 Internationalisation et régionalisation 88

Le but de ce chapitre est de donner quelques informations générales sur le développement de base d'applications web côté serveur, et plus particulièrement sur le langage PHP.

3.1 Introduction

3.1.1 Avantages du développement côté serveur

Le développement côté serveur a de nombreux avantages, lorsqu'on le compare au développement d'applications lourdes côté client : la portabilité des applications qui ne dépendent plus des

contraintes de chaque poste de travail (logiciels installés, compatibilité de l'OS voire du matériel, performance) mais d'un environnement relativement standard qu'est le client web, la simplicité des environnements de développement et la relative stabilité de l'ensemble, ainsi que la possibilité d'intégrer des langages, environnements et technologies de manière transparente, très souvent en logiciel libre. Les problèmes de compatibilité d'HTML et des CSS existent toujours, mais vont en diminuant et peuvent être complètement éradiqués par l'utilisation de frameworks puissants côté client.

Enfin, une approche hybride est de plus en plus adoptée : que cela soit l'application lourde côté client, le client mobile natif ou *progressive*, ou encore l'application web riche HTML/CSS, des **Web services** côté serveur seront interrogés dans toute application moderne.

3.1.2 Principes de fonctionnement

Techniquement et dans son implémentation la plus simple, le côté serveur envoie ou génère de l'HTML, des CSS et éventuellement du Javascript et/ou tout contenu multimédia (images, etc), qui seront exécutés ou traités du côté client, tout en maintenant du contexte client (cookie p.ex.) et du contexte serveur (base de données, session).

Très souvent, on développera aussi du côté client en code Javascript, en plus du côté serveur dans divers langages : l'exécution côté client de code Javascript peut ajouter de l'utilisabilité, une meilleure interactivité (**AJAX**) et une meilleure performance ou un accès à du matériel local spécifique, grâce aux API HTML5 qui permettent de créer des applications riches accédant à du matériel local (GPS, webcam, etc), sans autre composant logiciel.

3.1.3 Langages, environnements et frameworks côté serveur

Les langages, frameworks et environnements côté serveur sont multiples, citons par exemple :

— Perl (CGI, **fast-cgi**, **mod-perl**), ou framework MVC **Catalyst** [19] ou **Perl Mojolicious**
— Python (classique ou framework **Django** p.ex.)
— servlets Java
— PHP (classiquement intégré à Apache via mod-php, **php-fpm** pour le support de fortes charges ou la séparation des applications, via fast-cgi), frameworks MVC Symfony, Zend.
— langages orientés **templates** (chablons), comme par exemple les **SSI** (server-side-includes), le **Template Toolkit**, **JSP**, **Microsoft ASP**, XSLT, PHP dans une certaine mesure, etc.
— tout exécutable appelable par **CGI** ou fast-cgi : y compris C ou C++ compilés, scripts shells, etc – en particulier sur les systèmes embarqués.

Ils diffèrent par la puissance du langage de base (interprété, compilé, typage dynamique ou non, orientation objet), par les bibliothèques de support (p.ex. génération automatique de code HTML, JS, etc), par la façon d'interfacer au serveur web (templating ou non) et par le support d'une méthodologie particulière (framework MVC).

3.2 Le langage PHP

3.2.1 PHP en bref

3.2.1.1 Introduction

PHP (au début *Personal Home Page*, aujourd'hui *PHP Hypertext Preprocessor* – acronyme récursif) est un langage spécialement conçu pour le développement d'applications web. Même si certains développeurs l'utilisent aussi dans d'autres domaines d'application (p.ex. la configuration système dans la distribution de firewall **MonoWall**, voire le traitement de données – parsing), il ne s'agit pas d'un langage générique comme C, C++, Perl, Python ou Java.

A l'instar des technologies basées sur les templates, comme ASP et JSP, il n'est pas nécessaire de générer la totalité du document HTML à renvoyer au client : on peut placer des instructions PHP au sein du template (entourées de balises particulières `<?php` et `?>` qui indiquent au serveur d'exécuter le code PHP).

PHP domine assez largement le marché des applications web simples, en particulier pour ce qui est de la couche présentation (templating). Même si les implémentations de Web services sont encore rares en PHP et sont plutôt implémentées en général dans d'autres langages (Perl, Java, Python...), elles existent et sont amenées à progresser. L'existence d'une base logicielle importante et de frameworks puissants en fait aussi un bon langage pour des applications web plus complexes.

Il est à noter que PHP fait son entrée dans le monde du **cloud**, en particulier de type **PaaS** : en effet, il existe une implémentation Java de PHP (incluse dans Resin), où le PHP est compilé en Java bytecode, et qui permet de l'exécuter dans des clouds Java classiques (p.ex. Google App Engine, encore qu'il supporte maintenant aussi expérimentalement le PHP nativement).

Quelques caractéristiques de ce langage :

- interprété (contrairement p.ex. au C qui est compilé)
- faiblement typé (capacité d'une variable à changer de type en fonction du contexte)
- objet (dès PHP5), mais possibilité de l'utiliser comme un langage impératif
- code embarqué dans un document HTML, portant généralement l'extension `.php`

3.2.1.2 Forces et faiblesses

Points forts :

- simple à prendre en main grâce à sa syntaxe proche du C
- **libre** (licence **GNU GPL**), et généralement gratuit
- disponible sur de nombreux systèmes d'exploitation (UNIX, GNU/Linux, Microsoft Windows, Apple MacOS X) et supporté par une grande majorité des serveurs web
- interfaçable à un grand nombre de SGBD
- de nombreuses fonctions permettant un domaine d'application très large (bases de données, connexions Internet (mail, annuaires LDAP, news, ftp...), gestion de sessions, génération et modification d'images, PDF, Flash, XML...)
- grande bibliothèque de logiciels packagés et installables via **Composer** (https://getcomposer.org/)

Faiblesses :

- évolution parfois trop rapide du langage : il faut toujours vérifier dans les manuels que des fonctions utilisées ne sont pas marquées comme bientôt obsolètes.

— nombreux sont les codes existants qui sont simplement mal programmés, notamment au vu des **register-globals** ou des **magic-quotes**. Il faut faire particulièrement attention lors de l'intégration de logiciels tiers.
— sans framework particulier, l'interfaçage aux bases de données est fastidieux, vous lie à une BD particulière, voire même vous expose à des dangers (injection de données) sans échappement manuel systématique
— le concept orienté objet a été rajouté après-coup et n'est pas intégré toujours élégamment au langage
— les expressions régulières s'approchent de celles de Perl, sans toutefois égaler sa puissance d'expression et sa sécurité ; les implémentations sont diverses et changeantes.

3.2.1.3 Historique

On doit la première version de PHP à Rasmus LERDORF qui l'a mise au point pour ses propres besoins en 1994. Cette version était destinée à son usage personnel, d'où le nom (Personal Home Page). Vers 1995, une version qui permettait l'exécution de quelques macros fut mise à disposition du public.

PHP 5 est sorti en juillet 2004 après un long développement et plusieurs pré-versions. Il est régi par son moteur, le Zend Engine 2.0 qui implémente un nouveau modèle objet (avec modificateurs d'accès), une base de données embarquée (SQLite) et des dizaines de nouvelles fonctionnalités, notamment une gestion simplifiée du XML, la gestion des exceptions et la notion d'interface.

Les versions actuelles de PHP ajoutent un meilleur support de l'orienté-objet, la possibilité de typer les fonctions, la généralisation des exceptions, des améliorations de performance et des nouveaux opérateurs.

PHP est utilisé par des centaines de milliers de développeurs, et plusieurs millions de sites web indiquent qu'ils sont configurés avec PHP.

3.2.2 Principe de fonctionnement

PHP a été conçu pour générer dynamiquement du contenu. Il est donc étroitement lié au serveur HTTP. Son principe de fonctionnement est le suivant :

1. le client appelle une page PHP depuis son client web par un URL [1] de la forme `http://` ou `https://`
2. le serveur HTTP reçoit la requête et la transmet à l'interprète PHP
3. l'interprète PHP exécute le code se trouvant à l'intérieur de toutes les zones PHP (délimitées par des **balises**) et remplace ces balises par le résultat de leur exécution
4. le document obtenu (HTML, XML, autre) est renvoyée au client par l'intermédiaire du serveur HTTP

Lors de l'étape 3, le script PHP peut, entre autres, rechercher des informations dans une base de données ou configurer des cookies ou d'autres parties de l'entête HTTP (si aucune sortie HTML n'a déjà été faite).

Lors de l'étape 2, l'interprète PHP , décode les paramètres éventuels de formulaires ou de liens (**post**, **get**), les **cookies** éventuels, et met tout cela à disposition du script sous forme de variables.

1. une erreur de débutant est de charger le fichier PHP sous forme de chemin local dans le système de fichiers du client, plutôt que d'accéder à l'URL du serveur

3.2.2.1 Balises HTML

Lorsque PHP commence à traiter un fichier, il ne fait qu'afficher (donc renvoyer au serveur, qui lui-même le renvoie au client) le texte contenu dans le fichier PHP, jusqu'à ce qu'il rencontre une balise spéciale qui lui indiquera qu'il peut interpréter le code qui suit.

L'interprète PHP va alors exécuter ce code, jusqu'à ce qu'il rencontre une balise de fin de code PHP. A ce moment là, il retourne en mode texte, et affiche simplement le contenu.

C'est ce mécanisme de **templating** (chablon) qui vous permet d'inclure du code PHP dans des pages HTML : tout ce qui est placé hors des balises PHP est affiché sans modification, tandis leur contenu est exécuté.

Il y a plusieurs méthodes pour délimiter des blocs de code ou d'expressions PHP, dont seules les deux suivantes sont recommandées :

1. `<?php echo "méthode classique et recommandée pour du code"; ?>`
2. `<?= expression ?>` (sans besoin d'`echo` : évalue l'expression, par exemple une variable, et l'affiche (sans échappement)

3.2.2.2 Exemple

Cette page HTML contenant du PHP, telle qu'elle doit être stockée comme fichier sur le serveur (p.ex. `test.php` à la racine **htdocs**, `/var/www` sous UNIX) :

```
<html>
   <head>
      <title>Heure</title>
   </head>
   <body>
      <p>Sur le serveur, il est actuellement
         <?= date("H:i:s"); ?>.
      </p>
   </body>
</html>
```

sera renvoyée ainsi au client web (après exécution comme ci-dessus) :

```
<html>
   <head>
      <title>Heure</title>
   </head>
   <body>
      <p>Sur le serveur, il est actuellement
         14:06:02.
      </p>
   </body>
</html>
```

et sera affichée de cette façon dans le client web :

```
Sur le serveur, il est actuellement 14:06:02.
```

L'expression à l'intérieur de la balise <?= . . . ?> a été évaluée côté serveur par l'interprète PHP. L'évaluation de cette expression remplace la balise dans le document qui sera ensuite renvoyée au client.

Du point de vue du client, rien ne permet de distinguer une page renvoyée par un script PHP d'une page HTML statique, vu que les balises PHP ont été remplacées.

3.2.3 Environnement de travail

3.2.3.1 Principes

L'interprète PHP est généralement installé sur une machine qui est serveur web. Il est étroitement lié au serveur HTTP (souvent Apache ou Nginx), et peut se connecter à des bases de données (par exemple client-serveur comme le SGBD MySQL).

L'utilisateur a juste besoin d'un client web pour appeler les pages PHP.

En phase de développement, on ne travaille jamais directement sur le serveur de déploiement : cela imposerait de devoir envoyer le script PHP sur le serveur pour pouvoir le tester. C'est pourquoi la plupart des développeurs PHP installent sur leur machine un environnement ***AMP** (Apache, MySQL, PHP), afin de pouvoir mettre au point leurs scripts plus rapidement. C'est alors la même machine qui fait client et serveur.

Il est à noter qu'il est assez facile de trouver des hébergements *AMP, mais il faut tout d'abord déterminer les besoins (voir section 1.3 en page 16).

Attention : il est important de vérifier, avant le début du développement, que les versions des divers logiciels utilisés en développement sont compatibles avec ceux disponibles sur le serveur d'hébergement (déploiement), en particulier si vous n'avez pas les droits d'administration complets sur ce dernier.

3.2.3.2 Logiciels et packagings

PHP évoluant sans cesse, et son packaging faisant de même, il est difficile de recommander l'une ou autre version d'installateur. Le cours en ligne contiendra toujours les instructions d'installation à jour pour les diverses plateformes. On peut cependant recommander les packagings **EasyPHP** (Microsoft Windows), ou **XAMPP** (Apple Mac OS X, Microsoft Windows. . .). Sous environnement GNU/Linux, PHP, Apache et tous les outils nécessaires sont aussi intégrés à la distribution et peuvent donc être installés par le système de packaging.

Il est recommandé de développer sur sa propre machine et d'automatiser le **déploiement** sur le serveur, par exemple via de l'intégration continue (**Continuous Integration**) d'une **forge** comme Gitlab.

Pour gérer facilement une base de données, l'application web **Adminer** est recommandée car elle supporte plusieurs SGBD, contrairement à **phpmyadmin**.

3.2.3.3 Editeur

Tout est question du degré d'intégration (IDE) désiré et de la plateforme de développement retenue.

Parmi les solutions libres multiplateformes, on peut recommander **GNU Emacs**, qui est très extensible, ou le **Eclipse**.

3.2.4 Syntaxe de PHP

Les principaux éléments de syntaxe PHP (décoration) sont :

;	séparateur d'instructions
{ ... }	délimitation de blocks par { et }
$	toutes les variables sont préfixées par le dollar
/* ... */	commentaires multi-lignes
// ou #	commentaires de fin de ligne

En PHP, les variables sont représentées par un signe dollar ($) suivi du nom de la variable. Un identificateur[2] doit commencer par une lettre ou un souligné (_), suivi de lettres, chiffres ou soulignés. PHP est sensible à la casse ($x est différent de $X).

3.2.5 Opérateurs

PHP dispose des opérateurs arithmétiques et logiques courants.

Il offre, en plus, des opérateurs pour manipuler les chaînes de caractères (affectation, comparaison et concaténation) et les tableaux (union et comparaisons).

Le tableau à la figure 3.1 en page 68 dresse une liste de la priorité des différents opérateurs. Les plus prioritaires sont en haut du tableau. Les opérateurs sur une même ligne ont une priorité équivalente et dans ce cas, leur association décide de l'ordre de leur évaluation.

3.2.6 Erreurs

PHP supporte un opérateur de contrôle d'erreur : c'est @. Lorsque cet opérateur est ajouté en préfixe d'une expression PHP, les messages d'erreur qui peuvent être générés par cette expression seront ignorés[3]. On peut aussi globalement modifier l'affichage sur le client web des erreurs via la fonction PHP error_reporting(), que l'on peut appeler par exemple ainsi : error_reporting(E_ALL & ~E_NOTICE);

Si l'option track_errors est activée, les messages d'erreurs générés par une expression seront sauvés dans la variable globale $php_errormsg. Cette variable sera écrasée à chaque erreur. Il faut alors la surveiller souvent pour pouvoir l'utiliser.

La version 5 de PHP dispose d'un mécanisme simple d'**exceptions**.

3.2.7 Structures de contrôle

Le PHP dispose des même structures de contrôle que le C, mais accepte une syntaxe supplémentaire (liée au fait que le code soit embarqué dans une page web) pour les utiliser, et propose une instruction de boucle facilitant le parcours des tableaux.

---

2. Sous la forme d'une **expression régulière**, un identificateur est exprimable comme : $[a-zA-Z_][a-zA-Z0-9_]*^, avec en plus la possibilité d'utiliser des symboles compris entre 0x7f et 0xff – ce qui n'est pas recommandé vu qu'ils varient suivant le jeu de caractères

3. Similaire au Makefile de la commande UNIX make

opérateurs	**associativité**	**remarques**
`new`	unaire	
`[`	droite	tableaux
`++, --`	unaire	incrémentation décrémentation
`!, ~, -, (int), (float), (string), (array), (object), @`	unaire	négations, conversion de types et `@` est l'opérateur de contrôle d'erreur, voir section 3.2.6 en page 67.
`*, /, %`	gauche	arithmétique
`+, -, .`	gauche	arithmétique et chaînes
`<<, >>`	gauche	bit à bit
`<, <=, >, >=`		comparaison
`==, !=, ===, !==`		comparaison (attention : = est l'affectation qui peut retourner son résultat. L'opérateur === a été introduit en PHP4 et vérifie aussi le type des objets, voir un exemple à la section 5.3.2 en page 109).
`&`	gauche	bit à bit et références
`^`	gauche	bit à bit
`|`	gauche	bit à bit
`&&`	gauche	logique
`||`	gauche	logique
`?, :`	gauche	opérateur ternaire (voir point 2 en page 69)
`=, +=, -=, *=, /=, .=, %=, &=, |=, ^=, <<=, >>=`	droite	affectation
`and`	gauche	logique
`xor`	gauche	logique
`or`	gauche	logique
`,`	gauche	plusieurs utilisations

L'**associativité** de gauche signifie que l'expression est évaluée de gauche à droite, l'associativité de droite, l'inverse.

FIGURE 3.1 – Priorité des opérateurs en PHP

3.2.7.1 Alternatives, branchements

1. la structure `if ... else`.
   Cette structure peut être utilisée de la même façon qu'en C :

   ```
   if (expression) {
      bloc1
   }
   else {
      bloc2
   }
   ```

   L'expression sera évaluée, puis convertie en booléen. Le `bloc1` sera exécuté si le résultat est VRAI, et le `bloc2` sinon.

   Lorsque plusieurs structures `if...else` se suivent, il est possible d'utiliser l'instruction `elseif`, qui a le même effet qu'un `else` suivi d'un `if`.
2. l'opérateur ternaire ?
   L'opérateur ? est intéressant lorsque l'on souhaite retourner une valeur différente en fonction de la vérité d'une expression booléenne. Par exemple, si l'on souhaite affecter `positif` ou `négatif` à une chaîne de caractères en fonction du signe de la variable nombre, on pourrait écrire :

   ```
   $signe = ($nombre < 0) ? "négatif" : "positif ou nul"
   ```

   au lieu de :

   ```
   if ($nombre < 0) $signe = "négatif";
   else $signe = "positif ou nul";
   ```
3. structure `switch...case`
   Cette structure permet de comparer *une* variable à *plusieurs* valeurs *discrètes* et d'effectuer un traitement particulier pour chaque valeur de cette variable.

   ```
   switch ($taille) {
      case "petit": contenance = 20; break;
      case "moyen": contenance = 30; break;
      case "grand": contenance = 50; break;
      default: contenance = 0;
   }
   ```

3.2.7.2 Boucles

1. boucle `while`.
   Elle se comporte de la même façon qu'en C : `while (expression) { bloc }`

   `expression` est évaluée, puis convertie en booléen (si nécessaire), puis le bloc est exécuté si expression est VRAIE. Le bloc d'instruction est donc supposé changer une partie de l'expression.

   Si l'expression est FAUSSE dès la première évaluation, le bloc ne sera jamais exécuté.
2. boucle `do...while`
   Identique au C : le bloc est exécuté, puis l'expression évaluée. Le bloc sera au moins exécuté une fois, même si l'expression est FAUSSE dès la première évaluation.
3. boucle `for`
   Identique au C :

```
for (initialisation; condition; modification) { bloc }
```

avec

(a) exécution de la clause d'initialisation
(b) évaluation de la condition (sortie de la boucle si elle est FAUSSE)
(c) exécution du bloc (si la condition est vraie)
(d) exécution de la clause de modification
(e) retour à la deuxième étape (évaluation de la condition)

4. boucle `foreach`
   Permet le parcours d'un **tableau** (simple ou associatif) :

```
foreach ($tableau as $element) { bloc }
```

   Exécute le bloc autant de fois que `$tableau` contient d'éléments. A chaque itération la variable `$element` prendra la valeur de l'élément suivant du tableau [4].
   Dans le cas d'un **tableau associatif**, on utilisera la forme suivante :

```
foreach ($tableau as $cle => $valeur) { bloc }
```

   qui produit le même résultat, mais, pour chaque itération, `$cle` prendra la valeur de l'indice de l'élément courant et `$valeur` son contenu. Cette forme est particulièrement intéressante lorsque l'indice n'est plus un entier, mais une chaîne de caractères (tableau associatif). Lorsqu'on utilise cette fonction sur un tableau normal, on obtient les indices numériques dans `$cle`.

5. rupture de séquence (applicables aux boucles et à `switch`)

   `break [n]` permet de sortir du bloc en cours d'exécution. Accepte un paramètre facultatif `n` indiquant le nombre de blocs imbriqués que l'on souhaite quitter, valant par défaut 1.

   `continue [n]` permet de passer directement à l'itération suivante de la boucle en cours. Le paramètre facultatif `n` a le même rôle que pour le `break`.

3.2.7.3 Syntaxe alternative pour les boucles

PHP propose une autre manière d'écrire un bloc, pour les fonctions de contrôle `if`, `while`, `for`, `foreach` et `switch`. Dans chaque cas, le principe est de remplacer l'accolade d'ouverture par deux points (`:`) et l'accolade de fermeture par respectivement `endif;` `endwhile;`, `endfor;` ou `endswitch`. Cette autre syntaxe fonctionne aussi avec le `else` et le `elseif`.

```
if ($a == 5):
   echo "a égal 5";
   echo "...";
elseif ($a == 6):
   echo "a égal 6";
else:
   echo "a n'est ni 5, ni 6";
endif;
```

4. c'est également le cas si le tableau est en fait un tableau associatif

3.2.7.4 Autres structures de contrôle

`return [$resultat]` termine une fonction en renvoyant, s'il est indiqué, le résultat `$resultat` au programme appelant.

`include` **et** `require` permettent d'inclure un fichier PHP. Ces instructions seront remplacées par le contenu du fichier mentionné, puis l'interprète exécutera ce nouveau contenu avant de poursuivre l'exécution du script. La différence entre ces 2 instructions se situe dans leur gestion des erreurs : en cas d'erreur, `include` provoque une alerte (ou warning, le script continue), alors que `require` provoque une erreur fatale (script interrompu, recommandé). Le fichier inclus doit contenir la balise ouvrante PHP <?php si l'on veut pouvoir écrire du code PHP. On peut *renoncer* à la balise fermante ?> et c'est même recommandé de le faire pour éviter des lignes vides en particulier si le fichier ne contient pas d'HTML.

Attention au fait que suivant la configuration de PHP (**php.ini**), une inclusion distante par un URL est possible.

Exemple d'inclusion :

Fichier `test.php`

```
<html>
   <head><title>Include</title></head>
   <body>
      <?php
         $i = 10;
         echo "Avant include: " . $i . "<br/>";
         $res = include("inc.php");
         echo "après: \$res = " . $res . "<br/>";
       ?>
   </body>
</html>
```

Fichier `inc.php`

```
<?php
   echo "dans include: \$i = " . $i . "<br/>";
   return $i + 1;
?>
```

Le choix des extensions pour les inclusions a une signification importante. En effet, si elles finissent par `.php`, le serveur web va les exécuter. Sinon, il va simplement en afficher la source, qui sera interprétée par le client web. Cela a des implications de sécurité expliquées à la section 5.3.10 en page 113.

3.2.8 Variables

3.2.8.1 Types en PHP

PHP ne nécessite pas de déclaration explicite du type d'une variable. Le type d'une variable est déterminé par le contexte d'utilisation : il est dynamique. si on assigne une chaîne de caractères à la variable `$var`, `$var` devient une chaîne de caractères. Si on assigne un nombre entier à

`$var` elle devient un entier. De même pour un tableau par exemple. Dans certains cas, il est utile de faire un forçage de type qui aboutira à une conversion.

L'évaluation d'expressions peut générer des conversions automatique. L'opérateur d'addition (+), par exemple, effectuera les conversions nécessaires pour garder une précision maximale du calcul : si un des opérandes est de type double, alors tous les opérandes sont évalués comme des variables de type double et le résultat est de type double. Sinon, tous les opérandes sont évalués comme des variables de type entier et le résultat sera du type entier.

Il est à noter que cela *ne change pas* le type des opérandes. Le seul changement est la manière dont les opérandes sont évalués.

PHP utilise les 8 types suivants :

— 4 types scalaires : booléen, entier, nombre à virgule flottante, chaîne de caractères.
— 2 types composés : tableau, objet.
— 2 types spéciaux : ressource, NULL

Exemples :

```
$reponse = 42;              # entier
$pi = 3.14159;              # flottant
$verbe = "lire";            # chaîne
$phrase = "J'aime $verbe";  # chaîne avec interpolation
$phrase = 'It cost $100';   # chaîne sans interpolation
$ceci = $phrase;            # affectation de variable
$E = $m * $c * $c;          # expression
```

Pour imposer un type à une variable, on peut utiliser les opérateurs de transtypage (ou **type-cast**) :

`(int)`, `(integer)`	type entier
`(bool)`, `(boolean)`	type booléen
`(double)`, `(float)`, `(real)`	type flottant
`(string)`	type chaîne de caractères
`(array)`	type tableau
`(object)`	type objet

On peut utiliser l'opérateur & (référence de ...) pour indiquer une référence à une variable :

```
$a = 3;
$b = &$a;
$b = 4;   /* $a vaut maintenant 4 */
```

Le nom d'une variable peut être exprimé par une autre variable :

```
$var = "hello";
$$var =" world";
echo "${$var}<br/>"; // affiche world<br/>
echo "$hello<br/>"; // affiche world<br/>
```

3.2.8.2 Déterminer le type d'une variable

Parce que PHP détermine le type des variables et les convertit automatiquement, une variable n'aura pas toujours le type désiré. PHP inclut des fonctions permettant de déterminer

tableaux associatif	contenu
$GLOBALS	contient une référence sur chaque variable qui est en fait disponible dans l'environnement d'exécution global. Les clés de ce tableau associatif sont les noms des variables globales.
$_SERVER	les variables fournies par le serveur web, ou bien directement liées à l'environnement d'exécution du script courant. Ses valeurs ne sont pas forcément disponibles suivant la plateforme et la manière de lancer l'interprète PHP.
$_GET	les paramètres fournis au script via la chaîne de requête URL (méthode GET).
$_POST	les paramètres fournis par le protocole HTTP en méthode POST.
$_COOKIE	les variables stockées côté client, 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.
$_SESSION	les variables qui sont en fait enregistrées dans la session attachée au script, stockée côté serveur et référencée par un cookie côté client.

Originellement, les paramètres de scripts (**get**, **post**) étaient mélangés aux variables du script – ce qui était un grave problème de sécurité. Aujourd'hui, PHP vient en général avec la configuration **register-globals** désactivée (dépréciée dès PHP 5.3) et ces paramètres sont à consulter dans les tableau associatifs `$_GET` ou `$_POST`.

FIGURE 3.2 – Variables prédéfinies en PHP

le type d'une variable : `var_dump`, `gettype`, `is_array`[5], `is_float`, `is_int`, `is_object` et `is_string`.

3.2.8.3 Variables prédéfinies (super-globales)

PHP fournit un jeu de tableaux associatif prédéfinis, contenant les variables du serveur (si possible), les variables d'environnement et celle contenant les paramètres de formulaire ou d'URL.

Ces nouveaux tableaux (voir figure 3.2 en page 73) sont un peu particuliers, car ils sont automatiquement disponibles dans tous les environnements d'exécution (scopes, **portée**, voir section 3.2.8.4 en page 73). Pour cette raison, ils sont dits **super-globaux** ou **auto-globaux**.

3.2.8.4 Portée des variables

Nous avons déjà vu qu'en général, les variables ne sont pas déclarées en PHP. La portée d'une variable définie hors d'une fonction est en générale globale : toutefois, les variables *utilisées* dans une fonction sont automatiquement de portée *locale*, écrasant durant l'exécution de la fonction toute valeur déjà définie, ce qui peut être étonnant au premier abord, voir figure 3.3 en page 74.

Notons que dans ce cas, avec une variable utilisée dans la fonction mais non initialisée dans la portée locale de fonction, une erreur de niveau `E_NOTICE` est levée et figurera probablement dans les journaux, voire dans l'HTML (voir section 3.2.6 en page 67).

5. il n'est pas trivial de savoir si c'est un tableau associatif ou un tableau normal.

```
$a = 3;

function toto() {
   // global $a;

   echo $a; # n'affiche rien si la ligne global manque
   echo $GLOBALS['a']; # fonctionne toujours
}

toto();
```

Figure 3.3 – Portée locale de fonctions

Il est toutefois possible de spécifier la portée désirée à l'aide du mot-clé **global** ou d'accéder à la variable via le tableau super-global `$GLOBALS` (voir section 3.2.8.3 en page 73).

Il est à noter que PHP ne dispose *pas* de portée bloc.

Comme en langage C, une variable statique (mot-clé **static**) garde sa valeur à la sortie de la fonction la contenant et peut donc être utilisée pour stocker un contexte local.

3.2.9 Constantes

En PHP, les constantes sont définies avec la fonction `define()`. Une fois qu'elle a été définie, une constante ne pourra jamais être modifiée, ni détruite. Le contenu d'une constante est de toujours type **scalaire**.

Les noms de constantes doivent être conformes aux règles de syntaxe PHP concernant les identificateurs, et ne doivent pas commencer par le caractère $.

Exemple :

```
define("CONSTANTE", "Bonjour!");
echo CONSTANTE; // affiche Bonjour!
```

3.2.9.1 Constantes magiques

PHP fournit des constantes particulières, dont la valeur dépend du contexte où elles sont utilisées (voir figure 3.4 en page 74).

nom	**description**
`__LINE__`	la ligne courante dans le fichier.
`__FILE__`	le nom de la fonction (dès PHP 4.3).
`__CLASS__`	le nom de la classe courante (dès PHP 4.3).
`__METHOD__`	le nom de la méthode courante (dès PHP 4.3).

Figure 3.4 – Constantes magiques en PHP

La fonction `defined()` permet de savoir si une constante existe [6] ou non ; `get_defined_constants()` permet de connaître la liste des constantes définies.

6. Pour une variable voire un élément de tableau, utiliser `isset()`.

3.2.10 Chaînes de caractères

3.2.10.1 Introduction

Les chaînes de caractères sont des séquences de caractères. En PHP, par défaut, chaque caractère est représenté par un octet.

Une chaîne de caractères en PHP pourra être délimitée par des guillemets simples ou doubles, ou encore avec la syntaxe **HERE document**, aussi appelée **heredoc**, syntaxe présentée ci-dessous : la chaîne délimitrice `EOD` peut être n'importe quel identificateur. Il est possible d'écrire `<<<EOD` comme `<<<'EOD'` : dans ce cas, les interpolations de variables ne seront pas effectuées au sein de la chaîne (voir chaîne à apostrophes dans la section suivante).

```
   $sql_query = <<<EOD
   SELECT description, SUM(valeur)
      FROM comptabilite
      WHERE (compte LIKE '20-%')
      ORDER BY id ASC
EOD; // aligner 1ère colonne
```

3.2.10.2 Interpolation (expansion)

Il y a une différence dans l'**interpolation** des chaînes apostrophées (guillemet simple, `'`) et entre guillemets (guillemets doubles, `"`) :

avec les apostrophes Généralement utilisé pour éviter l'interpolation ou pour facilement insérer des guillemets :
- pour afficher une apostrophe, il faut l'échapper avec un anti-slash (backslash, \).
- les variables ne sont pas remplacées par leurs valeurs (elles ne sont pas interpolées).

avec les guillemets Généralement utilisé pour permettre l'interpolation de variables ou pour facilement insérer des apostrophes :
- pour afficher un guillemet, il faut l'échapper avec un backslash.
- les variables sont interpolées.
- les séquences d'échappement suivantes seront interpolées comme indiqué :

séquence	**interpolation**
`\n`	nouvelle ligne (linefeed, LF, 0x0A (10) en ASCII, fin de ligne standard d'UNIX)
`\r`	retour à la ligne (carriage return, CR ou 0x0D (13) en ASCII, dans les environnements Microsoft, CR + LF dénote la fin de ligne)
`\t`	tabulation horizontale (HT, 0x09 (9) en ASCII)
`\\`	anti-slash (backslash \)
`\$`	caractère $
`\"`	guillemets (", doubles)

Exemples :

```
echo 'Ceci est une chaîne simple';
echo 'Vous pouvez inclure des nouvelles
lignes dans une chaîne,
comme ceci.';
```

```
// affiche: "I'll be back"
echo 'Arnold a coutume de dire: "I\'ll be back"';

// affiche: Etes-vous sûr de vouloir effacer le dossier C:\*.*?
echo 'Etes-vous sûr de vouloir effacer le dossier C:\*.*?';
echo "Etes-vous sûr de vouloir effacer le dossier C:\\*.*?";

// affiche: Les variable ne sont pas affichées $ici \n ni les
//          séquences spéciales
echo '$ici restera tel quel, comme \n';

$boisson = 'vin';

// affiche: Du vin, du pain et du fromage!
echo "Du $boisson, du pain et du fromage!";

// Incorrect: 's' peut faire partie du nom de variable, PHP
//            interpole alors $boissons qui n'est pas ce que
//            l'on attend
echo "Il a goûté plusieurs $boissons";

// Version correcte
echo "Il a goûté plusieurs ${boisson}s";

echo "C'est assez <strong id=\"ici\">pénible</strong> d'écrire du HTML
      entre guillemets: le mieux est de fermer le PHP ou d'utiliser ";
echo 'les <em id="la">apostrophes</a> PHP';
```

Attention : l'apostrophe inverse (**backtick**, `) est interpolée comme une exécution de commande UNIX ou MS-DOS – en dehors des chaînes de caractères.

3.2.10.3 Fonctions et opérateurs pratiques sur les chaînes

Les fonctions suivantes sont très courantes dans tout programme PHP :

- l'affectation d'une chaîne à une variable se fait avec l'opérateur =
- la comparaison de 2 chaînes peut se faire avec l'opérateur ==, mais il existe des fonctions plus appropriées, comme `strcmp()`
- les chaînes peuvent être concaténées avec l'opérateur .
- les octets[7] d'une chaîne sont accessibles et modifiables en spécifiant leur décalage (offset, le premier octet a l'offset 0) entre crochets, ou accolades[8], après le nom de la variable.

```
$str = 'Ceci est un test.';

// Lit le premier caractère de la chaîne
$first = $str[0];

// Lit le troisième caractère de la chaîne
```

7. consulter la mise en garde en page 77
8. interdit dès PHP 6

```
// (version incompatible dès PHP 6)
$third = $str{2};

// Lit le dernier caractère de la chaîne
$last = $str[strlen($str) - 1];

// Modifie le dernier caractère de la chaîne
$str[strlen($str) -1 ] = '!';
```

On trouvera une présentation exhaustive des fonctions relatives aux chaînes de caractères dans la documentation officielle de PHP [2].

Attention : PHP a été conçu en supposant des caractères de exactement un octet. Cela a évolué au fil du temps avec la norme Unicode et, donc, par exemple avec les chaînes de caractères multi-octets UTF-8, il faut éviter les fonctions `strlen()` et `substr()` et utiliser leurs alternatives `mb_strlen()` et `mb_substr()` (voir section 3.2.17.1 en page 88).

3.2.11 Tableaux et tableaux associatifs

3.2.11.1 Définitions

On distinguera le **tableau** classique (indexable par des entiers de 0 à la taille du tableau moins 1, comme en C ou Java) du tableau associatif, qui lui est indexé par un scalaire quelconque, la clé : une valeur, de n'importe quel type PHP y compris les tableaux et tableaux associatifs, est associée à chaque clé.

Une **association** est un type de données qui fait correspondre des valeurs à des clés. La structure de donnée PHP implémentant une telle association est le **tableau associatif**, à ne pas confondre avec le **tableau de hachage** (hash table)[9], qui est optimisé pour la recherche mais ne conserve pas l'ordre des éléments.

Un **tableau associatif** PHP est, en réalité, une **association ordonnée**.

Si, contrairement au tableau, le tableau associatif PHP n'est pas directement indexable par numéro d'index, mais par sa clé, il est possible, si nécessaire, de convertir un tableau associatif en simple tableau de valeurs avec la fonction `array_values()`.

3.2.11.2 Affectation

Un tableau ou tableau associatif peut être créé avec la fonction `array()` ou la syntaxe simplifiée avec crochets, et les valeurs (tableau) ou clés-valeurs (`key => value`, tableau associatif), séparées par des virgules.

Exemples :

```
$arr = array("foo" => "bar", 12 => true);
$arr = [ "foo" => "bar", 12 => true ]; // PHP >= 5.4
$tab = [ "prems", "deux", "trois" ];

echo $arr["foo"]; // bar
```

9. par exemple Perl propose le tableau de hachage : la sémantique est légèrement différente, notamment l'ordre n'est pas maintenu et l'unicité est garantie par un index, qui offre aussi un avantage de performance.

```
// un tableau associatif n'est pas indexable
// par la position des éléments (créerait une ambiguité):
// ici on indexe la clé qui a la valeur 12!
echo $arr[12]; // 1

echo $tab[1]; // deux
```

Lorsque la clé est omise lors de la spécification d'un tableau, l'indice maximum + 1 sera utilisé comme clé par défaut. Si aucun indice numérique n'a été généré, ce sera 0. Si la clé spécifiée a déjà été utilisée, la nouvelle valeur écrasera la précédente :

```
// Ces deux tableaux associatifs sont identiques
[ 5 => 43, 32, 56, "b" => 12 ];
[ 5 => 43, 6 => 32, 7 => 56, "b" => 12 ];
```

Inutile de dire que ce n'est pas une bonne pratique de faire ce genre de choses !

Une valeur peut être de n'importe quel type :

```
$arr = [ "untableau" => [ 6 => 5, 13 => 9, "a" => 42 ] ];

echo $arr["untableau"][6]; // 5
echo $arr["untableau"][13]; // 9
echo $arr["untableau"]["a"]; // 42
```

On peut aussi modifier un tableau existant en lui assignant simplement des valeurs. L'assignation de valeurs de tableau se fait en spécifiant la clé entre crochets : `$arr[ $key ] = $value;`

Si la clé est omise, la valeur sera ajoutée à la fin du tableau : `$arr[] = $value;`, ce qui donne des effets intéressants :

```
$a = [ "un" => "abcd", 12 => "toto", "bla" ];
$a[] = "truc";

// à partir d'ici, $a contient la même chose que $b:
$b = [ "un" => "abcd", 12 => "toto",
       13 => "bla", 14 => "truc"
     ];
```

Si $a n'existe pas, il sera créé. Cela en fait une alternative pour créer un tableau.

Pour supprimer une valeur, on dispose de la fonction `unset()`.

```
$arr = [ 5 => 1, 12 => 2 ];

$arr[] = 56; // Ceci revient à $arr[13] = 56;
$arr["x"] = 42; // Ceci ajoute un nouvel élément avec l'index "x"

unset($arr[5]); // Ceci efface un élément du tableau
unset($arr); // Ceci efface tout le tableau
```

3.2.11.3 Fonctions et opérateurs pratiques sur les tableaux

exemple	nom	résultat
`$a + $b`	union	union de $a et $b.
`$a == $b`	égalité	TRUE si `$a` et `$b` contiennent les mêmes paires clés / valeurs.
`$a === $b`	identique	TRUE si `$a` et `$b` contiennent les mêmes paires clés / valeurs dans le même ordre et du même type.
`$a != $b`	inégalité	TRUE si $a n'est pas égal à $b.
`$a <> $b`	inégalité	TRUE si $a n'est pas égal à $b.
`$a !== $b`	non-identique	TRUE si $a n'est pas identique à $b.

L'opérateur + ajoute le tableau de droite à celui de gauche. Les clés communes voient leurs valeurs affectées à celles du tableau de gauche.

Pour comparer des tableaux, PHP dispose par exemple d'`array_diff()`.

```
$a = [ "pomme", "banane" ];
$b = [ 1 => "banane", "0" => "pomme" ];

var_dump($a == $b); // bool(true)
var_dump($a === $b); // bool(false)
```

Pour afficher un tableau complexe, on peut aussi utiliser `print_r()`, par exemple ainsi :

```
<pre>
    <?php
        print_r($t);
    ?>
</pre>
```

Comme pour les chaînes de caractères, PHP fournit de nombreuses fonctions de traitements sur les tableaux. En voici quelques unes :

`count($tab)` retourne le nombre d'éléments du tableau.

`split($sep, $chaine)` **ou** `explode()` transforme une chaîne en tableau (`$sep` étant le séparateur).

`join($sep, $tab)` **ou** `implode()` transforme un tableau en chaîne de caractères (`$sep` étant le séparateur).

`in_array($val, $tab)` teste si `$val` est une valeur du tableau

`array_map($callback, $tab)` parcourt le tableau : chaque valeur du tableau est passée par valeur à la fonction `$callback` qui retourne un résultat : l'ensemble des résultats ordonnés forme un nouveau tableau retourné par la fonction `array_map()`

`array_splice()` supprime des éléments

`array_key_exists($key, $tab)` teste si `$key` est une clé du tableau

`array_keys($a)` retourne le tableau des clés d'un tableau associatif : voir aussi `array_values()`

L'instruction `foreach` (voir point 4 en page 70) est dédiée aux tableaux : elle permet de passer en revue les valeurs d'un tableau – et/ou les clés s'il s'agit d'un tableau associatif.

Il y a aussi quelques fonctions de parcours (`reset()`, `end()`...) plus ésotériques.

3.2.12 Fonctions

3.2.12.1 Fonctions natives

PHP intègre de nombreuses fonctions prédéfinies, toutefois certaines d'entre elles peuvent nécessiter des extensions de PHP (décrites au chapitre *Pré-requis* de la documentation de chacune des bibliothèques).

On pourra utiliser les fonctions `phpinfo()` ou `get_loaded_extensions()` pour savoir quelles extensions sont installées.

3.2.12.2 Définition de fonctions

Une fonction peut être définie en utilisant la syntaxe suivante :

```
function foo($arg_1, $arg_2, /* ..., */ $arg_n) {
   $retval = "demo";

   echo "Exemple de fonction.\n";

   return $retval;
}
```

Tout code PHP, correct syntaxiquement, peut apparaître dans une fonction.

Les valeurs sont retournées en utilisant le paramètre optionnel de l'instruction `return`. Tous les types de variables peuvent être renvoyés, tableaux et objets compris. `return` met un terme immédiat à l'exécution de la fonction et passe le contrôle à la ligne appelante. Contrairement au C, il est inutile de fournir le type de valeur renvoyée par `return` dans la définitions de fonction.

Il n'est pas possible de retourner plusieurs valeurs, mais on peut renvoyer un tableau.

Des informations peuvent être passées à une fonction en utilisant une liste d'arguments séparés par une virgule.

Par défaut, les arguments sont passés à la fonction par valeur, mais il est possible d'ajouter l'opérateur & devant l'argument dans la déclaration de la fonction :

```
function add_some_extra(&$string) {
   $string .= ', et un peu plus.';
}

$str = 'Ceci est une chaîne';

add_some_extra($str);

echo $str; // affiche 'Ceci est une chaîne, et un peu plus.'
```

On peut définir des valeurs par défaut pour les arguments de type scalaire :

```
function servir_cafe ($type = "cappuccino") {
   return "Servir un $type.\n";
}
```

```
echo servir_cafe(); // Servir un cappucino
echo servir_cafe("espresso"); // Servir un espresso
```

Rappelons que la portée des variables utilisées dans les fonctions est par défaut locale (sans déclaration avec le mot clé **global**) et remplace donc les variables définies ailleurs durant l'exécution de la fonction (voir section 3.2.8.4 en page 73).

Enfin, une fonction peut être définie dans une fonction, ce qui complique encore la notion de portée (les variables de la fonction englobante ne font, pas défaut, pas partie de la portée de la sous-fonction), voir un exemple à la section 4.3 en page 102.

3.2.12.3 Fonctions anonymes et closures

On peut créer une fonction anonyme très simplement :

```
$f = function($x) { return strlen($x); };
echo $f("demo"); // 4
```

Il est possible de définir des fonctions qui prennent avec elles partie du contexte d'appel (p.ex. variables définies dans la portée où elles ont été instanciées) : on les appelle des **closures** ou fermetures :

```
function demo($v) {
   return function($p) use ($v) {
      return $p . ": " . $v;
   };
}

$f = demo(42); // générons une fonction avec $v = 42

echo $f("vaut"); // vaut: 42
```

NB : pour modifier la variable `$v` dans la fonction anonyme ci-dessus, il suffirait d'écrire `use (&$v)`. Toutefois, ici, cela ne modifierait pas chez l'appelant de la fonction `demo()`, uniquement au sein de cette dernière fonction, car la variable n'y est pas passée par référence.

3.2.13 Namespaces

PHP permet de déclarer des variables, fonctions et classes dans des espaces de nommages séparés, par exemple pour résoudre des conflits ou pour améliorer la maintenabilité en structurant le code. Associé à des **autoloaders** pour charger automatiquement les classes nécessaires, cela transforme PHP en un langage très moderne.

3.2.14 Orientation objet

3.2.14.1 PHP comme langage orienté-objet

Jusqu'ici, nous avons vu PHP comme langage principalement procédural, avec un tout petit peu de paradigme fonctionnel avec les closures. S'il est vrai que PHP a été initialement conçu

comme langage procédural, au fur et à mesure des versions, le paradigme de programmation orientée-objet (POO) y a été ajouté.

L'intérêt principal de l'orienté-objet est l'encapsulation, sous forme d'objets, de données (attributs) ainsi que des fonctions qui servent à les manipuler. Ces objets sont en général liés à des concepts et interagissent avec d'autres objets. Comme le dit Wikipedia [10], la programmation orientée-objet en PHP permet de réaliser une architecture MVC (voir section 1.2.2 en page 16).

L'objectif de cette section n'est pas d'introduire la POO, on supposera que le lecteur a déjà des notions de base et pratiqué dans d'autres langages et on se concentrera sur les spécificités de PHP. Il ne s'agit d'ailleurs que d'un résumé, les slides du cours vont bien plus en profondeur.

3.2.14.2 Comparaison avec d'autres langages

Le fait que PHP n'ait pas été initialement orienté-objet se voit par exemple dans la nature procédurale de certaines fonctions de base du langage : par exemple, on écrit `strlen($v)` et pas `$v.strlen()`.

La déclaration de classes est similaire à d'autres langages, ainsi que la visibilité (`public` par défaut, `protected` ou `private`). On utilisera la notation double-pointée `::` pour accéder aux méthodes ou variables de classe (définies avec le mot-clé `static`). Dans le code d'une classe, l'objet-variable `$this` dénote l'objet courant, et le préfixe `self::` permet d'accéder une variable ou méthode de la même classe où le code exécuté se trouve, similairement `static::` pour la classe de l'objet en cours et `parent::` pour la classe parente. Les méthodes et variables d'objet sont appelées via la notation fléchée `->`.

PHP ne supporte que l'héritage simple, même si l'héritage multiple peut être émulé avec les **traits**.

Le constructeur et le destructeur sont définis à l'aide de méthodes magiques, respectivement `__construct()` et `__destruct`, plutôt que par le nom de classe comme dans d'autres langages. D'autres méthodes magiques permettent d'implémenter des getters et setters ou des méthodes dynamiques, de sérialiser les objets ou de représenter textuellement un objet (`__toString()`).

La surcharge de méthode n'existe *pas* [11] : il est possible de redéfinir une méthode dans une classe fille, tant que l'on ne change pas la signature (le nombre d'arguments) de la méthode – sauf pour le constructeur qui peut être complètement redéfini dans une classe fille, par exemple.

Enfin, des fonctions d'introspection permettent de déterminer le nom de la classe d'un objet (`get_class()`) ou les attributs de celui-ci (`get_object_vars()`), ce qui est pratique pour rendre le code plus générique.

3.2.15 Exemples courants du web

3.2.15.1 Utilisation de formulaires

Sur une page web, l'utilisation des formulaires permet d'exploiter des données entrées par l'utilisateur. Par exemple, les moteurs de recherche utilisent un formulaire pour permettre à l'utilisateur de saisir les mots recherchés et les envoyer au logiciel de recherche sur le serveur.

---

10. https://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_objet#Diff%C3%A9rents_usages_de_la_POO

11. on peut l'émuler soit à l'aide d'arguments variadiques et de paramètres par défaut dans les cas simples, ou via la méthode magique `__call()` dans le cas général.

En HTML, les formulaires peuvent notamment contenir les éléments suivants : champs texte (et mot de passe), boutons radio, cases à cocher, boutons, listes déroulantes et champs cachés. Chacun de ces éléments porte un nom via l'attribut `name` (voir section 2.1.19 en page 43) et on retrouvera, en PHP, dans le tableau associatif `$_GET` ou[12] `$_POST`, les valeurs des attributs `name` associés à la valeur par défaut ou entrés par l'utilisateur.

Au moment de l'envoi du formulaire (bouton `submit` ou touche ENTREE), les données (paramètres du formulaire) sont envoyées au script situé à l'URL indiquée par l'attribut `action` de l'élément `form` concerné. Ce script peut très bien être un script qui a généré auparavant le HTML contenant le formulaire (un simple test d'existence des paramètres requis est suffisant pour discriminer ces deux cas[13]).

L'un des points forts de PHP est sa capacité à gérer les formulaires. Le concept de base est que tous[14] les champs d'un formulaire seront automatiquement disponibles dans le script PHP d'action (sous forme de paramètres dans les tableaux associatifs ad-hoc, voir la section 3.2.8.3 en page 73).

Soit le formulaire HTML5 suivant :

```
<form action="action.php" method="post">
   Nom: <input type="text" name="username"><br>
   Email: <input type="text" name="email"><br>
   <input type="submit" name="submit" value="Envoi!">
</form>
```

Dans le script `action.php`, il suffira d'accéder `$_POST['username']` pour obtenir la valeur du champ `username` envoyée par le client.

Attention : dans ce qui précède, on n'a pas prêté attention ni à vérifier l'existence des paramètres, ni, s'ils sont affichés, à échapper leur contenu pour éviter des risques de sécurité (voir section 5.3.3 en page 109).

3.2.15.2 Images maps

Une implémentation simple d'images cliquables avec transmission des coordonnées peut se faire comme suit :

```
<input type="image" src="image.gif" name="sub">
```

Lorsque l'utilisateur clique sur cette image, le formulaire associé est envoyé au serveur, avec deux données supplémentaires, `sub_x` et `sub_y`[15]. Elles contiennent les coordonnées du clic de l'utilisateur dans l'image, qui peut ensuite être traité par le script.

On notera que ces variables sont envoyées par le navigateur avec un point dans leur nom, mais PHP convertit ces points en soulignés (le point étant interdit dans les identificateurs PHP).

Une autre façon de le faire qui déplace le traitement sur le client est via les éléments `map`.

---

12. selon l'attribut `method` spécifié

13. avec `isset()` – lorsqu'on utilise le modèle MVC, on créera une route pour afficher le formulaire, et une pour le traiter.

14. PHP exige cependant que les **paramètres multivalués** soient suivis des caractères `[]` à leur définition dans le formulaire, ce qui peut poser des problèmes de compatibilité. Ils seront alors disponibles comme un tableau dans les paramètres.

15. C'est PHP qui modifie automatiquement le nom du paramètre transmis en changeant le point – illégal dans un nom de variable PHP – en souligné. Voir section 2.1.19.9 en page 49.

3.2.15.3 Cookies

Les cookies sont un mécanisme permettant à un serveur de stocker une petite quantité d'informations sur le *client*. Une utilisation courante est d'y stocker une valeur non devinable qui référence plus de données du côté serveur (**session**, voir section 3.2.15.4 en page 85).

Ces informations sont enregistrées dans un fichier sur le client et contiennent les informations suivantes : Domaine/chemin, nom du cookie, valeur du cookie, date d'expiration. Un cookie est valable pour un certain temps et sera envoyé par le client lorsque le nom de domaine du serveur correspond, dans les entêtes HTTP.

En PHP, on peut créer un cookie grâce à la fonction `setcookie()` : les cookies font partie intégrante des entêtes HTTP et donc la fonction `setcookie()` *doit* être appelée avant que le moindre affichage ne soit envoyé au navigateur (la même restriction que pour la fonction `header()`, toujours car ces informations sont envoyées dans les entêtes HTTP *avant* les données).

Les données contenues dans les cookies sont alors disponibles en lecture seule dans le tableau (**auto-globaux**) de cookies `$_COOKIE`.

Pour assigner plusieurs valeurs à un seul cookie, il vous faut ajouter [] au nom du cookie (obligation PHP, similaire à l'obligation d'ajouter [] aux **paramètres multivalués**).

Par exemple : (sans sortie HTML avant !)

```
<?php
setcookie("MyCookie[0]", 'Test 1', time() + 3600);
setcookie("MyCookie[1]", 'Test 2', time() + 3600);
?>
```

Cela va créer deux cookies distincts bien que `MyCookie` soit maintenant un simple tableau du point de vue de PHP.

Il est à noter qu'un cookie remplace le cookie précédent par un cookie de même nom tant que le domaine/chemin est identique. Les règles de sécurité associées sont cependant relativement complexes.

Par exemple, pour une application de panier d'achat sans contexte côté serveur (rare), on peut utiliser un compteur et un panier stockés dans des cookies :

```
<?php
if (isset($_COOKIE['compte'])) {
   $compte = $_COOKIE['compte'] + 1;
}
else {
   $compte = 1;
}

setcookie('compte', $compte, time() + 3600);
setcookie("Panier[$compte]", $item, time()+3600);
?>
```

Attention : comme les cookies sont stockées – et modifiables – sur le client, il n'est pas possible d'y stocker des informations qui, si manipulées, changeraient fondamentalement la sécurité de l'application serveur. Dans l'exemple ci-dessous, il serait mieux de stocker, sur le client, une

valeur aléatoire de pas trop mauvaise qualité qui associe un contexte sur le serveur (voir la section suivante), qui peut faire l'objet d'expiration contrôlée.

Néanmoins, si l'on veut éviter à tout prix le contexte côté serveur – par exemple pour le **REST** – il est possible d'utiliser des cookies signés (**signed cookies**) : l'idée est que les informations stockées dans les cookies seront toujours lisibles [16] mais toute modification sera détectée grâce à une signature électronique [17], également stockée dans le cookie et vérifiée à chaque transaction [18]. Une valeur d'expiration peut être ajoutée aux données avant signature pour limiter l'impact d'un vol de cookie.

3.2.15.4 Sessions

On entend par session le suivi d'un utilisateur durant une suite d'opérations, avec ou sans authentification. Par exemple, la gestion d'un panier de commande peut être effectué par une session.

Les informations sur la session sont stockées sur le serveur (contrairement aux cookies qui sont stockés sur le client). Bien sûr, pour retrouver la session d'un utilisateur, il faut un identifiant permanent du côté client. Cet identifiant est garanti unique et cryptographiquement sûr [19].

Les variables sont stockées [20] côté serveur et la session elle-même est référencée par `PHPSESSID` (en général stocké comme cookie, du côté du client). Les variables sont accessibles en lecture et écriture dans le tableau associatif super-global `$_SESSION`.

Exemple qui crée une nouvelle session ou reprend la session courante :

```
<?php
session_start(); // une seule fois par script et avant toute sortie!
?>
<html>
   <head><title>Session demo</title></head>
   <body>
   <?php
   if (isset($_SESSION["compteur"]) {
      if ($_SESSION["compteur"] > 5) {
         // suppression de la variable dans la session
         unset($_SESSION["compteur"]);
      }
      else {
         echo "<p>le compteur: ", $_SESSION["compteur"], "</p>";
         $_SESSION["compteur"]++;
      }
   }
   else {
      echo "<p>1ère connexion</p";
      $_SESSION["compteur"] = 1;
   }
```

16. donc éviter les informations confidentielles !
17. la signature se fait avec un hachage cryptographique, par exemple SHA2, du contenu concaténé à une valeur secrète connue uniquement du serveur.
18. le client enverra l'ensemble des cookies correspondants à chaque requête, ce qui peut être lourd, toutefois.
19. il doit être difficile, pour un attaquant, de deviner un identificateur de session valide – des anciennes versions de PHP étaient vulnérables.
20. fichier indexé DBM, voire base de données

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

Par défaut, la session sera associée à un cookie et ceci aussi longtemps que le client web n'est pas relancé. Pour changer cela, modifier le paramètre `session.cookie_lifetime` de **php.ini**, ou appeler `session_set_cookie_params()`.

Pour supprimer une variable de session, on peut utiliser `unset($_SESSION['variable'])`. Pour détruire une session entièrement, la méthode recommandée aujourd'hui est de faire : `$_SESSION = []`. S'il y a besoin, on peut regénérer l'identificateur de session en effaçant le cookie associé.

Il est recommandé de vérifier que le mode `session.use_strict_mode` soit activé.

3.2.15.5 Envoi de fichiers

Il est facile de gérer le téléchargement (upload) de fichiers en HTTP avec PHP. Le premier élément nécessaire est un formulaire permettant au client de choisir un fichier local :

```
<form method="post" enctype="multipart/form-data" action="fupload.php">
    <!-- MAX_FILE_SIZE doit précéder le champ de type file -->
    <input type="hidden" name="MAX_FILE_SIZE" value="1024">
    <!-- Le nom du champ file détermine le nom dans le tableau $_FILES -->
    <input type="file" name="userfile">
    <input type="submit" value="Envoyer">
</form>
```

L'attribut `enctype` de la balise `form` doit contenir la valeur `multipart/form-data` comme type d'encodage **MIME**. Le bouton `<input type="file">` affiche un bouton permettant de parcourir les systèmes de fichiers du client. Le champ caché `MAX_FILE_SIZE` est une extension PHP (validée du côté serveur) et contient la taille maximale du fichier (en octets) et doit impérativement précéder le bouton `file`. Une limite maximum peut être configurée dans **php.ini** (variables `upload_max_filesize` et `post_max_size`) et c'est recommandé, vu que le champ caché est sous contrôle de l'utilisateur.

Le script appelé utilisera le tableau (auto-global) `$_FILES` pour exploiter ces différentes valeurs (`userfile` sera remplacé par le nom donné au bouton `<input type="file">` :

`$_FILES['userfile']['name']`	le nom original du fichier, tel que sur la machine du client web.
`$_FILES['userfile']['type']`	le type MIME du fichier, s'il a été fourni par le navigateur.
`$_FILES['userfile']['size']`	la taille, en octets, du fichier téléchargé.
`$_FILES['userfile']['tmp_name']`	le nom temporaire du fichier qui sera chargé sur le serveur.
`$_FILES['userfile']['error']`	le code d'erreur associé au téléchargement de fichier.

Le fichier téléchargé sera stocké temporairement dans le dossier temporaire du système (qui devrait disposer d'assez de place). Il faudra ensuite le déplacer vers le dossier destination [21].

Le script suivant `fupload.php` va traiter un fichier venant du formulaire ci-dessus.

21. de préférence hors de l'arborescence du serveur web, si l'extension `.php` n'est pas refusée ou renommée

```
<?php
$uploaddir = '/tmp/uploads/'; // à créer
$uploadfile = $uploaddir . basename($_FILES['userfile']['name']);

if (move_uploaded_file($_FILES['userfile']['tmp_name'],
                       $uploadfile)) {
   // fichier valide et déplacé avec succès
}

echo '<pre>';
print_r($_FILES);
echo '</pre>';}
?>
```

Si aucun fichier n'est sélectionné dans le formulaire, PHP retournera 0 dans `$_FILES['userfile']['size']` et rien du tout dans `$_FILES['userfile']['tmp_name']`

Le fichier temporaire sera automatiquement effacé à la fin du script, s'il n'a pas été déplacé ou renommé.

3.2.15.6 Manipulation d'images

3.2.15.6.1 Introduction Parmi les nombreux modules d'extension disponibles pour PHP, nous allons ici nous concentrer sur le traitement d'image. Même si les clients web modernes permettent une interprétation directe de formats vectoriels comme **SVG**, il peut toujours être utile de générer ou manipuler des images bitmap (dans différents formats) du côté serveur. Ces manipulations se font via la bibliothèque multi-language **GD graphics library**.

3.2.15.6.2 Fonctions principales

`imagecreatetruecolor(int xsize, int ysize)` retourne une ressource représentant une image noire de largeur `xsize` et de hauteur `ysize`.

`imagecolorallocate(resource image, int red, int green, int blue)` retourne un identifiant de couleur, représentant la couleur composée avec les couleurs fondamentales RGB (red, green, blue). L'argument `image` est le résultat d'une des fonctions `imagecreate()`. Les paramètres `red`, `green` et `blue` sont les valeurs respectives des composantes rouge, vert et bleue de la couleur désirée, sous forme d'entier entre 0 et 255 (hexadécimal : 0 et 0xFF). `imagecolorallocate()` doit être appelée pour créer chaque couleur représentée par image.

`imagepng(resource image[, string filename])` envoie l'image GD au format PNG sur la **sortie standard** (typiquement en PHP, le client web), ou, si `filename` est fourni, dans le fichier indiqué.

`imagecreatefrompng(string filename)` retourne un identifiant d'image représentant l'image créée à partir du fichier indiqué. En cas d'échec, la chaîne vide est retournée et un message d'erreur est envoyé au client web.

`imagecopyresized( resource dstimage, resource srcimage, int dstx, int dsty, int srcx, int srcy, int dstw, int dsth, int srcw, int srch)` copie une partie rectangulaire d'une image dans une autre image. `dstimage` est l'image destination, `srcimage` est l'image source. Si les dimensions des deux images ne correspondent pas, un étirement adéquat est effectué. Les coordonnées sont définies par rapport au coin

supérieur gauche. Cette fonction peut être utilisée pour recopier des régions à l'intérieur d'une même image, mais si les régions se chevauchent, le résultat pourra être incohérent.

Exemple

```
<?php
   // http://www.php.net/manual/fr/ref.image.php

   // Envoi d'un entête au client web pour lui indiquer qu'il s'agit
   // d'une image
   header("Content-type: image/png");

   // Création d'une image 200x100 (largeur x hauteur) (rectangle noir)
   // @ permet d'éviter d'envoyer d'éventuelles erreurs au client web
   $im = @imagecreatetruecolor(200, 100) or die("Erreur création image GD");

   // Allocation d'une couleur
   $text_colour = imagecolorallocate($im, 255, 128, 128);

   // Ajout d'un texte dans l'image, en commençant à (10, 40) du haut
   // dans la police 4 (les polices 1 à 5 sont prédéfinies)
   imagestring($img, 4, 10, 40, "Ceci est un test", $text_colour);

   // Transparence comme dernier paramètre (0: opaque à 127: transparent)
   $italic_text_colour = imagecolorallocatealpha($im, 128, 128, 255, 64);

   // Ajout du texte avec une police particulière (true-type: TTF)
   // les paramètres sont l'image, la taille, l'angle, et la position
   imagettftext($im, 20, 30, 40, 80, $italic_text_colour, "arial.ttf", "Ici");

   // Envoi du PNG au client web
   $imagepng($im);

   // Libération de l'espace mémoire GD
   imagedestroy($im);
?>
```

3.2.16 Identificateurs réservés

Des identificateurs sont réservés [22] par PHP : fonctions prédéfinies, constantes, etc. Vous ne pourrez pas les utiliser pour vos fonctions ou constantes. Il sera toutefois possible de les redéfinir avec les **namespaces** (voir section 3.2.13 en page 81), au risque de créer de la confusion.

3.2.17 Internationalisation et régionalisation

3.2.17.1 Internationalisation

Les applications web modernes ne peuvent plus se limiter au jeu de caractères classique régional ISO-8859-1, ni ses extensions comme ISO-8859-15 ou d'autres jeux régionaux. Idéalement, toute

22. https://www.php.net/manual/fr/reserved.php

la chaîne de traitement (du navigateur à la base de données) devrait aujourd'hui supporter l'Unicode et le jeu de caractères UTF-8.

En pratique, on doit faire des concessions : par exemple le support UTF-8 de la plateforme Microsoft Windows n'est pas encore idéal. Toutefois, dès la version 5.6, PHP utilise par défaut l'UTF-8.

Dans tous les cas, il faut vérifier quels jeux emploient en interne :
- l'interprète PHP : p.ex. configuration dans **php.ini**, variable `default_charset`
- la base de données : ne pas confondre entre le jeu natif de stockage de la table et le jeu éventuel de conversion pour la transaction vers le client, qui peuvent différer
- le serveur web (entêtes de jeu de caractères du protocole HTTP)
- la page web (entête éventuel `meta`, variant suivant les versions du langage HTML)

Mentionnons également que les fonctions qui traitent des chaînes de caractères peuvent ou non supporter les chaînes multi-octets de l'UTF-8. Comme illustration, la fonction `strlen()` retournera le nombre d'octets (et non pas le nombre de caractères) de la chaîne : c'est la fonction `mb_strlen()` qui comptera le nombre de caractères juste, dans la mesure où la chaîne est dans le jeu utilisé par défaut par PHP, ou si le jeu réel est précisé en argument de la fonction. Voir aussi p.ex. la fonction `mb_substr()` qui remplace `substr()`.

Le problème est aussi valable pour la fameuse fonction d'échappement `htmlentities()` (voir section 5.3.3 en page 109). Si la chaîne de traitement n'est pas cohérente, on peut, pour tester, spécifier le jeu de caractères de la chaîne à échapper en plus du standard HTML utilisé :

```
$s = htmlentities($s, ENT_XHTML, "ISO-8859-1");
```

3.2.17.2 Régionalisation

La régionalisation consiste en la traduction du logiciel, ce qui est facilité par des moyens techniques, comme par exemple l'utilisation explicite de la fonction `gettext()` dans le code PHP et le format standard des fichiers de langue **PO**. Des logiciels supportant ce standard existent pour traduire facilement n'importe quel logiciel. Il existe même des outils pour rendre traduisible un logiciel qui n'a pas été développé pour, en remplaçant automatiquement toutes les chaînes par des appels à `gettext()`.

Chapitre 4

Interfaçage de bases de données (SGBD)

Sommaire

4.1 Principes de base **91**
4.1.1 Interfaçage, sécurité et ORM 91
4.1.2 Complexité dans l'application et/ou dans la base de données ? . . . 92
4.1.3 Types de base de données particulières 92
4.2 Interfaçage entre PHP et MySQL **93**
4.2.1 Accès à une base MySQL 93
4.2.2 Un exemple avec PHP, PDO et MySQL 94
4.2.3 Exemples avec d'autres API 98
4.3 Assurer la sécurité, la performance et éviter le sur-codage **102**
4.4 Recommandations **103**
4.5 Vers les frameworks **104**

Le but de ce chapitre est de donner quelques informations sur l'interfaçage de bases de données avec des applications web, en donnant également quelques informations sur les types de bases de données appropriées. Nous n'irons pas dans les détails du fonctionnement et des principes des SGBD [1].

4.1 Principes de base

4.1.1 Interfaçage, sécurité et ORM

Le mode le plus simple d'interfaçage entre l'application et la base de données est basé sur des échanges de textes en langage SQL : on mélange code SQL des opérations **CRUD** (insertion, modification, suppression, lecture) et données littérales : il s'agit d'un cas particulier de **signalisation en-bande** (voir le chapitre traitant de la sécurité et en particulier la section 5.3.6 en page 111) et donc de l'échappement, spécifique à la base de données concernée, est nécessaire, pour un interfaçage correct et sûr, pour éviter notamment des attaques d'**injection**.

1. voir le cours de base de données de 2e année.

L'API d'interfaçage était, avec PHP, originellement spécifique à chaque logiciel de base de données : heureusement aujourd'hui il est possible d'utiliser une couche d'abstraction qui manipule ensuite des *backends* ou pilotes spécifiques, grâce à **PDO** (*PHP Data Objects*), qui peut de plus manipuler de véritables objets PHP – y compris un début d'*Object Relational Mapping* (**ORM**) – que l'on peut exploiter encore mieux grâce des frameworks.

4.1.2 Complexité dans l'application et/ou dans la base de données ?

Même si beaucoup de développeurs web les ignorent et préfèrent coder un maximum dans leur langage de script côté serveur préféré, jusqu'au format des données elles-mêmes, il ne faudrait pas complètement négliger les opérations [2] et fonctionnalités [3] complexes qu'une base de données peut offrir.

Le fait que beaucoup de développeurs web ignorent ces fonctionnalités peut s'expliquer par le fait qu'initialement, MySQL ne supportait quasi rien (**InnoDB** permet de passer outre les limitations du *backend* **MyISAM**), et que bien souvent on peut n'avoir besoin que d'une partie de ces fonctions, voire même les implémenter dans une couche transactionnelle logicielle.

Dans le cadre du cours, on proposera une approche adaptée, utilisant la complexité des bases de données lorsque c'est nécessaire (contraintes d'intégrité, y compris référentielles, atomicité, requêtes complexes parfois), partant des structures de données vers l'application [4].

Pour donner un exemple simple : on voit souvent du code qui teste l'existence d'un tuple dans la base de données, et s'il n'existe pas, ajoute ce tuple : il est pourtant bien plus simple d'insérer, et de traiter, si elle arrive, l'erreur de duplicat (par exemple sous forme d'exception, voir section 4.2.2.9 en page 97). Avantage : pas de fenêtre de vulnérabilité et code plus simple pour le cas normal.

4.1.3 Types de base de données particulières

La plupart du temps, l'application web côté serveur accédera à une base de données client-serveur typique comme **MySQL** ou **PostgreSQL**, via TCP, sur le même serveur ou sur un serveur dédié (voir section 1.2.1 en page 15).

Toutefois, certaines applications web sont mobiles ou embarquées. Dans ce cas, la base de données peut être distante et accédée par **Web services**, ou alors locale : il existe plusieurs API HTML5 accessibles en Javascript, et en applications mobiles natives ou en embarqué, il y a souvent la possibilité d'utiliser directement **SQLite** [5], ou sous forme de fichiers (indexés DBM, plat CSV...).

Certains types de données sont mieux traités dans des bases de données appropriées non relationnelles (p.ex. pour le XML **eXist**, ou pour des données très volumineuses du **big data**, des bases de données **NoSQL**).

---

2. notamment les recherches aggrégées complexes, évitant de nombreux allers-retours entre le **SGBD** et l'application

3. l'atomicité, la cohérence, l'intégrité et la durabilité (**ACID**), les **contraintes d'intégrité**, les **vues**, **schémas**, **règles**, **triggers** et les **procédures stockées**, voire même les **transactions**

4. une autre approche part de l'application, et notamment son modèle objet, pour générer et mettre à jour à l'aide d'un framework comme *Laravel Migrations*, les structures de données, ceci fait l'objet du cours d'approfondissement concernant les frameworks (3e).

5. SGBD fichier à interface SQL

4.2 Interfaçage entre PHP et MySQL

4.2.1 Accès à une base MySQL

4.2.1.1 Pour l'administration et la maintenance

Il existe plusieurs méthodes pour accéder à une base de données MySQL pour faire de la maintenance (créer des bases, créer des tables, altérer le schéma, ...)

1. on peut se connecter au SGBD en ligne de commande, ce qui est très pratique pour créer automatiquement une base depuis un simple fichier texte (développement, test ou déploiement)
2. on peut utiliser un outil complexe de gestion, par exemple implémenté en PHP comme **Adminer**
3. on peut utiliser un connecteur **ODBC** pour interfacer une application comme un tableur
4. on peut écrire nos propres scripts, p.ex. en PHP, qui interagiront avec des bases MySQL via différentes abstractions (fonctions, interface orientée objet, **ORM** *Object-Relational Mapping*, ...) – voir la section suivante

Le développeur est amené à faire un choix entre ces diverses méthodes, suivant le niveau d'*automatisation* du déploiement souhaité. La plupart du temps et en particulier lors de prototypage, Adminer est un excellent outil ne nécessitant pas de connaissances particulières de MySQL : il est toujours possible de sauvegarder les structures créées avec Adminer en format *dump* texte MySQL, de les stocker dans un contrôle de version, puis les recharger automatiquement à l'aide de l'outil `mysql`. Parfois, l'on voudra utiliser des frameworks de haut niveau pour la création et la maintenance des données (p.ex. *Laravel Migrations*) : à contrario, il est en général mauvais de traiter la création d'une base de données depuis des scripts PHP accessibles par web sans précautions.

4.2.1.2 Pour l'accès aux données par l'application

Le principe est d'écrire des scripts PHP qui se connecteront à MySQL et qui pourront envoyer des requêtes SQL aux différentes bases présentes, traitant des différentes tables (ou relations) créées.

L'application a plusieurs choix d'API pour échanger avec une base de données MySQL. La différence entre ces choix d'API réside principalement dans leur interface (impérative ou orientée objet), mais aussi dans leur niveau d'abstraction (faut-il changer le code si l'on change de base de données), les fonctionnalités de la base de données qui sont accessibles au développeur, une correspondance automatique relationnel-objet (**ORM**), et des fonctions de sécurisation.

Dans le cas de MySQL, les choix les plus classiques sont les suivants :

API	**indépendance de la BD**	**fonctionnalités avancées de la BD**	**OO**	**ORM**	**sécurité**
mysql (obsolète)	NON	NON	NON	NON	échappement manuel
mysqli	NON	OUI (*)	à choix	NON	**binding** ou échappement manuel
PDO	OUI (pilotes)	OUI	OUI	OUI	binding

Par rapport à l'API mysql (obsolète), mysqli ajoute le choix du paradigme orienté-objet (OO) et la possibilité de choisir le connecteur bas-niveau : entre la bibliothèque classique `libmysqlclient` et le pilote MySQL moderne natif (qui supporte certaines (*) fonctionnalités SGBD de MySQL, mais pas toutes d'où l'étoile dans le tableau ci-dessus).

Nous présenterons dans les sections suivantes principalement l'interfaçage de MySQL depuis vos applications web en PHP avec l'API **PDO** – c'est l'interfaçage le plus moderne. Il se peut toutefois que vous deviez un jour maintenir des applications utilisant les anciennes API : quelques exemples seront également donnés avec des API plus anciennes (voir section 4.2.3 en page 98).

4.2.2 Un exemple avec PHP, PDO et MySQL

4.2.2.1 Introduction

On se propose d'accéder à une base sur le serveur MySQL de la machine locale, contenant au moins une base de données `test`, elle-même contenant une table, appelée `user` et comportant les champs `login`, `password` et `email`, avec une contrainte `UNIQUE` sur le `login`.

4.2.2.2 Création de la BD

Créer un fichier texte contenant la structure de la base de données peut être utile, par exemple :

```
CREATE TABLE user (
   id INT AUTO_INCREMENT NOT NULL,
   login VARCHAR(16) NOT NULL,
   password VARCHAR(16) NOT NULL,
   email VARCHAR(200) NOT NULL,
   PRIMARY KEY(id),
   UNIQUE(login));
```

A ce stade, quelques commentaires :

- il n'est pas forcément nécessaire de contraindre les champs aussi précisément dans leur longueur : un champ `TEXT` peut parfaitement convenir – attention, en MySQL, il y a des restrictions sur les opérations possibles sur les champs TEXT.
- la clé primaire est ici le champ automatiquement incrémenté MySQL (`id`), une clé primaire est toujours unique – d'autres SGBD utilisent la notion de séquence
- de manière à garantir l'unicité du login, on a ajouté la contrainte `UNIQUE(login)` – probablement via un **index**
- les contraintes d'intégrité `NOT NULL` permettent d'assurer que les champs doivent contenir quelque chose (y compris une chaîne vide ici) : de meilleures contraintes peuvent être créées en fonction des besoins.

Ce fichier peut alors être géré dans un **contrôle de version** et réutilisé pour les tests ou le déploiement via l'outil `mysql` :

```
$ mysql -u root # -p pour demander le mot de passe
mysql> CREATE DATABASE test;
mysql> USE test # nécessite un ; si exécuté dans un fichier
mysql> source fichier
mysql> exit
```

Une autre façon plus simple au début est simplement d'utiliser l'outil interactif web Adminer, par exemple, en envoyant la requête suivante :

4.2.2.3 Se connecter à la base de données

On se connecte à la base de données MySQL se trouvant sur la machine locale, port 3306, nom de base de données `test`, avec l'utilisateur `root`[6], sans mot de passe (développement local) :

```
$db = new PDO("mysql:host=127.0.0.1;port=3306;dbname=test",
               "root", // utilisateur
               "", // mdp souvent vide en développement
               [
                  PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                  PDO::MYSQL_ATTR_INIT_COMMAND
                     => 'SET NAMES utf8mb4 COLLATE utf8mb4_unicode_ci',
                  PDO::ATTR_PERSISTENT => true
               ]);
```

On voit ici que plusieurs options sont passées au constructeur PDO : il s'agit ici d'activer les exceptions, d'éviter toute conversion de jeu de caractères entre MySQL et PHP (en forçant l'UTF-8 avec ordre de tri précisé) et de rendre les connexions persistantes pour la performance.

Maintenant nous sommes connectés, nous avons sélectionné une base particulière, et nous pouvons y envoyer des requêtes SQL : commençons par insérer un enregistrement, via une requête préparée exécutée ensuite.

4.2.2.4 Requête préparée

Nous voulons insérer un enregistrement : les requêtes préparées améliorent la performance et permettent de facilement utiliser le **binding**[7], qui est ici positionnel[8] avec le ? :

```
$st = $db->prepare(<<<EOT
INSERT
   INTO
      user(login, password, email)
   VALUES
      ('maurice', 'test', ?)
EOT);
```

Il faudra ensuite exécuter la/les requête(s) concrètes (voir ci-après) avec les paramètres positionnels.

6. ceci est l'utilisateur `root` de MySQL et non pas d'UNIX, fort heureusement : mais il est toujours recommandé – et parfois obligatoire en hébergement – de travailler sous un ou plusieurs utilisateurs spécifiques : voir la section 5.3.4 en page 110.

7. sans binding, il faudrait échapper manuellement les données : le problème est traité en détail dans le chapitre 5 en page 105

8. on peut aussi nommer les variables liées, avec typage possible (sans grands effets pour le moment, avec la méthode `bindParam()` plutôt que de passer des paramètres à l'`execute()`).

4.2.2.5 Exécution d'une requête préparée

Il s'agit ici d'une insertion, donc on va traiter l'exception de duplicat (voir section 4.2.2.9 en page 97) et propager sinon :

```
try {
   $st->execute([$email]); // binding positionnel simple par tableau
   $id = $db->lastInsertId();
}
catch (PDOException $e) {
   // https://www.php.net/manual/fr/exception.getcode.php
   if ($e->getCode() == 23000) {
      // traiter l'erreur de duplicat
      echo htmlentities($e->getMessage());
   }
   else {
      throw $e; // ne nous concerne pas, relancer
   }
}
```

Notons aussi ci-dessus comment on obtient, avec MySQL [9], la clé primaire du tuple inséré en dernier.

4.2.2.6 Exécution directe d'une requête

On peut aussi exécuter directement une requête :

```
$st = $db->query("SELECT login, email FROM user");
```

4.2.2.7 Configurer le retour des résultats

S'il y a des résultats (après un SELECT), on peut obtenir les tuples (lignes) retournés sous diverses formes, ici un tableau associatif [10] :

```
# assurer retourne uniquement association clé => valeur
$result = $st->setFetchMode(PDO::FETCH_ASSOC);
```

4.2.2.8 Parcourir les résultats

On bouclera pour lire les résultats un tuple à la fois :

```
while ($row = $st->fetch()) {
   echo "login: ", htmlentities($row['login']),
        " email: ", htmlentities($row['email']);
}
```

Il peut être utile de détecter des cas particuliers (p.ex. aucun résultat) et d'en informer proprement l'utilisateur. Il y a d'autres méthodes pour itérer sur les résultats (par exemple les curseurs). L'affichage en HTML des données nécessitera de l'échappement (voir section 5.3.3 en page 109).

9. les séquences PostgreSQL fonctionnent différemment
10. alternatives : tableaux (positionnels), tableaux d'objets instanciés à partir du nom d'une classe.

4.2.2.9 Traitement d'erreurs

Si l'on a, à l'ouverture, activé l'option d'exceptions, on pourra traiter les exceptions attendues (exemple : insertion d'un duplicat, voir section 4.1.2 en page 92 et ci-dessus). Les exceptions non attendues seront non traitées ou relancées (en mode développement), et donc affichées dans le navigateur, ou traitées par le programme principal en production (jolie page d'erreur présentée, fichier de log alimenté).

4.2.2.10 Fermeture

En fait, la fermeture n'est pas nécessaire, et avec l'option de persistance, la base de données n'est ouverte qu'une fois.

4.2.2.11 Exemple complet

```
<?php
// configuration (peut être séparée, facilite le déploiement)
$config = [ 'connection' => 'mysql:host=127.0.0.1',
            'port' => '3306',
            'dbname' => 'test',
            'username' => 'root', // ou mieux
            'pw' => '', // vide en développement
            'options'
                => [
                     PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                     PDO::MYSQL_ATTR_INIT_COMMAND
                       => 'SET NAMES utf8mb4 COLLATE utf8mb4_unicode_ci',
                     PDO::ATTR_PERSISTENT => true
                   ]
          ];

// données provenant de sources incertaines
$email = "maurice@he-arc.ch";

// En PDO
// (avec concept de binding)

$db = new PDO($config['connection']
               . ';port=' . $config['port']
               . ';dbname=' .$config['dbname'],
               $config['username'],
               $config['pw'],
               $config['options']);

// ? représente un champ qui sera lié (binding) positionnellement
$st = $db->prepare(<<<EOT
INSERT
   INTO
      user(login, password, email)
   VALUES
```

```
      ('maurice', 'test', ?)
EOT);
   try {
      $st->execute([$email]); // binding positionnel simple
      $id = $db->lastInsertId();
   }
   catch (PDOException $e) {
      // https://www.php.net/manual/fr/exception.getcode.php
      if ($e->getCode() == 23000) {
         // traiter l'erreur de duplicat
         echo htmlentities($e->getMessage());
      }
      else {
         throw $e; // ne nous concerne pas, relancer
      }
   }

   $st = $db->query("SELECT login, email FROM user");

   # assurer retourne uniquement association clé => valeur
   $result = $st->setFetchMode(PDO::FETCH_ASSOC);

   while ($row = $st->fetch()) {
      echo "login: ", htmlentities($row['login']),
           " email: ", htmlentities($row['email']);
   }

   if (isset($id) && ctype_digit($id)) {
      // sans binding car validé
      $db->exec("DELETE FROM user WHERE id = $id"); // err. ign.
   }
?>
```

Noter ci-dessus le traitement d'exception par détection de duplicat à l'insertion (utilisant l'atomicité du SGBD) et les échappements HTML (via `htmlentities()`) pour ce qui est affiché dans le navigateur.

4.2.3 Exemples avec d'autres API

Ces API ne devraient plus être utilisées dans du nouveau code :

```
<?php
// configuration (peut être séparée, facilite le déploiement)
$host = "localhost";
$dbname = "test";
$user = "root"; // ou mieux
$pw = ""; // à changer

// données provenant de sources incertaines
$email = "maurice@he-arc.ch";
```

```
// ----------------------------------------------
// En mysql (obsolète, mais encore rencontré)
// (ici sans usage implicite du $db)

if ($db = mysql_connect($host, $user, $pw)) {
   if (mysql_select_db($dbname, $db)) {
      $sql_query = "INSERT INTO user(login, password, email) VALUES('maurice', "
                   . "'test', '"
                   . mysql_real_escape_string($email, $db) // échappement manuel
                   . "')";

      if ($result = mysql_query($sql_query, $db)) {
         $id = mysql_insert_id($db);

         echo "utilisateur ajouté";

         $sql_query = "SELECT login, email FROM user";

         if ($result = mysql_query($sql_query, $db)) {
            while ($row = mysql_fetch_array($result)) {
               echo "login: ", htmlentities($row["login"]),
                    " email: ", htmlentities($row["email"]);
            }

            // échappement de $i inefficace car pas entre apostrophes;
            // toutefois inutile car mysql_insert_id() ci-dessus
            // retourne un entier; sinon il faudrait *valider* que $id
            // est bien un entier, par exemple avec ctype_digit().
            $sql_query = "DELETE FROM user WHERE (id = " . $id . ")";
            mysql_query($sql_query, $db); // err. ign.
         }
         else {
            echo "erreur à traiter";
         }
      }
      else {
         echo "le login existe déjà ou autre erreur à traiter!";
      }
   }
   else {
      echo "erreur selection: ", htmlentities(mysql_error($db));
   }

   mysql_close($db);
}
else {
   echo "erreur connexion mysql: ", htmlentities(mysql_error());
}

// -----------------------------------------------------
// En mysqli avec du traitement de texte, mode impératif

if (($db = mysqli_connect($host, $user, $pw, $dbname))
    && !mysqli_connect_errno()) {
```

```
   $sql_query = "INSERT INTO user(login, password, email) VALUES('maurice', "
                . "'test', '"
                . mysqli_real_escape_string($db, $email) // éch. manuel
                . "')";

   if ($result = mysqli_query($db, $sql_query)) {
      $id = mysqli_insert_id($db);

      echo "utilisateur ajouté";

      $sql_query = "SELECT login, email FROM user";

      if ($result = mysqli_query($db, $sql_query)) {
         # sans MYSQLI_ASSOC, retourne association numérique ET nommée
         while ($row = mysqli_fetch_array($result, MYSQLI_ASSOC)) {
            echo "login: ", htmlentities($row["login"]),
                 " email: ", htmlentities($row["email"]);
         }

         // il faudrait valider que $id est un entier (p.ex. avec
         // ctype_digit(), car ici un échappement ne fonctionnera pas)
         $sql_query = "DELETE FROM user WHERE (id = " . $id . ")";
         mysqli_query($db, $sql_query); // err. ign.
      }
      else {
         echo "erreur à traiter", htmlentities(mysqli_error($db));
      }
   }
   else {
      echo "le login existe déjà ou autre erreur à traiter!";
   }

   mysqli_close($db);
}
else {
   echo "erreur connexion mysqli: ", htmlentities(mysqli_connect_error());
}

// ------------------
// En mysqli style OO

if (($db = new mysqli($host, $user, $pw, $dbname))
    && !$db->connect_errno) {
   $sql_query = "INSERT INTO user(login, password, email) VALUES('maurice', "
                . "'test', '"
                . $db->real_escape_string($email) // éch. manuel
                . "')";

   if ($st = $db->query($sql_query)) {
      $id = $db->insert_id;
      echo "utilisateur ajouté";

      if ($result = $db->query("SELECT login, email FROM user")) {
         while ($row = $result->fetch_array(MYSQLI_ASSOC)) {
```

```
            echo "login: ", htmlentities($row["login"]),
              " email: ", htmlentities($row["email"]);
         }

      // validation de $id encore nécessaire!
      $sql_query = "DELETE FROM user WHERE (id = " . $id . ")";
      $db->query($sql_query); // err. ign.
      }
      else {
         echo "erreur à traiter";
      }
   }
   else {
      echo "le login existe déjà ou autre erreur à traiter!";
   }

   $db->close();

   $db = null;
}
else {
   echo "erreur connexion mysqli/OO: ", htmlentities(mysqli_connect_error());
}

// ----------------------------------------------------------------------------
// En mysqli en mode orienté-objet et prepare statements (binding, sécurité)

if (($db = new mysqli($host, $user, $pw, $dbname))
    && !$db->connect_errno) {
   if ($st = $db->prepare(<<<EOD
INSERT INTO user(login, password, email)
   VALUES('maurice', 'test', ?)
EOD
   )) {
      $st->bind_param("s", $email); // binding positionnel d'une string
      if ($st->execute()) {
         $st->close();

         $id = $db->insert_id;

         echo "utilisateur ajouté";

         if (($st = $db->prepare("SELECT login, email FROM user"))
             && $st->execute()) {
            $st->bind_result($login, $email);

            while ($st->fetch()) {
               echo "login: ", htmlentities($login),
                    " email: ", htmlentities($email);
            }

            $st->close();

            if ($st = $db->prepare("DELETE FROM user WHERE id = ?")) {
```

```
            // binding positionnel, de type entier (integer)
            $st->bind_param("i", $id);
            $st->execute(); // err. ign.
            $st->close();
         }
         else {
            echo "erreur à traiter";
         }
      }
      else {
         echo "erreur à traiter";
      }
   }
   else {
      echo "le login existe déjà ou autre erreur à traiter!";
      $st->close();
   }
}
else {
   echo "erreur au prepare: ", htmlentities($db->error);
}

$db->close();

$db = null;
}
else {
   echo "erreur connexion mysqli/OO/bind: ", htmlentities(mysqli_connect_error());
}

?>
```

4.3 Assurer la sécurité, la performance et éviter le sur-codage

On remarque déjà que l'on peut obtenir les tuples (lignes de données) de résultats d'un `SELECT` sous forme d'un tableau associatif. On peut utiliser le fait que le nom des colonnes est la clé du tableau pour l'affichage, par exemple par des templates paramétrables.

Aussi, lorsqu'on insère (`INSERT`) ou met à jour des données (`UPDATE`), on peut utiliser du **binding** sur la base d'un tableau associatif nom des colonnes et valeurs correspondantes.

Il peut être très intéressant de centraliser ce genre de choses dans une fonction paramétrable !

Si l'on utilise encore l'échappement manuel plutôt que le binding, voici un exemple de fonction d'insertion plus générique :

```
# Insère le tuple $values (tableau associatif nom de colonne => valeur)
# passé en paramètre dans la table de la BD indiquée. Echappe les valeurs.
function mysql_insert($db, $table, $values) {
   $p = function($v) use ($db) {
      return "'" . mysqli_escape_string($db, $v) . "'";
   };
```

```
    $cols = join(', ', array_keys($values));
    $vals = join(', ', array_map($p, array_values($values)));

    $sql_query = 'INSERT INTO ' . $table . '(' . $cols . ') '
                 . 'VALUES(' . $vals . ')';

    return mysqli_query($db, $sql_query);
}
```

Noter l'usage de la fonction PHP `mysqli_escape_string()` et d'une fonction appliquant une autre fonction à chaque élément d'un tableau (`array_map()` – programmation fonctionnelle). On suppose bien évidemment ici qu'à la fois la variable `$table` et les noms des colonnes (clés du tableau de hachage `$values`) ne sont pas dangereuses et donc ne doivent pas être validées.

En reprenant l'exemple d'insertion ci-dessus, le code appelant deviendrait :

```
# $login et $email peuvent avoir été soumis par l'utilisateur ou
# la base de données et sont donc potentiellement dangereux, mais
# l'échappement est fait par notre fonction mysql_insert()
if ($result
       = mysql_insert($db,
                      'user',
                      ['login' => $login, 'email' => $email])) {
   echo "utilisateur ajouté";
}
else {
   # pourrait p.ex. être une erreur de clé primaire déjà existante
   echo "le login existe déjà ou autre erreur à traiter!";
}
```

Attention au fait que les chaînes *ne figurant pas entre apostrophes* (p.ex. nom de la table, nom des colonnes, clauses d'un ORDER BY, ...) devraient être *validées* (entier, présence dans un tableau de valeurs sûres, etc) plutôt qu'échappées – ce qui n'aurait pas d'effet !

La fonction PHP `mysql_insert()` décrite ci-dessus comporte cependant toujours quelques faiblesses (dépendance d'une BD particulière p.ex.). De plus, à force de simplifier et d'améliorer son code PHP, on risque bien de finir par implémenter un micro framework ... ce qui sera le but pédagogique du cours.

4.4 Recommandations

Quelques recommandations :

- nommer les bases de données, tables et colonnes uniquement en minuscules pour une meilleure compatibilité : utiliser le souligné si nécessaire
- utiliser le **backend InnoDB** car c'est le seul qui supporte les contraintes d'intégrité et référentielles (**Foreign Keys**)
- ne pas oublier de sauvegarder régulièrement la base de données
- apprendre les requêtes complexes SQL pour optimiser les requêtes
- utiliser un framework d'accès BD ou une couche d'abstraction pour les applications plus complexes.

4.5 Vers les frameworks

La bibliothèque **PDO**, en standard dans les versions actuelles de PHP, implémente un modèle objet pour l'accès de bases de données, de manière indépendante du type de base de données : on parle de modèle **ORM** simple.

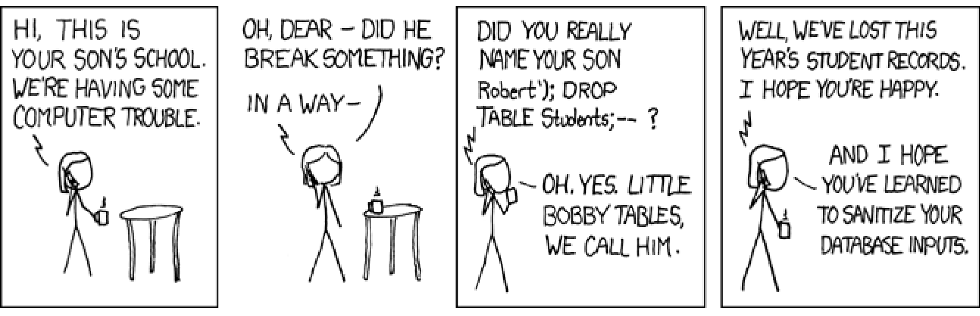
Des frameworks orientés **MVC** proposent de lier des objets du modèle directement à la base de données, et de gérer le flot de contrôle et la présentation, souvent par des templates.

Les frameworks sont des ensembles de fonctions, méthodes, classes, objets et méthodologies structurant votre code et permettant d'améliorer la qualité du code, la sécurité et la maintenance. Cependant, ils nécessitent un apprentissage et créent de nouvelles dépendances à des bibliothèques logicielles, pas forcément installées chez votre hébergeur.

Le framework **Perl Catalyst** est un excellent exemple de framework MVC intégrant tous ces avantages, qui peuvent alternativement être utilisés indépendamment du framework qui agit comme un *catalyseur*. Nous pouvons aussi citer **Perl Mojolicious**, **Java struts**, **PHP Laravel**, **PHP Symfony**, **PHP Zend**, ou encore **Ruby on Rails**.

Chapitre 5

Sécurité des applications web



http://xkcd.com/327/ – Usage autorisé explicitement par l'auteur

Sommaire

5.1 La problématique **106**
5.2 Types d'attaques **106**
5.2.1 Injection 106
5.2.2 Cross-scripting / XSS 107
5.2.3 Intégration de contenu / click-jacking 107
5.2.4 Social engineering 107
5.3 Méthodes de protection **107**
5.3.1 Introduction 107
5.3.2 Validation 108
5.3.3 Echappement 109
5.3.4 Confinement 110
5.3.5 Tainting (coloriage) 110
5.3.6 Signalisation hors-bande ou en-bande 111
5.3.7 CAPTCHA 112
5.3.8 Chiffrement 112
5.3.9 Recommandation pour l'authentification 112
5.3.10 Recommandations PHP 113

Le but de ce chapitre est de donner quelques informations de base sur la conception sécurisée d'une application Internet, sans vouloir être un cours complet traitant de sécurité informatique [1]

1. Le cours sécurité et le cours de développement web de 3e traiteront plus en profondeur ces aspects, notamment les attaques de **cross-scripting** (**XSS**).

5.1 La problématique

La plupart des problèmes de sécurité découverts aujourd'hui ne sont plus les classiques **buffer overflows** des langages comme C, mais exploitent souvent des failles multiples dans de nombreux logiciels interconnectés, voire du **social engineering**. La sécurité d'un système est dépendante de la sécurité de son plus faible maillon. En ce sens, un simple Wiki ou Webmail non sécurisé peut amener à des opérations incontrôlées sur un intranet ou une application web.

Or, la plupart du temps, ces problèmes sont dérivés de deux problèmes fondamentaux : une trop grande liberté pour l'application (pas de contraintes de privilèges) et la **signalisation en-bande** (ou l'**injection**). Ce dernier concept nous vient des réseaux et décrit tout système dans lequel la signalisation insérée par le système lui-même (p.ex. établir un appel téléphonique, insérer des données dans une base de données) n'est pas complètement séparée des données contrôlables par l'utilisateur.

Il est aussi parfois important d'assurer que l'on a affaire à un véritable utilisateur plutôt qu'un protocole automatique d'attaque, en particulier dans le contexte du web social, par exemple avec un **CAPTCHA**.

5.2 Types d'attaques

On trouvera une caractérisation détaillée de ce qui suit ainsi que de nombreux types d'autres attaques liées au développement d'application web ainsi que des conseils pour les éviter sur [20]. En reprenant la classification 2017 de OWASP, ce chapitre traite surtout des vulnérabilités A1 et A3, avec quelques éléments de A7 et de A9 traités durant le cours [2]

5.2.1 Injection

Une attaque d'injection est une attaque lors de laquelle on insère des données d'un attaquant et qui seront interprétées de manière incontrôlée, que cela soit dans une base de données, dans des structures de données web (pour affichage au client, mise à disposition de liens ou de code Javascript), ou dans du code quelconque.

Par exemple :

- injection de données
- injection de code HTML : pour cross-scripting p.ex.
- injection de code SQL : passer outre des autorisations d'accès, voler des tokens d'authentification, effacer, ajouter ou modifier des données
- injection de code Javascript : pour voler des cookies, changer le comportement de sécurité du navigateur, etc
- injection de code PHP : faire exécuter du code côté serveur, contrôlé par l'attaquant ; plusieurs techniques existent, les plus intéressantes consiste à forcer un include distant (suite à une mauvaise validation de paramètres) ou à abuser une configuration incorrecte du serveur vis-à-vis des paramètres exprimés non pas classiquement dès le séparateur ? mais avec /, et d'exécuter un fichier non PHP comme PHP.
- injection de code shell : exécuter n'importe quel code côté serveur
- injection de code quelconque : similaire aux **buffer overflows**

2. OWASP est traité plus en détail en 3e année : notamment Application Web II et la sécurité du logiciel, en général, dans le cadre du cours Sécurité.

5.2.2 Cross-scripting / XSS

Cette attaque, qui peut mener à l'activation de fonctions non voulues par l'utilisateur, est souvent rendue possible par une injection, par exemple de code HTML. Mais même sans injection, on peut mener l'utilisateur à suivre un lien ou soumettre un formulaire qui mènera à une action authentifiée non désirée sur un autre site. Pour l'éviter, en plus d'une bonne politique de validation et d'échappement, on ajoutera une confirmation HTML ou Javascript aux actions en cas de simple liens, et un champ caché (**hidden**) avec une valeur aléatoire connue du serveur (p.ex. grâce à la session) pour assurer l'intention de l'utilisateur.

5.2.3 Intégration de contenu / click-jacking

L'intégration de contenu d'un site dans un autre, par exemple via des éléments `frame`, `iframe`, `embed` ou `object` peut créer des risques : il est possible d'interdire l'inclusion avec un entête **X-Frame-Options** que l'on peut même systématiquement envoyer avec le serveur web [3] ou en fonction des besoins dans les entêtes générés par l'application.

L'entête HTTP **Content-Security-Policy** permet de configurer la directive **frame-ancestors** qui est une méthode plus moderne, pour les clients web qui le supportent, que celle indiquée ci-dessus.

5.2.4 Social engineering

On entend par **social engineering** toutes les méthodes dépassant les techniques informatiques permettant d'obtenir un résultat, par exemple de récolter des données protégées (**phishing**) ou même de pirater un compte. En effet, quoi de plus simple que se faire passer pour l'administrateur système, par e-mail ou téléphone, et obtenir des informations sensibles.

5.3 Méthodes de protection

5.3.1 Introduction

Dans notre cours, nous verrons principalement les méthodes manuelles de protection. Lorsque vous utiliserez plutôt un **framework**, une partie de ces méthodes seront automatiquement implémentées. Charge à vous de déterminer lesquelles le sont, dans quels cas, en lisant la documentation et en effectuant des tests.

Les règles de la **programmation défensive** sont un précieux atout pour éviter des abus, mais ne sont pas forcément suffisantes. Des connaissances techniques des problèmes des langages utilisés sont absolument nécessaires [4].

Le choix d'une méthodologie de conception (**MVC**), d'une méthode de développement, de cycle de vie et de test (p.ex. **XP**), d'un langage adapté et d'un framework éventuel intégrant déjà des notions de sécurité permet déjà d'assurer une certaine qualité et sécurité de l'application,

3. https://tecadmin.net/configure-x-frame-options-apache/

4. notamment dans les versions de PHP antérieures à 5.4.0, citons le **safe-mode**, à activer, ou le **register-globals** de PHP, à désactiver, notamment, sans oublier les très perturbantes fonctions magiques de ce langage comme les **magic-quotes** : si vous ne pouvez les désactiver, il vous faudra *enlever* les backslashes avant de traiter les paramètres !

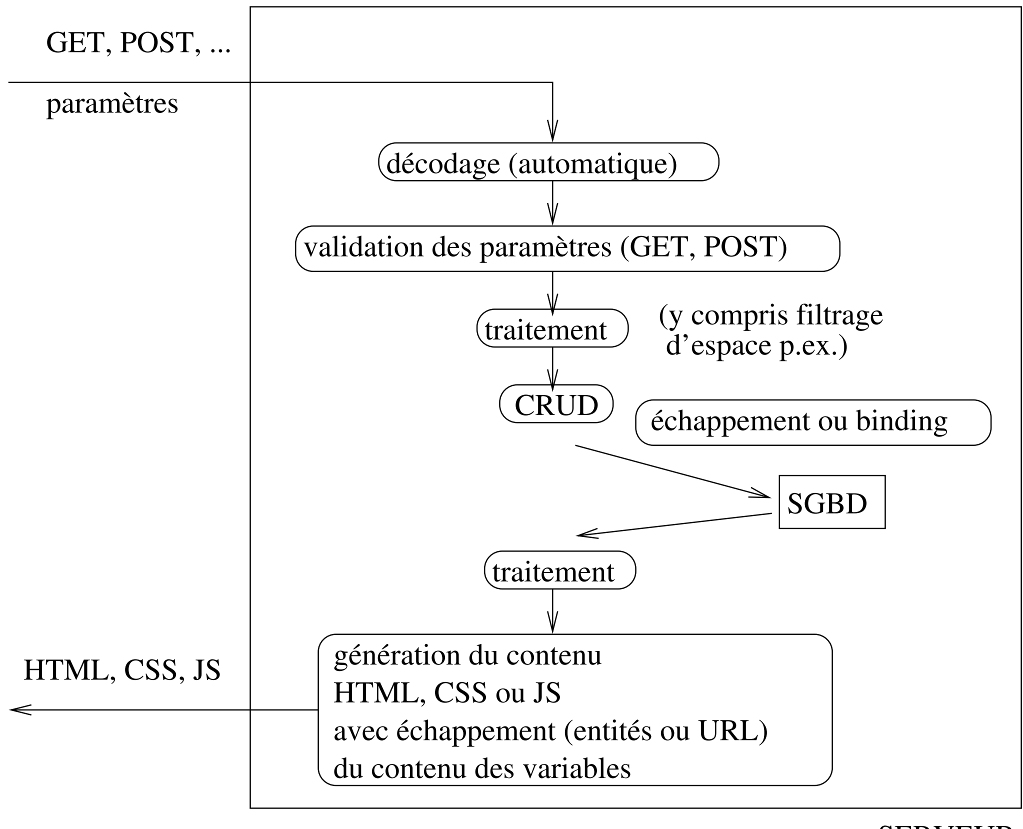


FIGURE 5.1 – Flot de validation et d'échappement d'une application Internet typique

notamment en évitant largement la duplication de code et en simplifiant le flot de contrôle de l'application (voir la figure 5.1 en page 108).

Quelques symptômes classiques d'un mauvais flot de validation et d'échappement :

— la base de données contient des caractères d'échappement dans ses tuples (p.ex. si vous trouvez les chaînes `\'` ou `&`)
— les données affichées sur le client web contiennent des caractères d'échappement visibles
— les séquences `'`, `&` ou `<hr />` écrites telles quelles dans un champ de formulaire ne sont pas stockées telles quelles dans la base données ou les deux dernières sont interprétées plutôt qu'affichées dans le client web
— des URLs contiennent des caractères interdits.
— l'HTML généré n'est pas valide.

5.3.2 Validation

Toute donnée provenant de l'utilisateur devrait être validée (par exemple par une **expression régulière**). Cela peut se faire simplement par une vérification ou un forçage de type [5], ou par une **expression régulière**. Idéalement, cette validation devrait être faite *au bon moment* (voir la figure 5.1 en page 108 pour le placement idéal des validations et échappements).

L'intégrité référentielle et les contraintes d'intégrité directement implantables dans la base de données forment la dernière ligne de défense de l'application, mais n'est pas suffisante : de la

5. En Perl, par exemple, pour garantir par exemple qu'une variable contient un entier, il suffit d'y ajouter l'entier 0.

validation, ou au minimum de l'échappement ou du **binding** sont nécessaires lors des échanges avec la base de données.

Il est totalement faux de ne faire des validations qu'en Javascript ou en **HTML5**. La validation des données est une partie indispensable de tout script côté serveur : le Javascript permet, côté client, d'éviter une certaine lourdeur dans la notification des problèmes, d'améliorer l'interface utilisateur et d'éventuellement de limiter la charge du serveur, mais pas d'améliorer la sécurité.

Pour illustrer la difficulté de bien concevoir une validation dans des langages à typage dynamique, voici un exemple surprenant en PHP :

```
<?php if ($_POST['p'] == '1234') { echo "Welcome"; }
else { echo "Tsss..."; } ?>
```

En effet, si le paramètre `p` est mis à `0x4d2`, cela fonctionnera aussi (ainsi que toute chaîne évaluable par conversion de type à 1234. Il faudrait ici utiliser l'égalité de type et de valeur (===, voir figure 3.1 en page 68).

Et un en Perl :

```
my $q = new CGI;
# vérifions l'inocuité de $q->param('password')
if (defined($q->param('password')) && ($q->param('password') =~ /^[a-z0-9]+/)) {
   # appel d'une insertion de base de données non protégée p.ex.
}
```

Ce code est peut-être vulnérable si le paramètre `password` est passé de multiples fois (la validation doit se faire sur tous les éléments du tableau, pas juste le premier, ou on doit vérifier que la variable est de type scalaire). En effet, en Perl le type de `$q->param('password')` sera défini en fonction du contexte : soit un scalaire, soit un tableau. En PHP, un paramètre de GET ou de POST dont le nom finit par `[]` devient automatiquement un tableau dont le nom ne comporte pas les crochets : il est donc recommandé de tester avec `is_scalar()` pour éviter des effets étranges.

5.3.3 Echappement

Toute donnée provenant de l'utilisateur et étant réaffichée au client, ou toute donnée provenant d'une base de données pour réaffichage, doit être protégée de l'interprétation par le client (échappée). En pratique, cela se fait en passant toute chaîne susceptible de contenir du balisage en (X)HTML (notamment & et <, mais pas seulement) à une fonction d'échappement, par exemple en PHP `htmlentities()` [6], ou en Perl via `HTML::Entities::encode_entities()` ou `CGI::escapeHTML()` (voir section 2.1.7 en page 34).

Les paramètres d'URL présentées au client doivent être encodées encore plus spécialement (en encodant certains caractères spéciaux en les précédant du caractère % suivi du code hexadécimal à 2 chiffres du caractère, comme %20 pour l'espace – qui peut aussi être représenté comme +). On utilisera pour ce faire les fonctions `urlencode()` (PHP) et `URI::Escape::uri_escape()` (Perl).

De plus, l'interface textuelle (**signalisation en-bande**, voir section 5.3.6 en page 111) à une base de données nécessite de l'échappement de caractères particuliers, qui dépendent de la base

6. pour la problématique complète, voir section 3.2.17.1 en page 88

de données. En PHP, on utilisera par exemple, pour MySQL, `mysqli_escape_string()`. En Perl ou avec des frameworks PHP, on préférera utiliser le concept de **binding** (voir section 5.3.6.2 en page 111).

Enfin, en cas d'utilisation d'outils de **templating**, pour générer du contenu HTML, XML ou autre depuis un chablon et des données, il faudra également assurer l'échappement. Par exemple, avec le **Template Toolkit**, il faut utiliser le filtre `htmlentities` pour chaque donnée à échapper.

Ajoutons que si l'on utilise l'apostrophe plutôt que le guillemet usuel comme délimiteur des valeurs d'attributs d'éléments HTML, la fonction htmlentities() ne suffit pas. De même si l'on veut protéger des chaînes qui ne sont pas entre guillemets (p.ex. le nom d'un élément plutôt que sa valeur).

5.3.4 Confinement

Toute application devrait être confinée à un ensemble d'opérations, à des droits d'accès et à la visibilité de certains fichiers. Suivant les langages et environnements d'exécutions, les options suivantes sont disponibles :

virtualisation confinement par virtualisation de divers types : du simple confinement `chroot` jusqu'à la véritable machine virtuelle

droits d'accès en exécutant l'application sous des droits d'accès (UNIX : UID, GID) limités. Cela s'applique à l'exécution de l'application PHP (propre UID – par exemple avec **suPHP** ou **php-fpm** – avec les permissions bien configurées) aussi aux connexions à la base de données : plusieurs utilisateurs peuvent être définis avec chacun leurs règles d'accès aux tables.

MAC en applicant des règles explicites de contrôle d'accès (*Mandatory Access Control*), basées sur le le nom de l'application et des chemins globalement autorisés ou interdits : p.ex. **app-armor** : ces règles éviteront les problèmes les plus graves suite à un piratage, ou au moins permettront de le détecter par les journaux systèmes produits.

5.3.5 Tainting (coloriage)

L'idée du tainting, tel qu'il est disponible notamment au sein du langage Perl, est de colorier les variables provenant de l'utilisateur (arguments passés à un programme sur la ligne de commande, paramètres de formulaires ou de requêtes, etc), en contaminant (coloriant) automatiquement tout résultat d'opération [7] de variables déjà coloriées.

Si à un moment donné de l'exécution, une variable coloriée (taintée) est utilisée comme paramètre d'une fonction sensible, une exception est soulevée, et l'exécution du programme est terminée si elle n'est pas traitée.

Cet outil est donc appréciable pour détecter d'éventuelles failles dans le flot de validation.

7. En Perl, seule l'extraction de sous-chaînes via une recherche d'expressions régulières permet de dé-teinter une variable. En bref, une validation.

5.3.6 Signalisation hors-bande ou en-bande

5.3.6.1 Problématique

Comme déjà vu à de nombreuses reprises, de nombreux problèmes de sécurité sont liés au fait que des données sont mélangées au contrôle (signalisation, méta-données, commandes, ...) et lorsque des données sont par erreur interprétées comme du contrôle, des attaques spécifiques sont possibles. Lorsque les données sont en-bande (mélangées), un échappement correct est indispensable. Une meilleure méthode est de séparer de manière claire ces deux types d'informations : on parle alors de signalisation hors-bande. Ces notions sont universelles.

5.3.6.2 Binding

L'interfaçage de bases de données par des commandes textes mélangeant commandes SQL (**SQL statements**) et données est un cas typique de **signalisation en-bande**, selon l'exemple Perl suivant : on devrait, dans le cas général, échapper le contenu de la variable, ce qui n'est pas fait :

```
# $email et $name viennent de l'utilisateur ou de la base de données
# et ne sont pas échappés pour cette base de données, ce qui est
# incorrect.
my $sql_statement = <<EOF;
INSERT INTO table(name, email)
   VALUES ($name, $email);
EOF
$dbh->prepare($sql_statement);
$dbh->execute();
```

(la variable est affectée ici via un **HERE document** du langage Perl)

Pour éviter ces problèmes, le **binding** permet, par exemple en Perl avec **DBD** ou les **prepared statements** de **mysqli** ou de **PDO** en PHP, d'assurer la sécurité en séparant données et signalisation par association de variables :

```
# $email et $name doivent être tels quels
# (non échappés pour cette BD)
my $sql_statement = 'INSERT INTO table(name, email) VALUES(?, ?)';
$dbh->prepare($sql_statement);
$dbh->execute($email, $name);
```

L'idée est ici de soit laisser la méthode `execute()` effectuer le bon échappement qui correspond à la bonne base de données, ou d'exécuter la commande SQL par une interface non texte (XML, binaire) avec contrôle du format (voir la section suivante). Vu que les données sont séparées syntaxiquement des ordres SQL, on peut parler ici de **signalisation hors-bande**.

Attention, certains problèmes de sécurité restent possibles : par exemple le code suivant, en supposant que `$pw` vient de l'utilisateur. En effet, le `LIKE` autorise l'usage de métacaractères au contraire du =.

```
# retournera tous les tuples si $pw = '%'
$dbh->execute('SELECT id FROM utilisateur WHERE password LIKE ?', $pw)
```

5.3.6.3 Formats stricts

Des formats stricts, validés, comme XML, permettent d'assurer – si l'échappement des données est fait correctement – une signalisation distincte des données. Par contre, ce n'est pas strictement de la **signalisation hors-bande**, juste du formatage strict et validable.

5.3.7 CAPTCHA

Le principe d'un CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) est de distinguer un programme informatique d'un humain, dans le but d'éviter, notamment, le **SPAM**.

Cela peut se faire de diverses manières (calcul simple à effectuer par l'utilisateur, images encodant de manière trouble des nombres ou lettres, etc).

Malheureusement, la plupart des systèmes de CAPTCHA ont leurs vulnérabilités techniques – et non techniques : on peut montrer que si l'on cache derrière un CAPTCHA X des informations très recherchées[8], cela incitera des utilisateurs à résoudre le CAPTCHA. Mais si ce CAPTCHA X est en fait une copie d'un CAPTCHA Y qui lui sert à protéger un site contre le SPAM, on a une méthode très efficace, distribuée, et dotée d'une multitude de cerveaux humains disponibles pour passer outre cette protection. C'est un cas très particulier de **social engineering** (voir la section 5.2.4 en page 107).

De plus, la plupart des CAPTCHAs, s'ils sont bons, rendent l'accès difficile aux personnes handicapées. On préférera, en règle générale, l'identification via une adresse e-mail vérifiée (via un message de confirmation, qui lui peut être un CAPTCHA, cette fois unique), ou via un système d'authentification unique comme **OpenID**.

5.3.8 Chiffrement

HTTPS est devenu aujourd'hui quasi obligatoire (voir section 1.1.6.14 en page 12).

Toutefois, HTTPS ne protégera pas contre les attaques directement sur le client ou le serveur, ni contre certaines attaques du réseau[9] : il existe pour en réduire l'impact quelques bonnes pratiques, comme l'utilisation de **HSTS** (*HTTP Strict Transport Security*, forçant pendant une durée configurable l'utilisation d'HTTPS même lorsqu'un URL HTTP est précisé), la désactivation d'algorithmes vulnérables[10], l'utilisation de politiques HTTP concernant l'intégration d'un site dans un autre via framing ou de l'origine de composants d'une page **same-origin** (**CORS**, *Cross-Origin Resource Sharing*), l'utilisation de **PFS** (*Perfect Forward Secrecy*) ou d'un **reverse proxy** filtrant ou plus généralement d'un **WAF** (*Web Application Firewall* – forcément positionné hors du chiffrement).

5.3.9 Recommandation pour l'authentification

L'usage de **token** d'authentification plutôt que des mots de passe ou des cookies, en particulier avec les API REST, où l'on pourra limiter les droits et la durée de vie liés à un usage particulier,

8. Pas besoin de chercher très loin, quelques images suffisent ...
9. Exemples : certificat invalide accepté sur le client malgré les mises en garde, redirection entre HTTP et HTTPS pouvant être interceptée par un attaquant, attaque de **downgrading** visant à forcer le client à utiliser des versions vulnérables mais encore activées, vol de clé privée suite à une vulnérabilité comme **Heartbleed**, . . .
10. vérifiez votre certificat serveur par exemple avec `https://www.ssllabs.com/ssltest/analyze.html`

est un bon moyen d'assurer une meilleure sécurité et traçabilité. Un exemple standardisé de token est le *JSON Web Token* (**JWT**, RFC-7519).

On stockera les mots de passe côté serveur de manière hachée (avec un *salt*) afin de limiter les conséquences d'un vol de la base d'authentification. On assurera que le transport de ceux-ci est effectué en HTTPS, et on choisira entre une authentification par l'application, par le serveur web, ou une combinaison des deux (voir section 1.1.6.13 en page 12)

En cas d'usage de **cookies**, il est recommandé d'ajouter l'attribut `Secure` de manière à ce que le cookie ne soit envoyé qu'en HTTPS. On peut interdire le Javascript d'utiliser les cookies de l'utilisateur avec l'attribut `HTTPOnly`.

Parmi les protocoles liés à l'authentification, **OAuth** (RFC-6749) est un protocole de délégation d'authentification qui est particulièrement utile pour authentifier des agents (un site web ou une application mobile utilisant l'API d'un autre site web).

5.3.10 Recommandations PHP

— si votre version de PHP est inférieure à 5.4.0, vérifiez que les diverses options dangereuses de PHP sont désactivées (**register-globals**, **magic-quotes**, etc) et que le **safe-mode** est activé.

— en particulier lors d'hébergement de masse (voir 1.3.3.2 en page 18), demandez à ce que votre application soit exécutée sous son utilisateur (UID) propre (voir 5.3.4 en page 110), et vérifiez les versions des logiciels (interprète, bibliothèques, ...) utilisés.

— utilisez un fichier de configuration séparé pour y stocker les informations de paramétrisation (base de données, etc). Cela a l'avantage de simplifier le déploiement de test et d'éviter des erreurs.

— généralement, si vos fichiers inclus contiennent des informations sensibles (mot de passe de la base de données p.ex.), il est recommandé de soit de les terminer par une extension `.php` et de s'assurer que leur exécution indépendamment du programme principal, via Apache, soit non dangereuse ; soit de les déposer dans un répertoire interdit à l'accès Apache et alors de les terminer par `.inc`

— vous pouvez limiter les droits d'accès depuis le web par exemple avec un fichier `.htaccess`.

— un fichier `.htaccess` peut aussi servir à configurer certaines options PHP

Vous trouverez d'autres recommandations dans [17].

Références et bibliographie

[1] Luke WELLING, Laura THOMSON & Eric JACOBONI, *PHP et MySQL*, 4e édition, Pearson Education, ISBN 2-74402308-6.

[2] http://www.php.net/

[3] http://www.manuelphp.com/cours/

[4] http://www.lephpfacile.com/

[5] http://www.phpdebutant.org/

[6] http://www.css-faciles.com/ : Introduction, propriétés des feuilles de style

[7] http://www.openweb.eu.org/css/ : Divers thèmes spécifiques : exemples de positionnement, utilisation d'images, etc.

[8] http://www.csszengarden.com/ : Déclinaison d'un site web avec des centaines de feuilles de style

[9] http://www.w3.org/TR/REC-CSS2/ : Normes CSS2 (version officielle), voir aussi http://www.yoyodesign.org/doc/w3c/css2/cover.html (traduction française de la norme)

[10] http://www.htmldog.com/reference/cssproperties/ : Liste des propriétés CSS

[11] http://www.htmldog.com/guides/css/beginner/ et http://www.htmldog.com/guides/css/advanced/ : Guides en ligne CSS

[12] Sommaire de référence CSS, http://www.validome.org/doc/HTML_fr/navigation/css.htm

[13] Jérôme MOLIÈRE, *Les cahiers du programmeur J2EE*

[14] Richard R. BAUD, *Guide pratique et progressif du langage Javascript*, http://richard.geneva-link.ch/951v.html

[15] Predrag VICEIC, *WebDAV, du HTTP au partage de fichiers*, Flash informatique 6/2008, EPFL, http://flashinformatique.epfl.ch/IMG/pdf_6-8-page20.pdf

[16] Predrag VICEIC, *WebDAV – la recherche, les droits d'accès et versioning*, Flash informatique 7/2008, EPFL, http://flashinformatique.epfl.ch/IMG/pdf_7-8-page1.pdf

[17] Glück ORTWIN, *PHP Best Practices*, http://www.odi.ch/prog/design/php/guide.php

[18] IANA, *Liste des MIME types officiels*, http://www.iana.org/assignments/media-types/media-types.xhtml

[19] Kieren DIMENT & Matt S. TROUT, *The definitive guide to Catalyst : writing extensible, scalable, and maintainable Perl-based Web applications*, 2009, Apress, ISBN 978-1-4302-2365-8.

[20] The Open Web Application Security Project (OWASP), *OWASP Top Ten Project*, http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[21] Norman WALSH, *DocBook 5 : The Definitive Guide*, 2010, O'Reilly, ISBN 978-059680501-2.

[22] Erik T. Ray, *Learning XML, 2nd Edition*, 2003, O'Reilly, ISBN 978-059600420-0.

[23] Doug Tidwell, *XSLT, 2nd Edition*, 2008, O'Reilly, ISBN 978-059652721-1.

[24] XML Schema Reference, W3 Schools, http://www.w3schools.com/schema/

[25] Marc Schaefer, *Polycopié Réseaux ISC1*, 2023, ISBN 978-2-940387-07-6

[26] Marc Schaefer, *Gitlab des cours*.

Lexique

CGI Common Gateway Interface : standard permettant à un serveur HTTP de produire des pages HTML de façon dynamique : Au lieu de renvoyer le contenu d'un fichier (HTML ou image) tel qu'il est stocké, le serveur exécute un programme (écrit dans n'importe quel langage, pourvu que le fichier soit exécutable), et c'est la sortie de cette exécution qui sera retourné au client.

En général des paramètres sont passés au programme (par exemple les termes à trouver, dans le cas d'un moteur de recherche) qui les prendra en compte pour générer la page de résultat.

CRUD Create Read Update Delete : les opérations de base que l'on veut implémenter dans une interface web de modifications de données. Ces opérations sont si courantes que de nombreux frameworks les implémentent par défaut. La philosophie **REST** assigne à chacune des méthodes ou verbes HTTP de base une sémantique particulière correspondant aux méthodes de base CRUD (voir section 1.1.6.10.2 en page 10).

framework Un framework est une bibliothèque logicielle qui impose certaines conventions à l'utilisateur. Plus précisément, on retrouve en général certaines caractéristiques communes[11] comme l'inversion de contrôle (le framework décide quand il appelle partie de votre code, sous la forme de **callback**s par exemple pour un framework orienté **programmation événementielle**, cette décision est basée sur l'association de méthodes à des événements, au sein de votre code ou d'un fichier de description externe – l'inversion de contrôle peut alors être vue comme un cas particulier d'injection de dépendances), des comportements par défaut, et une extensibilité selon le principe open/closed[12] (extensibilité par redéfinition de méthodes et classes et non pas par modification du code du framework lui-même).

HTML Hypertext Markup Language : langage de balisage utilisé pour décrire le contenu des pages web, respectant la syntaxe définie par SGML.

JSP JSP (Java Server Pages) Technologie Sun, basée sur les servlets, permettant d'embarquer du code java dans des pages HTML qui porteront alors l'extension .jsp.

Microsoft ASP Active Server Pages : à ne pas confondre avec Application Service Provider (un hébergeur de services et d'applications). Technologie Microsoft de génération dynamique de pages web (templating). Nécessite une plate-forme Windows avec IIS (Internet Information Server), les scripts seront écrits en VBScript ou Jscript (langages interprétés) et seront embarqués dans des pages HTML portant l'extension .asp. Existe même en version .NET (.aspx).

proxy Serveur mandataire. Effectue un relais entre le client et le serveur d'un protocole. Dans le cas d'HTTP, il peut être combiné à un **cache** et/ou à un filtrage (sécurité).

ORM Object-Relational Mapping : une technique de programmation qui permet de présenter à l'application une interface orientée objet permettant l'interrogation d'une base de données relationnelle.

regex Expression régulière : grammaire permettant de décrire ou traiter des chaînes de caractères à partir d'un modèle utilisant une syntaxe particulière. Par exemple, l'expression régulière suivante : `^[0-9]*$` décrit une chaîne vide ou formée uniquement de chiffres.

SGML Standard Generalized Markup Language : c'est un langage normalisé qui utilise des balises pour décrire la structure et la nature des informations contenues dans un

11. voir notamment `http://en.wikipedia.org/wiki/Software_framework`
12. selon la mnémonique SOLID, consulter `http://en.wikipedia.org/wiki/Solid_%28object-oriented_design%29`

document. Il intègre le concept de méta-données, soit la possibilité de créer ses propres balises en fonction de ses besoins.

XML eXtensible Markup Language : Version simplifiée (sous-ensemble) de SGML permettant de produire des documents structurés en respectant une syntaxe stricte.

XHTML eXtensible Hypertext Markup Language : successeur d'HTML, ce langage de balisage, utilisé surtout en entreprise en présence d'une chaîne de traitement XML, respecte la syntaxe (plus stricte) définie par XML, avec notamment l'obligation de réaliser des document bien formés (i.e. tous les éléments doivent être balisés, les balises fermantes sont obligatoires, ...).

Index des concepts

*AMP, 66
.htaccess, 113
===, 109
2.0, 13
3.0, 13

A, 4, 15
a, 38
absolu, 4
Accept, 48
accesskey, 51
ACID, 92
action, 48
Adminer, 66, 93
adminer, 66
AJAX, 12, 14, 62
Akamai, 19
Amazon S3, 18
annuaire, 14
ANY, 21
Apache, 66
app-armor, 110
Applet, 59
application/x-www-form-urlencoded, 49
application/xml, 48
arguments par défaut, 80
association, 77
    ordonnée, 77
associativité, 68
ATOM, 13
attribute, 25
auth/basic, 4, 12
auth/digest, 4, 12
auto-globaux, 73, 84, 86
autoloaders, 81
Autonomous System, 18

B2B, 17
B2C, 17
backend, 103
backtick, 76
balises, 20, 64, 65
base64, 6, 11
big data, 92
binding, 93, 95, 102, 109–111
block, 22, 37, 43
body, 22, 37
BOSH, 14
buffer overflows, 106

CA, 13
cache, 117
Cache-control, 9
callback, 117
CAPTCHA, 106, 112
cascade, 56
Catalyst, 62, 104
CDN, 19
CGI, 62, 117
chaîne de confiance, 13
charset, 6, 11, 35, 43, 49
    ISO-8859-1, 35, 43, 49
    ISO-8859-15, 35
    ISO-latin-1, 35
    UTF-8, 11, 35, 49
choice, 27
click-jacking, 107
closures, 81
cloud, 63
CloudFlare, 19
Comet, 14
complexType, 27
Composer, 63
connection tracking, 15
connexion, 6
Content-Security-Policy, 107
Content-type, 48, 88
Continuous Integration, 66
contraintes d'intégrité, 92
contrôle de version, 94
Contrôleur, 16
cookies, 5, 12, 64, 73, 84, 113
    HTTPOnly, 113
    Secure, 113
CORS, 112
cross-scripting, 105, 107
CRUD, 10, 91, 117

CSS, 33, 38, 52, 55–57
  cascade, 56
  display, 38
  héritage, 55, 56
  media queries, 57
  media types, 56
  spécificité, 55, 56
CSS3, 52

data center, 17
DBD, 111
DDoS, 19
de transfert, 27
delete, 10
descriptif, 20
display, 38
Django, 62
DMZ, 17
DNS, 4, 15
  A, 4, 15
  PTR, 4
DocBook, 21
document, 24
DOM, 14, 28, 45, 58
DOM Storage, 5
downgrading, 112
DTD, 21, 22, 25, 35
  ANY, 21
  EMPTY, 21
  HTML 4.01 strict, 22
  séquence, 21
défensive, 107
déploiement, 66

EasyPHP, 66
EBNF, 20
Eclipse, 66
element, 22, 25, 27
em, 37
EMPTY, 21
en-bande, 91, 106, 109, 111
enctype, 49
  application/x-www-form-urlencoded, 49
  multipart/form-data, 49
entites, 21
entities, 27, 109
entité, 35, 42
ergonomie, 44
Ethereal, *voir* Wireshark
exceptions, 67
eXist, 28, 92
expression régulière, 21, 64, 67, 108, 117

fast-cgi, 62
fat controller, 16
Fetch API, 14
fieldset, 45
firewall, 5, 13–15
  connection tracking, 15
Flash, 59
focus clavier, 44
foreach, 70
Foreign Keys, 103
forge, 66
form, 43, 48
  action, 48
format, 20
frame-ancestors, 107
framework, ii, 14, 58, 92, 103, 104, 107, 117
FTP, 9

GD graphics library, 87
get, 5, 8, 10, 43, 48, 64, 73
GIT, 10
global, 74, 81
GNU Emacs, 66
GNU GPL, 63
grammaire, 25

HaaS, 18, 19
head, 36
Heartbleed, 112
HERE document, 75, 111
heredoc, *voir* HERE document
hidden, 48, 107
hN, 37
hors-bande, 111, 112
HSTS, 112
htdocs, 5, 65
HTML, 4, 22, 33, 35–40, 43–45, 47–49, 107, 117
  a, 38
  block, 22, 37, 43
  body, 22, 37
  element, 22
  em, 37
  fieldset, 45
  form, 43
  head, 36
  hidden, 48, 107
  hN, 37
  html, 35
  img, 39
  inline, 22, 37, 38, 43
  input, 43

- label, 44
- legend, 45
- mark, 37
- p, 37
- select, 47
- strong, 37
- submit, 49
- table, 40
- textarea, 48
- validator, 33

html, 35
HTML 4.01 strict, 22
HTML5, 6, 14, 32, 51, 109
- Fetch API, 14
- WebRTC, 14
- Websockets, 14

HTTP, 6–10, 13, 48, 49, 88, 117
- 2.0, 13
- 3.0, 13
- Accept, 48
- Cache-control, 9
- Content-type, 48, 88
- get, 8, 48
- image, 49
- keep-alive, 6, 7
- method, 48, 117
- post, 48
- SPDY, 13
- status, 9
- verbe, 8, 10

HTTPOnly, 113
héritage, 55, 56

IaaS, 18, 19
idempotente, 9, 49
image, 49
image/png, 48
img, 39
include, 71
index, 94
injection, 91, 106
inline, 22, 37, 38, 43
InnoDB, 92, 103
input, 43
instance, 21
interpolation, 72, 75
ISO-8859-1, 35, 43, 49
ISO-8859-15, 35
ISO-latin-1, 35
isset(), 74

Java, 59, 62, 104
- Applet, 59
- JSP, 62
- Servlet, 62
- struts, 104
- Webstart, 59

Javascript, 14, 51, 58
JQuery, 14, 58
JSON, 12, 14
JSP, 62, 117
JWT, 113

keep-alive, 6, 7
KHTML, 2

label, 44
langage, 20
- descriptif, 20
- à balises, 20

Laravel, 104
legend, 45
libre, 63
licence, 63
- GNU GPL, 63

light controller, 16
load-balancer, 15
locale, 27
long polling, 14
long query, 14

magic-quotes, 64, 107, 113
mark, 37
markup, 20
markup language, *voir* langages à balises
media queries, 57
media types, 56
method, 5, 10, 43, 48, 64, 73, 117
- delete, 10
- get, 5, 10, 43, 64, 73
- patch, 10
- post, 5, 10, 43, 64, 73
- put, 10

Microsoft ASP, 62, 117
MIME, 5, 6, 11, 32, 48, 86
- application/xml, 48
- image/png, 48
- text/html, 48
- text/plain, 48
- types, 5, 6, 11

mod-perl, 62
Modèle, 16
Mojolicious, 62, 104
MonoWall, 63

multi-homing, 18
multipart/form-data, 49
MVC, ii, 15, 16, 28, 104, 107
  Contrôleur, 16
  fat controller, 16
  light controller, 16
  Modèle, 16
  Routeur, 16
  Vue, 16, 28
MVVM, 16
MyISAM, 92
MySQL, 66, 92, 93, 103
  Adminer, 93
  Foreign Keys, 103
  InnoDB, 92, 103
  MyISAM, 92
mysqli, 111

namespace, 25, 26
namespaces, 81, 88
NodeJS, 58
nonce, 12
NoSQL, 92

OAuth, 113
ODBC, 93
OpenID, 112
ordonnée, 77
ORM, 92, 93, 104, 117

p, 37
P2P, 14
PaaS, 18, 19, 63
paramètres, 4, 48, 73
paramètres multivalués, 83, 84
passage par référence, 80
patch, 10
PDO, 92, 94, 104, 111
Perl, 62, 104, 111
  Catalyst, 62, 104
  DBD, 111
  mod-perl, 62
  Mojolicious, 62, 104
  Template Toolkit, 62
PFS, 112
phishing, 107
PHP, ii, 64–67, 70–75, 77, 80–84, 86, 89, 92, 94, 104, 107, 111
  adminer, 66
  arguments par défaut, 80
  auto-globaux, 73, 84, 86
  autoloaders, 81
  balises, 64, 65
  closures, 81
  cookies, 73, 84
  EasyPHP, 66
  exceptions, 67
  global, 74
  include, 71
  interpolation, 72, 75
  isset(), 74
  Laravel, 104
  magic-quotes, 64, 107
  mysqli, 111
  namespaces, 81
  paramètres, 73
  paramètres multivalués, 83, 84
  passage par référence, 80
  PDO, 92, 94, 104, 111
  php.ini, 71, 86, 89
  phpmyadmin, 66
  portée, 73, 81
  register-globals, 64, 73, 107
  safe-mode, 107
  scalaire, 74
  session, 73
  static, 74
  super-globaux, 73
  Symfony, 104
  tableau, 70
  tableau associatif, 70, 77
  track errors, 67
  traits, 82
  typecast, 72
  variables prédéfinies, 73
  variables super-globales, 73
  varyadic, 82
  XAMPP, 66
  Zend, 104
php-fpm, 62, 110
php.ini, 71, 86, 89
phpmyadmin, 66
plugin, 2
PO, 89
polyfill, 2, 33
portée, 73, 81
post, 5, 10, 43, 48, 64, 73
PostgreSQL, 92
prepared statements, *voir* binding
procédures stockées, 92
programmation, 58, 107, 117
  défensive, 107
  événementielle, 58, 117

proxy, 13, 14, 39, 117
PTR, 4
PULL, 5, 14
PUSH, 5, 6, 14
put, 10
Python, 62
    Django, 62

QoS, 18
QUIC, 13
quoted-printable, 11

redirection, 10
    sémantique, 10
regex, 52, *voir* expression régulière, 117
register-globals, 64, 73, 107, 113
regular expression, *voir* expression régulière
relatif, 4
REST, 5, 10, 85, 117
RESTful, *voir* REST
reverse proxy, 112
RFC, 20
RGB, 54
road warrior, 17
round-robin, 15
Routeur, 16
RSS, 13
Ruby on Rails, ii, 104
RWD, 57
règles, 92

SaaS, 18
Saas, 19
safe-mode, 107, 113
same-origin, 112
sandbox, 58
Sass, 52
SaX, 28
SBGD, 92
    triggers, 92
scalability, 15
scalaire, 74
schema, 24
schémas, 92
SCSS, 52
Secure, 113
select, 47
Selenium, 33
sequence, 27
Server-Sent Events, 14
Servlet, 62
session, 5, 12, 73, 84
SGBD, 91–94
    ACID, 92
    contraintes d'intégrité, 92
    index, 94
    ODBC, 93
    ORM, 93
    procédures stockées, 92
    règles, 92
    schémas, 92
    transactions, 92
    vues, 92
SGML, 20, 21, 32, 117
    DTD, 21
    instance, 21
signalisation, 91, 106, 109, 111, 112
    en-bande, 91, 106, 109, 111
    hors-bande, 111, 112
signed cookies, 85
SIP, 14
SLA, 18
SMTP, 9
SOAP, 14, 27
social engineering, 106, 107, 112
sortie standard, 87
SPAM, 18, 112
SPDY, 13
spécificité, 55, 56
SQL statements, 111
SQLite, 92
SSI, 62
static, 74
status, 9
strong, 37
struts, 104
style sheets, 33
submit, 49
super-globaux, 73
suPHP, 110
SVG, 59, 87
Symfony, 104
syntaxe, 25, 27
    de transfert, 27
    locale, 27
system management industrialisation, 19
sémantique, 10
séquence, 21

table, 40
tableau, 70, 77
tableau associatif, 70, 77
tableau de hachage, 77

TCP, 6
template, 18
Template Toolkit, 62, 110
templates, *voir* templating
templating, 62, 65, 104, 110
    Template Toolkit, 110
text/html, 48
text/plain, 48
textarea, 48
tier, 15
token, 112
tooltip, 39
track errors, 67
traits, 82
transactions, 92
triggers, 92
typecast, 72
types, 5, 6, 11

Unicode, 27
UNIX, 4, 87
    sortie standard, 87
URI, 3
URL, 3, 4
    absolu, 4
    relatif, 4
urlencode, 109
UTF-8, 11, 27, 35, 49

validator, 33
variables prédéfinies, 73
variables super-globales, 73
varyadic, 82
verbe, 8, 10
virtualisation, 19
VLS, 15
vocabulaire, 25
VPN, 17
Vue, 16, 28
vues, 92

WAF, 112
WCAG, 51
Web 2.0, 13, 14
web 2.0, 5
Web services, 3, 10, 16, 27, 58, 62, 92
WebDAV, 9, 10
WebKit, 2
WebRTC, 7, 14
Websockets, 7, 14
Webstart, 59
Wireshark, 8
WSDL, 27

X-Frame-Options, 107
X.509, 13
XAMPP, 66
XHTML, 32, 118
XML, 12, 14, 20, 24–26, 28, 32, 118
    attribute, 25
    document, 24
    DOM, 28
    DTD, 25
    element, 25
    namespace, 25, 26
    SaX, 28
    schema, 24
    XPath, 28
    XQuery, 28
    XSD, 25
    XSearch, 28
    XSLT, 28
xmlcopyeditor, 29
XMLHttpRequest, 14
XMLRPC, 27
XP, 107
XPath, 28
XQuery, 28
XSD, 25, 27
    choice, 27
    complexType, 27
    element, 27
    sequence, 27
XSearch, 28
XSLT, 28
XSS, 105, 107

Zend, 104

à balises, 20
échappement, 20, 21
événementielle, 58, 117

Table des figures

1.1 Modèle client-serveur du web 2
1.2 Modèle OSI à 7 couches 6
1.3 Modèle TCP/IP – Internet en 5 couches 7
1.4 Le triangle REST 10
1.5 Le protocole REST 11
1.6 Modèle 3-tier classique pour la répartition de charge 15
1.7 Exemple de fichier au format LaTeX 21
1.8 Expressions régulières dans une DTD 22
1.9 Extraits de la DTD SGML pour l'HTML4.01 strict 23
1.10 Entités prédéfinies en XML 27

2.1 Exemple d'un formulaire plus complet 50

3.1 Priorité des opérateurs en PHP 68
3.2 Variables prédéfinies en PHP 73
3.3 Portée locale de fonctions 74
3.4 Constantes magiques en PHP 74

5.1 Flot de validation et d'échappement d'une application Internet typique 108

Table des matières

Sommaire iii

1 Le modèle web **1**
1.1 Le modèle du WWW . . . 1
1.1.1 Internet et WWW . . . 1
1.1.2 Le modèle client-serveur . . . 2
1.1.3 Les documents hypermédia . . . 3
1.1.4 URL : Uniform Resource Locator . . . 3
1.1.5 Passer de l'information . . . 4
1.1.6 Le protocole HTTP . . . 5
1.1.6.1 Introduction . . . 5
1.1.6.2 Fonctionnement client-serveur : PULL . . . 5
1.1.6.3 Modèle OSI à 7 couches . . . 6
1.1.6.4 Historique et versions . . . 7
1.1.6.5 Phases . . . 7
1.1.6.6 Formats et exemples . . . 8
1.1.6.7 Status et codes d'erreurs . . . 9
1.1.6.8 Méthodes (ou verbes HTTP) . . . 9
1.1.6.9 Redirections sémantiques . . . 10
1.1.6.10 Méthodes étendues . . . 10
1.1.6.10.1 WebDAV . . . 10
1.1.6.10.2 REST . . . 10
1.1.6.11 Types MIME, jeux de caractères et encodages de transfert . . . 11
1.1.6.12 Formats de données du web . . . 12
1.1.6.13 Authentification par le serveur web . . . 12
1.1.6.14 HTTPS : Chiffrement SSL/TLS . . . 12
1.1.6.15 Avenir d'HTTP . . . 13
1.1.6.15.1 Evolution . . . 13
1.1.6.15.2 HTTP/2.0 – Google SPDY . . . 13
1.1.6.15.3 HTTP/3.0 – Google SPDY over QUIC . . . 13
1.1.7 Vers le modèle PUSH . . . 13
1.1.7.1 Besoins . . . 13
1.1.7.2 Implémentations actuelles . . . 14
1.1.7.3 Evolutions . . . 14
1.2 Modèles de conception et de déploiement logiciels . . . 15
1.2.1 Le modèle de conception et de déploiement 3-tier . . . 15
1.2.2 Le modèle de conception et de développement (pattern) MVC . . . 16
1.2.3 Autres modèles . . . 16
1.2.3.1 Architecture logicielle de type gestion . . . 16
1.2.3.2 L'architecture Web service à 5 couches . . . 16
1.3 Mise en production d'une application web . . . 16
1.3.1 Approche non hébergée : exploitation en interne . . . 16
1.3.2 Approche hébergée . . . 17
1.3.3 L'hébergement . . . 17

1.3.3.1 L'hébergeur 17
1.3.3.2 Types d'hébergement 18
1.3.3.3 Contrats de service 18
1.3.4 Les clouds 19
1.4 Les langages à balises 20
1.4.1 Introduction 20
1.4.2 Standard Generalized Markup Language (SGML) 21
1.4.2.1 Introduction 21
1.4.2.2 DTD 21
1.4.2.3 Exemple d'une application de SGML : l'HTML 22
1.4.3 XML 24
1.4.3.1 Introduction 24
1.4.3.2 Schémas 24
1.4.3.3 Jeu de caractères et entités 27
1.4.3.4 Cas d'utilisation 27
1.4.3.5 Traitements 28
1.4.3.6 SaX et DOM 28
1.4.3.7 Transformation avec XSLT 28
1.4.3.8 Outils 29

2 Développement côté client 31
2.1 HTML 32
2.1.1 Introduction 32
2.1.2 HTML, XHTML, HTML5 32
2.1.3 Conformité et compatibilité 33
2.1.4 Séparation entre contenu et mise en forme 33
2.1.5 HTML en pratique 33
2.1.6 Les étiquettes (tags) et les éléments 33
2.1.7 Les entités 34
2.1.8 Les commentaires 35
2.1.9 Création d'un document, déclaration XML et DTD 35
2.1.10 Eléments, balises, attributs 36
2.1.11 Titre du document et entête 36
2.1.12 Corps du document, titres et paragraphes 37
2.1.13 Types d'éléments du corps HTML 37
2.1.13.1 Introduction 37
2.1.13.2 Eléments de type bloc 38
2.1.13.3 Eléments de type en-ligne (inline) 38
2.1.14 Faire des liens 38
2.1.15 Insérer des images dans vos documents 39
2.1.16 Les listes 40
2.1.17 Tableaux 40
2.1.17.1 Tableaux simples 40
2.1.17.2 Tableaux avec cellules recouvrantes 41
2.1.17.3 Titre de tableau 42
2.1.18 Sauts de ligne et espace insécable 42
2.1.19 Les formulaires 43
2.1.19.1 Définir un formulaire simple 43
2.1.19.2 Structuration de la page 43
2.1.19.3 Champs d'entrées de données 43
2.1.19.4 Description du champ 44
2.1.19.5 Désignation des champs (paramètres) 44
2.1.19.6 Grouper les champs 45
2.1.19.7 Les autres types de champs de formulaires 45

2.1.19.7.1 Zone de texte courte . . . 46
2.1.19.7.2 Saisie de mots de passe . . . 46
2.1.19.7.3 Séries d'options à cocher . . . 46
2.1.19.7.4 Listes de sélection . . . 47
2.1.19.7.5 Zones de saisie étendue . . . 48
2.1.19.7.6 Envoi de fichiers . . . 48
2.1.19.7.7 Champs cachés . . . 48
2.1.19.8 Cible et méthode d'envoi du formulaire . . . 48
2.1.19.9 Envoi et traitement du formulaire . . . 49
2.1.19.10 Encodage de transfert . . . 49
2.1.19.11 Notions d'accessibilité et d'utilisabilité . . . 49
2.1.19.12 Accès rapide par raccourcis clavier et ordre de parcours . . . 51
2.1.19.13 Recommandations d'utilisabilité (UX) . . . 52
2.2 Feuilles de style (CSS) . . . 52
2.2.1 Introduction . . . 52
2.2.2 Association de feuilles de style à un document . . . 52
2.2.2.1 Feuille de style externe dans un fichier séparé . . . 52
2.2.2.2 Feuille de style incluse dans le document . . . 53
2.2.2.3 Style propre à un élément seulement . . . 53
2.2.3 Appliquer un style à des éléments : les sélecteurs . . . 54
2.2.3.1 Sélection d'éléments . . . 54
2.2.3.2 Sélections de classes . . . 54
2.2.3.3 Sélection d'un élément unique identifié . . . 55
2.2.3.4 Autres sélecteurs . . . 56
2.2.4 Priorité des spécifications CSS : spécificité, héritage et cascade . . . 56
2.2.5 Indépendance du média et du terminal . . . 56
2.2.5.1 Media Types . . . 56
2.2.5.2 Responsive Web Design . . . 57
2.3 Javascript . . . 58
2.4 Autres langages côté client . . . 59

3 Développement côté serveur 61
3.1 Introduction . . . 61
3.1.1 Avantages du développement côté serveur . . . 61
3.1.2 Principes de fonctionnement . . . 62
3.1.3 Langages, environnements et frameworks côté serveur . . . 62
3.2 Le langage PHP . . . 63
3.2.1 PHP en bref . . . 63
3.2.1.1 Introduction . . . 63
3.2.1.2 Forces et faiblesses . . . 63
3.2.1.3 Historique . . . 64
3.2.2 Principe de fonctionnement . . . 64
3.2.2.1 Balises HTML . . . 65
3.2.2.2 Exemple . . . 65
3.2.3 Environnement de travail . . . 66
3.2.3.1 Principes . . . 66
3.2.3.2 Logiciels et packagings . . . 66
3.2.3.3 Editeur . . . 66
3.2.4 Syntaxe de PHP . . . 67
3.2.5 Opérateurs . . . 67
3.2.6 Erreurs . . . 67
3.2.7 Structures de contrôle . . . 67
3.2.7.1 Alternatives, branchements . . . 69
3.2.7.2 Boucles . . . 69

3.2.7.3 Syntaxe alternative pour les boucles . . . 70
3.2.7.4 Autres structures de contrôle . . . 71
3.2.8 Variables . . . 71
3.2.8.1 Types en PHP . . . 71
3.2.8.2 Déterminer le type d'une variable . . . 72
3.2.8.3 Variables prédéfinies (super-globales) . . . 73
3.2.8.4 Portée des variables . . . 73
3.2.9 Constantes . . . 74
3.2.9.1 Constantes magiques . . . 74
3.2.10 Chaînes de caractères . . . 75
3.2.10.1 Introduction . . . 75
3.2.10.2 Interpolation (expansion) . . . 75
3.2.10.3 Fonctions et opérateurs pratiques sur les chaînes . . . 76
3.2.11 Tableaux et tableaux associatifs . . . 77
3.2.11.1 Définitions . . . 77
3.2.11.2 Affectation . . . 77
3.2.11.3 Fonctions et opérateurs pratiques sur les tableaux . . . 79
3.2.12 Fonctions . . . 80
3.2.12.1 Fonctions natives . . . 80
3.2.12.2 Définition de fonctions . . . 80
3.2.12.3 Fonctions anonymes et closures . . . 81
3.2.13 Namespaces . . . 81
3.2.14 Orientation objet . . . 81
3.2.14.1 PHP comme langage orienté-objet . . . 81
3.2.14.2 Comparaison avec d'autres langages . . . 82
3.2.15 Exemples courants du web . . . 82
3.2.15.1 Utilisation de formulaires . . . 82
3.2.15.2 Images maps . . . 83
3.2.15.3 Cookies . . . 84
3.2.15.4 Sessions . . . 85
3.2.15.5 Envoi de fichiers . . . 86
3.2.15.6 Manipulation d'images . . . 87
3.2.15.6.1 Introduction . . . 87
3.2.15.6.2 Fonctions principales . . . 87
3.2.16 Identificateurs réservés . . . 88
3.2.17 Internationalisation et régionalisation . . . 88
3.2.17.1 Internationalisation . . . 88
3.2.17.2 Régionalisation . . . 89

4 Interfaçage de bases de données (SGBD) 91
4.1 Principes de base . . . 91
4.1.1 Interfaçage, sécurité et ORM . . . 91
4.1.2 Complexité dans l'application et/ou dans la base de données ? . . . 92
4.1.3 Types de base de données particulières . . . 92
4.2 Interfaçage entre PHP et MySQL . . . 93
4.2.1 Accès à une base MySQL . . . 93
4.2.1.1 Pour l'administration et la maintenance . . . 93
4.2.1.2 Pour l'accès aux données par l'application . . . 93
4.2.2 Un exemple avec PHP, PDO et MySQL . . . 94
4.2.2.1 Introduction . . . 94
4.2.2.2 Création de la BD . . . 94
4.2.2.3 Se connecter à la base de données . . . 95
4.2.2.4 Requête préparée . . . 95
4.2.2.5 Exécution d'une requête préparée . . . 96

4.2.2.6 Exécution directe d'une requête . . . 96
4.2.2.7 Configurer le retour des résultats . . . 96
4.2.2.8 Parcourir les résultats . . . 96
4.2.2.9 Traitement d'erreurs . . . 97
4.2.2.10 Fermeture . . . 97
4.2.2.11 Exemple complet . . . 97
4.2.3 Exemples avec d'autres API . . . 98
4.3 Assurer la sécurité, la performance et éviter le sur-codage . . . 102
4.4 Recommandations . . . 103
4.5 Vers les frameworks . . . 104

5 Sécurité des applications web 105
5.1 La problématique . . . 106
5.2 Types d'attaques . . . 106
5.2.1 Injection . . . 106
5.2.2 Cross-scripting / XSS . . . 107
5.2.3 Intégration de contenu / click-jacking . . . 107
5.2.4 Social engineering . . . 107
5.3 Méthodes de protection . . . 107
5.3.1 Introduction . . . 107
5.3.2 Validation . . . 108
5.3.3 Echappement . . . 109
5.3.4 Confinement . . . 110
5.3.5 Tainting (coloriage) . . . 110
5.3.6 Signalisation hors-bande ou en-bande . . . 111
5.3.6.1 Problématique . . . 111
5.3.6.2 Binding . . . 111
5.3.6.3 Formats stricts . . . 112
5.3.7 CAPTCHA . . . 112
5.3.8 Chiffrement . . . 112
5.3.9 Recommandation pour l'authentification . . . 112
5.3.10 Recommandations PHP . . . 113

Références et bibliographie 115

Lexique 117

Index des concepts 119

Table des figures 125

Table des matières 127

ISBN 978-2-940387-08-3
9 782940 387083