R209 - Initiation au Développement Web

BUT1 - R&T

2022

Table des matières

Introduction

Objectifs

Le cours “Initiation au développement web” fait suite au cours “Introduction aux tehnologies web” (R109). Il présuppose donc de maîtriser la structure des pages web en HTML+CSS et a pour objectif d’aller plus loin, c’est-à-dire vers de la programmation web.

Formellement, les objectifs sont :

— savoir développer des applications web,
— développer des applications client/serveur,
— acquérir des notions de langages de description.

Ce module est également l’occasion d’approfondir le module “Bases de la programmation” (R107) d’une part en appliquant dans le cadre du web les notions algorithmiques qui y ont été vues ; d’autre part en découvrant de nouvelles notions d’algorithmique et programmation.

Rappels de Web statique

Une page web est structurée à l’aide de balises (xml). Les balises forment une arborescence (similaire à l’arborescence des fichiers en R108 (bases des systèmes d’exploitation) ou aux arbres d’évaluations en R107). Un certain nombre de règles limitent les imbrications de balises (par exemple, on ne peut pas mettre un paragraphe `<p></p>` à l’intérieur d’un paragraphe `<p>...</p>`). Parmi les éléments importants sont apparus un entête (`head`) et un corps (`body`). L’entête donnant des directives globales comme l’encodage des caractères, le titre, des mots-clefs et aussi les emplacements des feuilles de style CSS à utliser.

Jusqu’ici, les pages web créées sont statiques (modulo responsivity et vidéos). Pourtant, sur le web, la plupart des sites sont dynamiques (possibilités de noter, commenter des articles), voire sont des applications qui sont exécutées par le navigateur (un webmail par exemple remplace une application “normale” de messagerie à la thunderbird tournant en dehors d’un navigateur). Pour passer à des applications web, il va falloir écrire des programmes, ce qui va se faire dans un nouveau langage de programmation, en effet

HTML et CSS ne sont pas des langages de programmation (par exemple on ne peut pas faire de boucles avec) mais des langages de description.

Rappels de bases de la programmation

Un programme s'écrit dans un langage de programmation donné pour être interprété, compilé ou exécuté par l'ordinateur. Il est la traduction d'un algorithme (qui présente l'idée du déroulement de façon lisible, compréhensible par un humain). Le programme est composé d'une suite d'instructions (actions) et de structures de contrôles qui permettent de faire des choix ou des répétitions et manipule des données sous forme de variables ou interagit avec l'utilisateur (lire/écrire).

Nous avons dans le module R107 utilisé le langage `java` pour formaliser nos algorithmes. Ce langage n'est malheureusement pas utilisé pour la programmation web, c'est donc dans d'autres langages que nous traduirons nos algorithmes, mais les notions de base sont réutilisables sans problème.

Cas d'utilisation

Penchons-nous maintenant sur trois cas d'utilisation, c'est-à-dire trois exemples classiques d'applications web et tentons d'agréger leurs points communs et de repérer leurs différences.

Nos trois exemples sont :

a. un webmail (par exemple zimbra/etumail),
b. un site de partage de vidéos (à la vimeo),
c. un forum.

Le premier cas est une application complète qui tourne dans le navigateur (et remplace une application native), les autres restent des pages web mais s'adaptant à l'utilisateur. Par exemple pour le partage de vidéos, le site présente une page adaptée au navigateur (choix entre flash et HTML5, choix entre h264 et webm pour le codec vidéo...) et une vidéo adaptée au matériel (résolution et codec spécifiques pour mobile/desktop).

Dans le cas du forum, on a besoin de plusieurs composants : d'une part une identification de l'utilisateur, d'autre part la possibilité d'intéragir avec la base de données des messages et enfin la génération de pages pour présenter les discussions par tranches de 20 messages.

Dans le cas du site de partage de vidéo les composants sont essentiellement la reconnaissance du navigateur (y compris le matériel sur lequel il tourne, la taille de la fenêtre), la gestion des événements utilisateurs (avancer dans la vidéo au temps indiqué par la souris, modification du volume...) et la présentation adaptée, y compris pour le codec à utiliser.

Le cas du webmail présente à la fois des propriétés du forum (identification, accès à une base de données centralisée) et des propriétés du site de partage de vidéos (page web adaptée au lecteur, animation pour avoir une interface réactive).

Client et serveur

Notions de client et de serveur

Nous avons dans l'introduction parlé d'exécution côté client et d'exécution côté serveur. Que signifient ces notions, qu'est-ce qu'un client, qu'est-ce qu'un serveur ?

Les applications réseaux sont souvent basés sur une architecture mettant en évidence d'un côté un serveur, de l'autre des clients. Dans cette architecture, les clients se connectent au serveur pour lui envoyer des requêtes, le serveur traite ces requêtes et envoie les résultats aux clients. Dans le cas du web, le serveur est le serveur web (httpd), le client est le navigateur. Ainsi, quand vous visitez une page web (http://arche.univ-lorraine.fr/ par exemple), votre navigateur contacte le serveur (ici une machine repérée par l'adresse `arche.univ-lorraine.fr`) suivant un protocole (ici `http`) et lui demande une page web en particulier (ici '/'). Le serveur reçoit cette requête, génère la page web en question et envoie le code html à votre navigateur pour que celui-ci l'interprète et l'affiche. Cette architecture n'impose pas que le client et le serveur soient sur des machines différentes, ainsi dans le monde UNIX, l'interface graphique fonctionne également suivant le modèle client/serveur : les fenêtre d'applications sont les clients, le programme les affichant sur l'écran est le serveur graphique (`Xorg` ou `wayland` sous Linux, `Quartz` sous Mac OS X, `X11` sous openBSD...). L'architecture client/serveur s'oppose au modèle pair à pair (P2P) où tous les membres sont de même niveau.

Le concepteur d'une page web (ou d'une application web) peut choisir que le programme générant les parties dynamiques soit exécuté chez le client (donc par le navigateur) ou par le serveur. Suivant les caractéristiques de l'application à réaliser, le choix côté client et le choix côté serveur peuvent être plus ou moins judicieux. Il est aussi possible d'avoir des parties exécutées côté serveur et des parties exécutées côté client dans une même application (c'est souvent le cas dans les applications complexes).

Les inconvénients de l'exécution côté serveur sont :

— Nécessité d'avoir un serveur (!)
— Plus de charge de travail pour le serveur (donc il pourra servir moins de clients efficacement).
— Toute action du client doit être transmise au serveur pour avoir une répercussion (il est donc difficilement imaginable d'avoir une interface graphique réactive).
— Des interventions clients agissent sur le serveur, il faut prendre des précautions, veiller à la sécurité (DDOS, injection SQL..).

Les inconvénients de l'exécution côté client sont :

— Le navigateur client peut ne pas être compatible avec le code à exécuter (les navigateurs firefox, chrome, internet explorer, opera ne comprennent et n'interprètent pas tout de la même façon. Les anciennes versions supportent moins de choses).
— L'utilisateur a la main sur le code et le navigateur. Il peut donc faire faire des choses non prévues à l'application. Par exemple, on ne doit pas vérifier un mot de passe côté client.

Dans diverses situations, on choisira donc plutôt une exécution côté client ou côté serveur (ou mixte). Par exemple,

— pour réaliser un menu qui apparaît quand la souris survole un mot-clef, on choisira côté client (en fait du css suffit),
— pour un forum où les utilisateurs postent des messages, on choisira côté serveur (stockage des messages sur le serveur),
— pour un jeu vidéo, une animation, côté client,
— pour les meilleurs scores d'un jeu vidéo, côté serveur,
— pour une carte dynamique, côté client.

Pour les cas d'utilisation présentés plus haut, le forum devra être exécuté côté serveur, la visionneuse de vidéos côté client et le webmail aura des composants côté client et côté serveur.

Technologies pour l'exécution côté client et l'exécution côté serveur

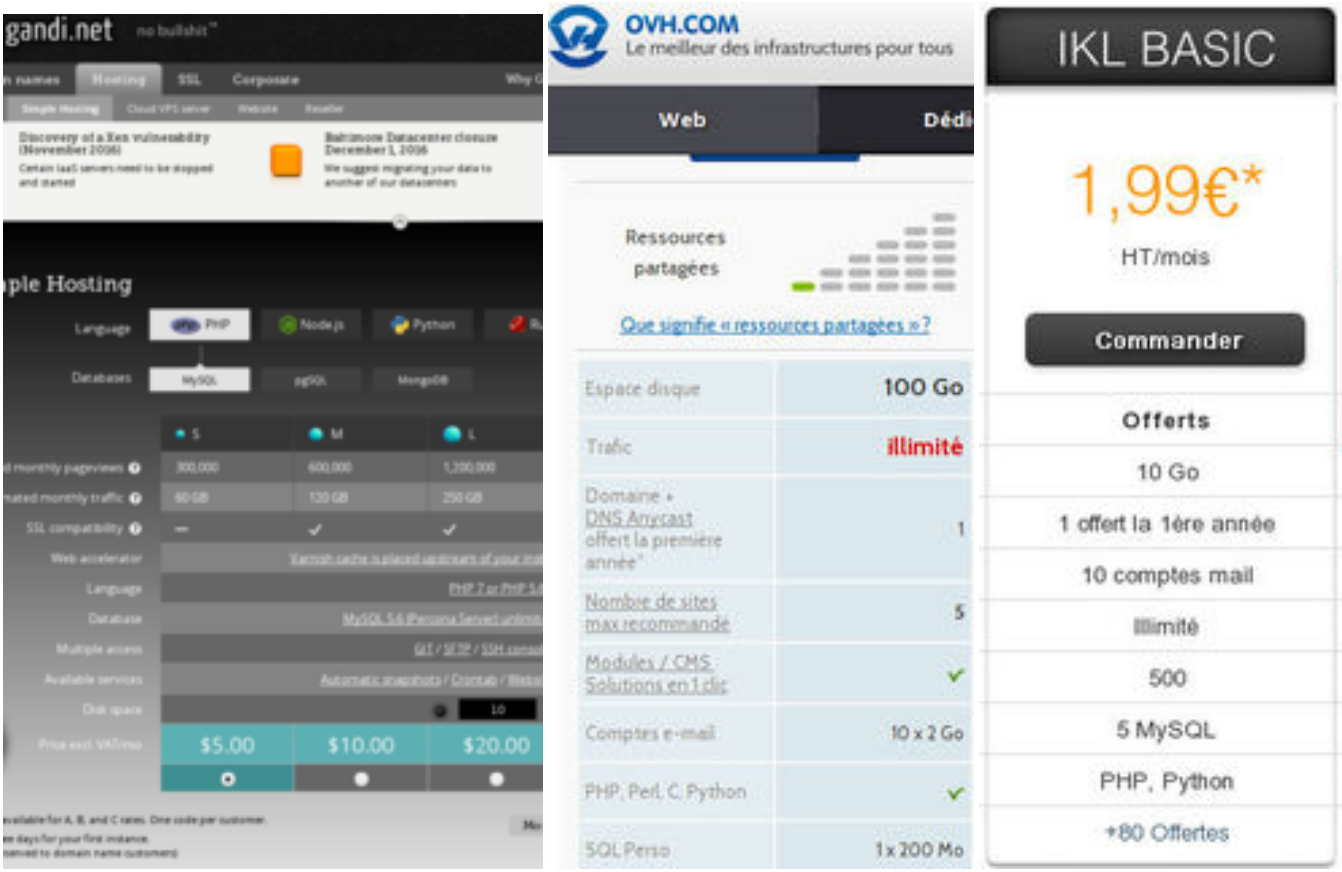
Technologies côté serveur

Quelles technologies (en fait quels langages de programmation) pouvons-nous utiliser pour une exécution côté serveur ?

A priori, puisqu'en tant que développeur web, nous maîtrisons le serveur, n'importe quel langage peut faire l'affaire : il suffit de convaincre le serveur web d'appeler le bon interpréteur pour générer la page web à transmettre au client.

En pratique, les serveurs web (apache, nginx, lighttpd, iis, bozohttpd...) utilisent de préférence quelques langages précis : php, perl, python, ruby ou java. Donc on se contente en général de ceux-ci. De plus, le développeur ne maîtrise pas nécessairement les serveur : il peut décider de faire héberger son application par un hébergeur n'acceptant que certains de ces langages. Ce qui fait que le premier langage qui a été utilisé de façon massive pour faire des pages web dynamiques côté serveur reste le langage le plus utilisé pour cela. Il s'agit de php (initialement pour *Personal Home Page*, aujourd'hui *PHP: Hypertext Preprocessor*).

Par exemple, voici les offres d'hébergement de 3 entreprises (Gandi, Ikoula et OVH) :



On voit les langages de programmation proposés par chacun (techniquement, node.js n'est pas un langage...), ce qui restreint les choix possibles.

Technologies côté client

Pour l'exécution côté client, nous sommes restreints par les navigateurs avec lesquels les utilisateurs accéderont à notre application web. Plusieurs technologies prétendent être disponible (flash, NaCl, Dart, javascript) mais une seule est réellement omniprésente : javascript (si on veut être précis, on doit appeler le langage ECMAScript).

javascript est un langage de programmation multi-paradigmes mais néanmoins simple développé à partir de 1995 chez Netscape (l'ancêtre de mozilla). Il a tout de suite été conçu pour être exécuté par les navigateurs web pour animer les sites web. Tout comme *Python* ou *Lua*, il s'agit d'un langage de script impératif. Il est aussi fonctionnel (nous verrons un peu ce que cela signifie) et objet (le paradigme objet sera étudié dans le module R207 "Sources de Données" et R307). Comme *Python* et *Lua* encore, il est interprété. Ce qui signifie que l'on peut taper les instructions une à une dans un interpréteur et voir leur résultat aussitôt.

Depuis 1997, javascript est standardisé (un organisme international, ecma, a validé et publié sa spécification). La version standard est nommée ECMAScript (nom qui permettrait d'éviter la confusion avec java si ce nom était utilisé de façon courante).

Contrairement à Java, le typage est implicite (il est inutile de préciser les types des variables dans le code, l'interpréteur les inférera automatiquement). Contrairement à Java, il y a un unique type numérique (donc la division est *toujours* la division réelle). Comme en Java (et contrairement à Lua), l'opérateur `+` est surchargé puisqu'il sert à la fois à l'addition et à la concaténation de chaînes. Les bases de la syntaxe peuvent être vues dans syntaxe.js. Précisons seulement la syntaxe des itérations bornées :

En lua, nous écrivons

```lua
for i=m, M do
    print(i)
    -- actions
end
```

En java, nous écrivions

```java
for (int i=m; i <= M; i=i+1) {
    System.out.println(i);
    // actions
}
```

En javascript, nous écrirons

```javascript
for (var i=m; i <= M; i=i+1) {
    alert(i);
    // actions
}
```

Les instructions d'initialisation de la boucle et d'itération, ainsi que la condition qui apparaissaient dans l'ordinogramme sont ici explicites !

Algorithmique : fonctions et procédures

Un des buts de l'algorithme est de faire faire le plus de travail possible à l'ordinateur plutôt qu'à l'humain. Il est donc intéressant de ne pas recopier du code en plusieurs endroits d'un programme/algorithme.

Pour cela, nous pouvons regrouper les suites d'instructions communes sous forme d'un sous-programme. Nous chercherons cependant à ce que ces sous-programmes aient un but clair, réalisent une tâche simple. Il s'agira donc de faire une division logique de l'algorithme en composants. C'est le principe du "diviser pour mieux régner" : transformer un problème complexe en plusieurs sous-problèmes simples

Procédures

Une procédure est un sous-programme simple réutilisable. Comme un algorithme, une procédure a un nom, un rôle et des déclarations.

Les procédures sont définies à l'aide d'un mot-clef : `def` (define) en python, `function` en lua ou javascript :

Python

```python
def p():
    #role : ...
    #decl : i: entier
    instructions
```

Java

```java
static void p () {
    // Role: faire des choses
    // Variables : x, y, z : entiers
    Instructions
}
```

Javascript

```javascript
function p() {
    //role : ...
    //decl : i: entier
    instructions
}
```

Pour utiliser une procédure, on l'appelle par son nom suivi de parenthèses : `p()`. Une procédure devient une instruction.

```java
public class Main {
    public static void effaceTerminal() {
        System.out.print("\033[H\033[2J");
        System.out.flush();
    }

    public static void quitter() {
        System.out.println("Voulez-vous quitter ?");
        Scanner sc = new Scanner(System.in);
        String reponse = sc.nextLine();
        if (reponse.toLowerCase().compareTo("o") == 0) {
            effaceTerminal();
            System.exit(0);
```

```
        }
    }

    public static void main(String[] args) {
        effaceTerminal();
        quitter();
        while (true) {
            quitter();
        }
    }
}
```

Fonctions

Les procédures telles que nous venons de les définir ont un intérêt limité :

- pas d'entrées/sorties ;
- ne peut donc faire qu'une seule chose.

En ajoutant des entrées/sorties, on peut définir des fonctions (comme en mathématiques)

- $f : \left\{ \begin{array}{ccc} \mathbb{R} \times \mathbb{N} & \to & \mathbb{R} \\ (x, n) & \mapsto & x^n \end{array} \right.$
- x et n sont les entrées (de types respectifs réel et naturel).
- la sortie est définie comme valant x^n
- on utilise la fonction par $f(\pi, 2)$

Une fonction se comporte exactement comme un algorithme :

- elle comporte un entête contenant
  - son nom
  - son rôle
  - ses entrées (assorties de préconditions)
  - ses sorties (assorties de postconditions)
  - ses déclarations
- et un corps contenant les *statements* définissant le corps de la fonction.

Définition d'une fonction nommée `f` prenant deux arguments

Java

```
static type f(type1 a1, type2 a2) {
    // role : ...
    // entrees : a1 : type1, a2 : type2
    // sorties : s : type
```

```
    // declarations : ...
    type s;
    Instructions
    return s;
}
```

Javascript

```
function f(arg1, arg2) {
    // role : ...
    // entrees : arg1 : type1, arg2 : type2
    // sorties : s : type
    // declarations : ...
    var s = ...
    ...
    return s
}
```

Utilisation de cette fonction :

- `v = f(1, 2)`
- `v = f(x, y)`
- `v = f(3*7-1, y)`

Une fonction devient une expression.

Appelé et Appelant

Lors de l'exécution d'un programme, l'appel à une fonction ou une procédure introduit un changement de contexte.

```
static double puissance (double x, int n)
    // entrées : x : réel, n : entier
    // sortie : y : réel
    double y = 1;
    for (int i=0; i < n; ++i) {
        y = y*x
    }
    return y;
}

public static void main(String[] args) {
    double x = 1.4;
    double carre = puissance(x, 2);
    double cube = puissance(x, 3);
```

```
    System.out.println(carre);
    System.out.println(cube);
    System.out.println(puissance(x, 4));
    System.out.println(puissance(math.pi, 7));
}
```

Dans l'exemple précédent, la ligne `double carre = puissance(x, 2)` appelle la fonction puissance.

- `puissance` est la *fonction appelée*.
- `main` est l'*appelant*.
- Une fonction peut en appeler une autre (ex : une fonction de calcul de n^n appellerait la fonction puissance).

La fonction appelée est exécutée avec ses propres données et arguments. La fonction appelante ne connaît de la fonction exécutée que ce qu'elle retourne.

Portée des variables

Les fonctions et procédures ont leurs propres déclarations.

Ces variables n'existent que pendant l'exécution de la méthode. Ces variables peuvent masquer une variable de l'algorithme ou de la fonction appelante.

> **Définition**
> Contexte ou environnement d'exécution : l'ensemble des variables connues à l'intérieur de la fonction (les déclarations, entrées et sorties)

Procédures

Une procédure est en fait une fonction sans sortie (c'est-à-dire sans `return`).

- Une procédure peut avoir des arguments.
- Une procédure est en réalité une instruction.
- Exemples : `System.out.println("Bonjour")`, `Thread.sleep(1)`.

Fonctions

- Une fonction est en réalité une expression.
- Sa valeur et son type sont la valeur et le type de la variable de sortie.
- Le statement `return` doit renvoyer une expression du bon type.
- Exemple : `scanner.nextInt()` est une fonction prenant 0 argument qui renvoie un entier.

Quelle est la nature de `puissance(2+3, 7) + puissance(2*3, 5) + puissance(puissance(2, 3), 3)` ?

Exercice : Réaliser son arbre d'évaluation.

Dans la définition d'une fonction, les arguments sont dénommés des paramètres formels. Par exemple, dans la déclaration `function f(x, y)`, les variables `x` et `y` sont les paramètres formels.

- Lors de l'appel de la fonction, elle est appelée avec des paramètres effectifs : les valeurs sur lesquelles la fonction travaillera.
- Les paramètres formels peuvent "cacher" une variable de l'algorithme.
- Lors de l'exécution de la fonction, les paramètres formels sont remplacés par la valeur du paramètre effectif sur lequel la fonction est appelée.

Passage des arguments par valeur, par copie, par référence

Transmission des paramètres :

- Par valeur (ou par copie) : le paramètre est transmis sous forme de copie, le modifier dans la fonction ne modifie pas le paramètre effectif.
- Par référence : le paramètre est transmis sous forme de pointeur (adresse mémoire). La fonction/procédure peut donc modifier le paramètre effectif (effets de bord).

Dans la plupart des langages (mais pas tous), les entiers sont passés par valeur, les tableaux et tous les objets de type complexe par référence. Nous avons déjà constaté ceci en TP de R107 dans un exercice sur les tableaux où nous avions implémenté le clonage de tableaux.

Algorithmique : tableaux associatifs

Les tableaux sont une structure de données permettant de stocker plusieurs éléments dans une seule variable. Cela permet par exemple de stocker toutes les notes obtenues par les étudiants à une épreuve donnée dans une structure plutôt que d'avoir autant de variables qu'il y a d'étudiants. Les cases d'un tableau sont indexés par des entiers naturels (notons qu'en javascript, les cases des tableaux sont numérotées à partir de 0).

Nombreux sont les usages où l'on voudrait indexer les cases par autre chose que des entiers. Par exemple, dans l'exemple des notes à une épreuve, nous aimerions associer la note au nom de l'étudiant. Pour cela, on utilise une nouvelle structure de données : les tableaux associatifs (ou dictionnaires) qui, comme leur nom l'indique, associe à un identifiant une valeur (ou à un mot leur définition). On peut avoir des tableaux associatifs de n'importe quelle paire de type, mais l'usage est le plus souvent d'avoir

comme identifiants des entiers (non nécessairement consécutifs comme pour les tableaux classiques) ou des chaînes de caractères.

On peut par exemple initialiser un tableau associatif comme suit :

```
// Java
Map<String, String> dico = new HashMap<>();
dico.put("r107", "Bases de la programmation");
dico.put("r108", "Bases des systèmes d'exploitation");
dico.put("r109", "Introduction aux technologie Web");
System.out.println(dico.get("r108");
```

La syntaxe javascript sera plus agréable :

```
// Javascript
dico = {
    "r107" : "Bases de la programmation",
    "r108" : "Bases des systèmes d'exploitation",
    "r109" : "Introduction aux technologies web"
}
```

Ce tableau, nommé `dico` associe des chaînes de caractères (les intitulés de modules du programme Réseaux et Télécoms) à des chaînes de caractères (les codes des modules). On peut ensuite accéder à une case du tableau, modifier une case comme avec un tableau classique. Créer une nouvelle case est simplement une affectation :

```
alert(dico["r107"])
dico["r107"] = "Algorithmique et programmation"
dico["r209"] = "Initiation au développement web"
```

Nous ne pouvons plus parcourir les tableaux associatifs comme nous parcourions les tableaux, nous avons besoin d'énumérer les clefs. Pour cela, on utilise toujours la construction `for` mais avec une syntaxe particulière. En java, l'élément de syntaxe additionnel est un `:`.

Java

Pour chaque entrée `m` de `dico`

```
for (Map.Entry m: dico.entrySet()) {
    System.out.println("En " +
        m.getKey() +
        " se trouve la valeur " +
        m.getValue());
}
```

Javascript

Pour chaque clef ***k*** *dans* ***dico***

```
for (var k in dico) {
    print("en " +
        k +
        " se trouve la valeur " +
        dico[k])
}
```

Dans ce bout de code, la variable `k` va parcourir l'ensemble des "clefs" qui apparaissent dans le tableau (ici r107, r108, r109 et r209) et le contenu de la boucle sera exécuté pour chacune de ces valeurs.

Les valeurs dans un tableau associatifs peuvent ne pas être des types simples mais des tableaux (éventuellement associatifs). Par exemple, le tableau qui suit montre un tableau associatif dont les clefs sont des noms d'équipes de footbal et les valeurs sont des tableaux associatifs contenant les données du classement de cette équipe. Nous le manipulons ensuite pour afficher le classement actuel puis ajouter le classement après une autre journée de championnat.

```
classement = { "Paris Saint-Germain FC": { "position": 1, "playedGames": 25, "won": 18, "
  "Olympique de Marseille": { "position": 2, "playedGames": 25, "won": 13, "draw": 7, "lo
  "OGC Nice": { "position": 3, "playedGames": 25, "won": 14, "draw": 4, "lost": 7, "point
  "RC Strasbourg Alsace": { "position": 4, "playedGames": 25, "won": 12, "draw": 6, "lost
  "Stade Rennais FC 1901": { "position": 5, "playedGames": 25, "won": 12, "draw": 4, "los
  "AS Monaco FC": { "position": 6, "playedGames": 25, "won": 10, "draw": 8, "lost": 7, "p

// Affichage du classement de 1 à 20
for (i=1; i<=20; ++i) {
    for (equipe in classements) {
        if (classement[equipe]["position"] == i) {
            print(str(i) + ": " + equipe)
}}}

// Mise à jour après la victoire de Rennes contre Montpellier :
classement["Stade Rennais FC 1901"]["won"]++
classement["Stade Rennais FC 1901"]["playedGames"]++
classement["Stade Rennais FC 1901"]["goalsFor"] += 4
classement["Stade Rennais FC 1901"]["goalsAgainst"] += 2
classement["Montpellier HSC"]["lost"]++
classement["Montpellier HSC"]["playedGames"]++
classement["Montpellier HSC"]["goalsFor"] += 2
classement["Montpellier HSC"]["goalsAgainst"] += 4
```

HTML : formulaires

Pour entrer des valeurs à nos programmes, nous allons utiliser une nouvelle balise html : `form`. Un formulaire est donc un conteneur dans lequel on peut mettre des boîtes de texte, des boutons (y compris boutons radio et cases à cocher). Ces dispositifs d'entrée seront matérialisés par des balises `input`. On peut ensuite lier ces entrées à des fonctions, par exemple, à l'appui sur le bouton `go` associer l'exécution d'une procédure javascript `go()` ou l'envoi au serveur de l'événement pour qu'il exécute le script php avec les nouvelles valeurs d'entrée.

Exemple de formulaire :

```
<form>
    <input type="text" name="ville" /><br />
    <input type="checkbox" name="couleurR" value="rouge" />Rouge
    <input type="checkbox" name="couleurV" value="vert" />Vert
    <input type="checkbox" name="couleurB" value="bleu" />Bleu<br />
    <input type="radio" name="booleen" value="vrai" checked="checked" />Vrai
    <input type="radio" name="booleen" value="faux" />Faux<br />
    <input type="button" name="gobutton" value="aller" />
    <select name="liste">
        <option value="m1105">Bases des systèmes d'exploitation</option>
        <option value="m2104">Bases de données</option>
        <option value="m2105">Web dynamique</option>
    </select>
</form>
```

Si on veut traiter ce formulaire en javascript, on peut ajouter à chaque input (ou seulement à ceux dont la modification doit produire quelque chose) un attribut `onclick` ayant pour valeur la fonction à exécuter. Par exemple :

```
<input type="button" name="ga" value="aller" onclick="go(this.form);" />
<input type="button" name="bu" value="faire" onclick='alert("vous avez cliqué sur le bout
```

Dans la fonction `go`, on peut accéder aux différents input via le paramètre de la fonction, le paramètre étant un tableau associatif ayant chaque nom d'`input` comme clef.

```
go = function(arg) {
    alert(arg["ville"]["value"] +
        "\nRouge : " + arg["couleurR"]["checked"] +
        "\nVert : " + arg["couleurV"]["checked"] +
        "\nBleu : " + arg["couleurB"]["checked"] +
        "\nmodule : " + arg["liste"]["value"]);
}
```

De même en php, on peut décider de mettre sur le formulaire un signal pour le renvoyer et de décider s'il s'agit d'un GET (je veux recevoir de nouvelles informations du serveur en fonction de mes choix ou un POST (pour lequel le serveur peut changer son état interne (par exemple la soumission d'un message dans un forum sera un POST)). Et le programme peut accéder aux valeurs stockées dans le formulaire de façon similaire au cas du javascript.

API : Application Programming Interface et langages de transmission de données

Un aspect intéressant de la programmation web est de récupérer des données provenant d'autres sites web et de la réutiliser dans notre propre application web. Pour cela, nous avons besoin de définir deux choses : d'une part un langage commun dans lequel seront transmises les dites informations, d'autre part les méthodes d'obtention de ces informations et en particulier comment s'utilisent les fonctions de ces autres sites.

Langage standardisé de transmission de données

Pour transmettre les données, il faut un langage structuré présentant les données que nous voulons obtenir sous forme exploitable et compréhensible. On peut penser au html dont c'est le rôle pour des données présentables. De façon un peu plus générale, pour n'importe quel type de données, on peut utiliser xml (eXtensible Markup Language) avec des balises et des attributs adéquats. La formalisation de ce qui sera acceptable peut être définie via des schémas DTD (Document Type Definition), XSD ou XML Schema.

Du côté algorithmique, pour structurer les données, nous sommes habitués à utiliser des tableaux associatifs. Les données arborescentes que nous savons stocker dans du xml peuvent tout à fait être conservées dans des tableaux de tableaux de tableaux... Il est assez standard d'utiliser cette idée pour échanger des données entre site web et on nomme ce format `json` (javascript object notation).

Par exemple si on utilise *ip-api*, service de géolocalisation d'IP, on peut avoir les réponse au format `json` : http://ip-api.com/json/ ou au format `xml` : http://ip-api.com/xml/. On pourra utiliser des outils de visualisation du json pour explorer le json :

- Firefox
- JShon
- Hoppscotch
- ~~JSONEditorOnline~~

Quelques questions se posent quand on utilise des données externes, comme

- doit-on attendre le résultat pour afficher le contenu de la page ?
- que faire si le site distant ne répond pas immédiatement ?

La réponse à la première question est “non”. Mais nous n’avons jamais vu d’algorithmes capables de traiter des événements comme l’attente d’une réponse de façon non bloquante. Nous allons pour cela utiliser deux notions : d’une part les requêtes seront asynchrones, ce qui signifie que l’on lance la requête mais n’en attend pas la réponse pour passer à la suite du programme ; d’autre part, un écouteur d’événement (*event handler*) traitera le résultat quand il sera obtenu. Voir à ce propos le TP 3 avec les `fetch`, les callback et la définition d’un event handler via l’attribut `onload` (de la même façon que dans les formulaires, le `onclick` définit un event handler.

API

En plus d’obtenir des données d’autres sites, on voudra utiliser des fonctions pré-programmées issues d’autres sites. Pour pouvoir les utiliser, il faut savoir ce que fait chaque fonction, connaître les entrées nécessaires, ce que l’on va obtenir en sortie (c’est-à-dire exactement ce que nous mettions dans les en-têtes de nos algorithmes). On appelle ces informations la signature de la fonction et l’ensemble des signatures des fonctions proposées par une bibliothèque l’API (Application Programming Interface).

Par exemple, wikipedia propose une API pour accéder aux images de l’encyclopédie en ligne. L’URL https://en.wikipedia.org/w/api.php?action=query&list=allimages&format=json&aiprefix=nancy va renvoyer un fichier *json* de toutes les images (*allimages*) correspondant à *nancy*. Changer la valeur liée à `aiprefix` permettra d’obtenir les résultats liés à cette valeur. `aiprefix` est donc un argument de la fonction appelée via cette requête. Les différents arguments et le comportement de cette fonction sont documentés. La documentation fait partie intégrante d’une API.

La présentation des données d’une application web via une API accessible à travers de simples requêtes HTTP est quelque chose de très standard et suit des normes plus ou moins écrites telles que REST (Representational State Transfer) ou SOAP (Simple Object Access Protocol).

Sécurité

Comme nous l’avons déjà dit, la programmation côté serveur implique d’exposer un serveur et des données à des clients potentiellement mal intentionnés. Il faut donc se protéger soi même mais aussi protéger les données des utilisateurs sains. Il est par exemple inacceptable que quelqu’un puisse découvrir le mote de passe ou le code de carte bancaire d’un utilisateur en écoutant sa connexion réseau.

Mots de passe

Les mots de passe sont un constituant important du web. Les concepteurs de sites web doivent les manipuler de façon adéquate (vérification côté serveur, transmission par canal sécurisé : https) et les stocker de façon sure (en cas de fuite de la base de données des mots de passe).

Nous verrons en TD les ordres de grandeur en temps du décodage d'un mot de passe faible par rapport à un mot de passe fort, en utilisant un algorithme rapide plutôt qu'un algorithme lent. Nous approcherons également quelques techniques utilisées par les crackers (utilisation d'un botnet pour paralléliser le calcul, création et utilisation de rainbow tables) et quelques procédés pour rendre ces techniques moins efficaces (utilisation d'un sel recours à un second facteur par exemple)

Il faut surtout retenir que le stockage de mots de passe est quelque chose de compliqué et donc utiliser des solutions pré-existantes plutôt que réinventer la roue.

Données utilisateurs

Les programmes côté serveur sont souvent amenés à manipuler des données utilisateurs (login, mot de passe, commentaires...). Si les données sont mal formées, elles peuvent amener le serveur à faire des des bêtises (plantage qui rend indisponible le site, fuite d'informations, effacement de bases de données...). Il ne faut donc pas faire confiance aux données entrées par les utilisateurs. Nous verrons en TD quelques problèmes qui peuvent se produire et réfléchiront à un moyen de les éviter.

Ce genre de problème se pose de façon particulièrement sensible dans le cas où le serveur utilise une base de données. Le problème décrit ci-dessus est alors connu sous le nom d'injection SQL.

Par exemple, imaginons que notre application web comporte un champ de recherche `search`. Une requête SQL typique pour exécuter cette recherche sera de la forme :

```
SELECT * FROM RVente WHERE champ='$search';
```

En remplaçant `$search` par la valeur à chercher, on obtient une requête SQL simple et a priori correcte. Que se passe-t-il si l'utilisateur met `'; DROP TABLE RVente;--` dans le champ `search` ? Le code exécuté sur la base de données sera alors la suite de requêtes suivante :

```
SELECT * FROM RVente WHERE champ='';
DROP TABLE RVente;
--';
```

La troisième ligne est un commentaire (pour ne pas tenir compte d'une éventuelle suite de la requête). La deuxième ligne va supprimer la table `RVente` de la base de données !

https

Enfin, les transferts réseau doivent eux aussi être sécurisés sans quoi le fournisseur d'accès à internet mais aussi les voisins peuvent voir passer les mots de passe.

Le protocole utilisé est alors https : HyperText Transfer Protocol *Secure*.

Il s'agit du protocole HTTP accompagné de chiffrement pour

1. assurer l'identité du site web (indispensable pour une banque par exemple)
2. éviter que les données transférées le soient de façon lisible par n'importe qui

Avec HTTPS, la transmission est chiffrée, ce qui donne des garanties :

— Les voisins (wifi) ne voient plus les contenus
— Le FAI ne voit plus les contenus

La transmission est également signée, ce qui garantit que le site est le bon.

Mais il reste des problèmes possibles :

— Le FAI connaît toujours les serveurs contactés
— Il reste possible de faire une attaque MitM

Un peu de cryptographie

On distingue 2 types de cryptographie : celle à chiffrement symétrique et celle à chiffrement asymétrique.

Avec le chiffrement symétrique, la même clef est utilisée pour chiffrer et déchiffrer (similaire aux clefs dans le monde réels). C'est un mode de chiffrement pratique pour chiffrer une sauvegarde, un coffre-fort de données (la même personne va chiffrer et déchiffrer). Mais pour des communications, il faudrait pouvoir transmettre de façon sûre la clef au destinataire...

Pour cela, le deuxième type de chiffrement, asymétrique, est celui à utiliser. Cette fois, il y a 2 clefs différentes : 1 clef publique et 1 clef secrête. Si l'on chiffre avec la clef publique, seule la clef secrête permet de déchiffrer et inversement.

Quand on chiffre avec la clef publique, il faut la clef privée pour déchiffrer, seul le destinataire peut lire. C'est le mode à utiliser pour envoyer un message chiffré.

Quand on chiffre avec la clef secrête, tout le monde peut déchiffrer (avec la clef publique). Cela nous donne donc une preuve de qui est l'émetteur (le seul possesseur de la clef secrête. C'est ce que l'on veut pour *signer* un message.

HTTPS utilise les deux modes : chiffrement et signature.

Il reste à résoudre la question de l'échange de clef. Comment transmet-on au monde entier la clef publique ?

Pour HTTPS, la solution repose sur des autorités de certifications de confiance connues du système d'exploitation (ou du navigateur). La clef publique (que l'on appelle dans ce cas un certificat SSL) est signée par une autorité de certification et on peut donc vérifier que ce certificat est bien validé par ce tiers de confiance pour le site que l'on veut visiter.

Le défaut de ce fonctionnement est la confiance en les autorités de certification. Il arrive que des autorités ne soient pas dignes de confiance et signent des certificats invalides. Ainsi, en 2019, le Kazakhstan a imposé l'installation de l'autorité de certification gouvernementale sur tous les appareils et a signé son propre certificat pour facebook, ce qui leur permettrait de déchiffrer les pages facebook et les rechiffrer et resigner à destination des citoyens sans que leur navigateur ne signale de problème de sécurité.

Des protocoles permettent de limiter l'efficacité de cette sorte d'attaque tels que le Public Key Pinning (HPKP) et la Strict Transport Security (HSTS).