Introduction application web

Master IMST "Système d'information et d'archivage numérique" (IMST-SIAN)

Fabien Rico

Univ. Claude Bernard Lyon 1

10 octobre 2022

Plan

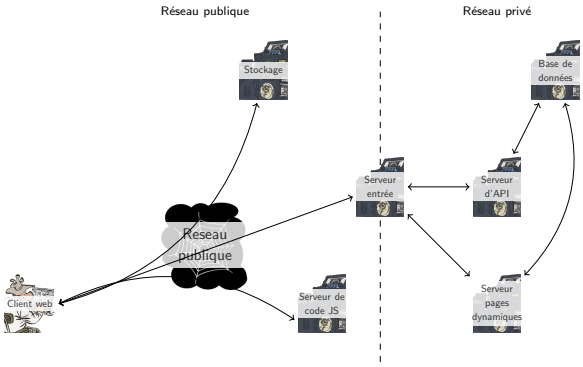
Introduction

Le web est le service le plus utilisé d'internet.

- Au départ, c'est un système de récupération de fichiers formatés avec des liens entre eux :
  - des navigateurs simple qui faisaient les requêtes et l'affichage ;
  - des *url* pour identifier les pages ;
  - des échanges initiés par le clients ;
  - le reste utilisait d'autres protocoles plus adaptés.
- Il y a eu une grosse évolution des besoins et des usages
  - apparition de services différents (streaming, reseaux sociaux, ...) ;
  - besoin d'échanges initiés par le serveur ;
  - besoin de sessions (liens entre les différentes requêtes, connexions durables) ;
  - utilisation de code sur les clients.

Pourquoi ces changements ?

Une application web classique



Les requêtes

Les requêtes se compliquent car les données sont séparées selon leur rôle

- La page de base : son contenu est fourni ou calculé par le serveur web (statique ou dynamique).
- Les informations de présentation (css et images), en général statiques.
- Les codes à éxécuter sur le client (JavaScript), en général statiques si on considère le point de vue du serveur.
- Les informations nécessaires à ce code, informations dynamiques confiées à des serveurs dédiés et transmis dans un format condensé (json).

La séparation permet de spécialiser les serveurs et de duppliquer ceux qui servent de calcul.

Les différentes parties

- Les serveurs de fichiers statiques :
  - contiennent les fichiers de code javascript exécutés sur le client ;
  - contiennent les fichiers d'image ou CSS, les pièces jointes etc
  - peuvent être des caches extérieurs ou des services externes (amazone S3, cloudflare, etc).
- Les serveur de pages ou les APIs :
  - permettent le dialogue avec les utilisateurs ;
  - utilisent différents langages (php, java, javascript) ;
  - les APIs sont contactés par les navigateurs clients pour obtenir des informations ;
  - sont en général des serveurs internes ou des applications gérées par le gestionaire du site.
- Les clients :
  - leur rôle est devenu très important dans les sites ;
  - utilisent le javascript ;
  - il y en a très peu (Chrome, Firefox, Safari).

Les API web

Ce sont des parties du site dont le rôle est de fournir ou de modifier les informations.

- Elles utilisent le protocole HTTP (le même que le site).
- Elles peuvent nécessiter une authentification.
- Elles renvoient des données exploitables par un programme.

Certaines API ont des formats normalisés pour aider le partage d'informations :

- Pour le partage de données documentaire, `OAI-PMH` est un format d'intérogation standard très répandu
  - qui permet d'interroger la base ;
  - qui renvoie en général des données en `Dublin Core (DC)` ou `Lom` ;
  - par exemple `http://oai.openedition.org/?verb=ListRecords&metadataPrefix=oai_dc`
- Certaines intitutions proposent une API en `SparQL`
  - Par exemple La BNF
- D'autres ont des API Ad-Hoc
  - Par exemple HAL

Les urls

Dans une page les ressources sont identifiées par des *url*

Example (Url)

`http://api.archives-ouvertes.fr/search/?q=test&wt=xml`

- `http` : le protocole de communication (le langage utilisé entre les machines)
- `api.archives-ouvertes.fr` : le nom de la machine à contacter
- `:80` (implicite) le port de la machine à contacter (c'est à dire le logiciel)
- `/search/?q=test&wt=xml` la requête
- `/search/` souvent le chemin vers le fichier de code sur le serveur (ici `/search/index.(php|js|??)`)
- `q=test&wt=xml` les paramètres fourni par `GET`

Protocole applicatif

Definition

Un protocole applicatif est un ensemble de règles définissant la façon dont deux applications vont échanger de l'information.

- Quand faut-il envoyer ou lire des données et quelle application le fait ?
- Comment est transmise l'information ?
- ...

Le choix du protocole est important car il rend plus ou moins difficile et fiable certaines opérations :

- Est-ce qu'il y a possibilité de tolérer les erreurs ?
- Est-ce qu'il faut des vérifications ?
- Est-ce que l'ensemble des données nécessaires au fonctionnement du protocole est important par rapport aux données utiles ?

Par exemple pour un flux audio en temps réel ou le téléchargement d'un petit fichier, d'un gros fichier ?

Protocole HTTP

`HTTP`, le protocole du web à été d'abord fait pour télécharger des pages simples avec des balises de présentation. Il a du beaucoup évoluer pour tenir compte des nouveaux usages.

- Problème pour les données qui ne sont pas du texte anglais (non accentué) : notion d'encodage et de type du contenu.
- Chaque téléchargement était séparé des autres (sans connexion) :
  - charge pour les serveurs lorsqu'il faut télécharger un grand nombre de fichiers (page, images, css,etc) ;
  - pas de moyen de suivre un utilisateur $\mapsto$ nécessité de garder une trace, des sessions et des données sur le client (`cookie`) ;
  - les données du protocole peuvent être importante par rapport au données utiles ;
- Normalement le serveur ne peux pas prévenir le client d'un évènement :
  - les clients interrogent régulièrement le serveur
  - utilisation d'une connexion persistante (websocket)

Protocole HTTP (suite)

Dans le protocole HTTP, le client commence la discussion par une requête :

- `GET` simple avec des paramètres dans l'url (transmis via l'entête) ;
- `POST` avec des données dans le corps de la requête ;
- `PUT`, `DELETE`, etc plus spécifiques pour des tâches précises.

L'entête permet de préciser un certain nombre de choses :

- quel type de réponse est demandé
- la description du navigateur
- la valeur des cookies conservés par le navigateur
- ...

Protocole HTTP (suite)

Le serveur répond aussi par le contenu demandé et un entête précisant :

- la réussite de la requête (code 200) ou des codes d'erreur (40.), (50.) ou des demande de modification de requête (30.) ;
- la description du serveur
- la description des données notament la taille, le type, le mode de transmission. . .
- . . .

Il existe de nombreuses bibliothèques pour programmer un client ou un serveur par exemple `requests` en python.

Le protocole HTTPS

C'est la version sécurisée du protocole `HTTP`, pour résumer :

- On utilise un autre protocole (`SSL/TLS`) pour mettre en place un canal chiffré entre le client et le serveur.
- Puis on utilise le protocole `HTTP` dans ce canal.

Le role de SSL est double :

- Vérifier que le serveur est bien celui demandé par un certificat.
- Mettre en place le chiffrement.

La chiffrement est simple, mais insuffisant car un attaquant pourrait se mettre entre le client et le serveur pour espioner (*man in the middle*)

Gestion des certificat

La gestion des certificats est un problème.

- Ils sont un équivalent de la carte d'identité, il faut donc qu'il soient délivrés par une autorité qui vérifie l'identité.
- L'ancienne méthode suppose un certain nombre d'entreprises dont le certificat est reconnu par tous et qui vérifient les identités :
  - assez sécurisé
  - ne supporte pas l'explosion des usages
  - problème des certificats autosignés
- Récement (2015), il a été proposé un protocole pour générer automatiquement des certificat *let's encrypt*
  - automatique
  - on prouve la maitrise d'un site web ou d'un nom de domaine et le certificat est délivré,
  - moins sur, il est souvent refusé par les institutions
  - massivement utilisé car gratuit et automatisable.

  De toute façon le certificat permet uniquement de savoir qu'une autorité reconnue a vérifié un nom.