HTTP

Michel Gagnon et Nikolay Radoev
École Polytechnique de Montréal

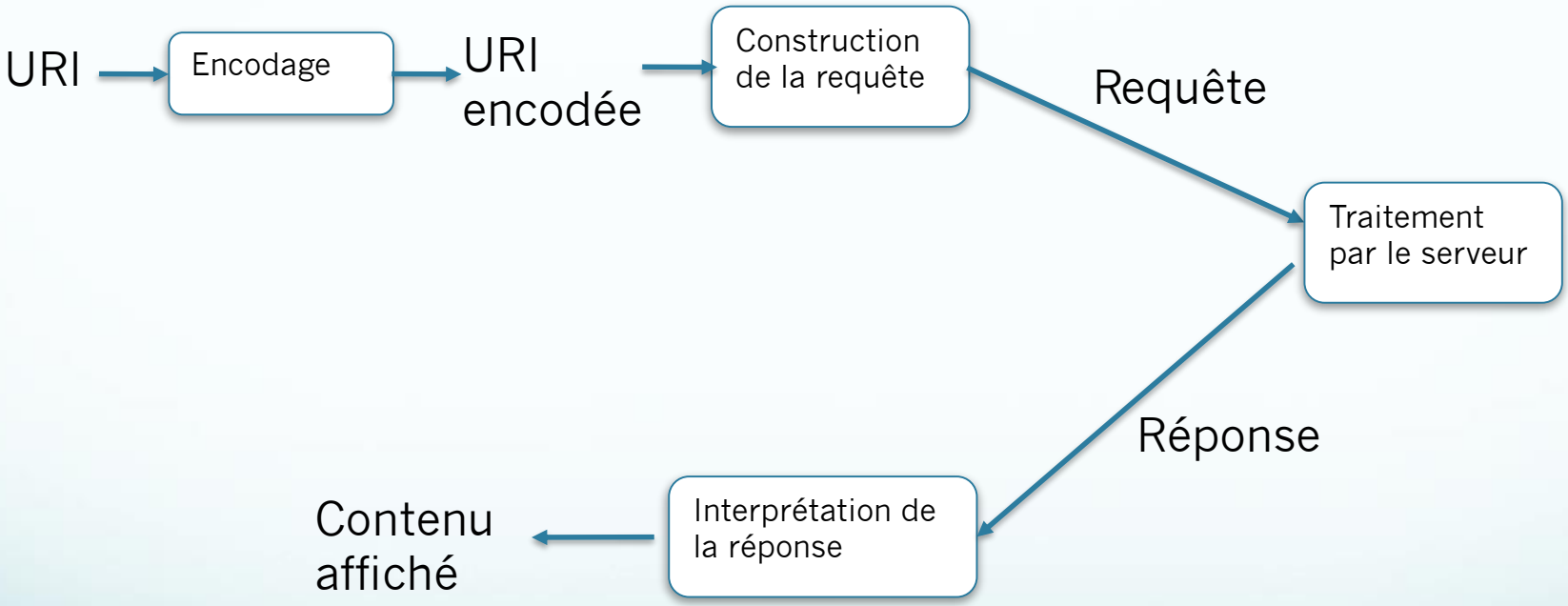
Protocole HTTP

- Version actuelle: HTTP 2.0
  - HTTP/3 encore sous proposition et encore expérimental
- Protocole asymétrique (*pull protocol*): le client envoie une requête et le serveur envoie une réponse
  - HTTP/2 offre une communication bidirectionnelle limitée
- Sans état (*stateless*)
- En principe sans connexion (une fois la requête lancée, le client se déconnecte et attend qu'une réponse du serveur établisse une nouvelle connexion)
- Permet de négocier le type de contenu échangé
- Permet la mise en cache

HTTP - Historique

- HTTP 0.9 (1991): Protocole simple proposé par Tim Berners-Lee
- HTTP 1.0 (1995) : Ajout de catégories MIME pour indiquer le type de contenu échangé
- HTTP 1.1 (1999) : Ajouts pour tenir compte des caches, proxys, hôtes virtuels, connexions persistantes, entre autres
- HTTP 2.0 (2015) : Connexions réellement persistantes et usage optimal des connexions, transmissions lancées par le serveur (push), encodage des échanges, pré-remplissage des caches, multiplexage.
- HTTP 3.0 (2020?) : Améliorations de la performance de transmission et correction de plusieurs défauts des versions précédentes. Utilisation du protocole QUIC au lieu de TCP+TLS

Protocole HTTP



URI

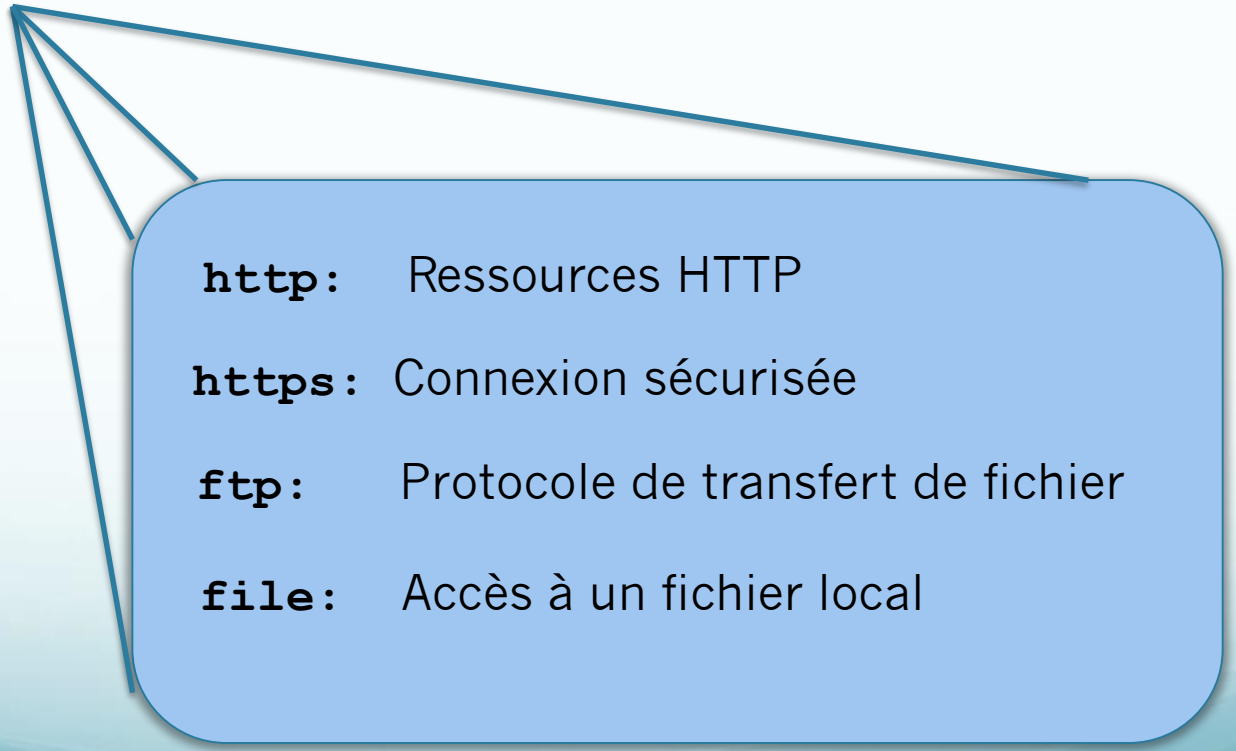
- Uniform Resource Identifier
- Il s'agit d'une courte chaîne de caractères identifiant une ressource
- Si elle désigne une ressource qu'on peut accéder par le Web, il s'agit alors d'une URL (Uniform Resource Locator)
- L'URI contient jusqu'à six éléments d'information, dans l'ordre suivant:
  - le protocole
  - l'hôte
  - le port (facultatif)
  - le chemin d'accès à la ressource
  - la chaîne de requête
  - un identificateur de fragment

URI

```
protocole://hôte:port/chemin?requête#fragment
```

URI

`protocole://hôte:port/chemin?requête#fragment`



URI

Dans certains cas, le protocole implique une forme différente de l'information qui suit:

```
mailto:<adresse>?<header1>=val1&header2=val2
```

Exemple:

```
mailto:michel.gagnon@polymtl.ca?subject=Coucou&body=Ceci%20est…
```

URI

`protocole://hôte:port/chemin?requête#fragment`

Nom de domaine DNS
exemple: **`www.polymtl.ca`**

Adresse IP
exemple: 132.207.235.136

`localhost` : désigne la machine locale

URI

```
protocole://hôte:port/chemin?requête#fragment
```

On spécifie le numéro de port qui sera utilisé par le serveur pour répondre à la requête.

Si on ne fournit pas cette information, le port 80 sera utilisé par défaut pour HTTP et 443 pour HTTPS.

URI

```
protocole://hôte:port/chemin?requête#fragment
```

Le chemin (ou route) est similaire à celui qu'on utilise pour parcourir une structure hiérarchique de fichiers en Unix.

Cette information est facultative.

Le chemin ne correspond pas nécessairement à un fichier sur le serveur. Son interprétation dépend du serveur.

URI

`protocole://hôte:port/chemin?requête#fragment`

Il n'y a pas de spécifications précises sur la forme de la requête.

En général on utilise la forme suivante:
`att1=val1&att2=val2&att3=val3$...`

URI

```
protocole://hôte:port/chemin?requête#fragment
```

Dans certains cas, la ressource demandée est décomposée en plusieurs sections.

Le fragment est un identificateur permettant d'identifier la section qui nous intéresse.

Par exemple, s'il s'agit d'une page HTML, le navigateur fera en sorte que la fenêtre d'affichage se positionne au début de la section désirée.

Encodage des URI

- Avant de se retrouver sur le réseau, certains caractères doivent être encodés, en utilisant le symbole « % » suivi du code ascii en hexadécimal
- Les espaces doivent être remplacés par « + » ou le code « %20 »
- Certains symboles, comme « : », « / », « ? », « % », « & », doivent être encodés, s'ils sont utilisés dans un autre contexte, comme par exemple le contenu textuel d'une valeur, ou le nom d'un fichier

URI relative

- Une URI peut ne pas contenir l'information sur l'hôte
- On peut retrouver une telle URI comme lien à l'intérieur d'une page HTML:

```
<img src="mon_image.jpg">
<a href="/help/howto.html">
<link script="../scripts/render.js">
```

- Il y a trois cas à considérer:
  - l'URI est un nom de fichier
  - l'URI commence par un /
  - l'URI correspond à un chemin relatif

URI relative (suite)

- Dans tous les cas, on suppose que la ressource doit être obtenue à partir du même hôte que celui qui nous a fourni la page HTML
- On prend ensuite comme référence le chemin indiqué dans l'URL qui a été envoyée pour extraire la page (si le dernier élément de ce chemin est un nom de fichier, on le retire)
- À partir de cette référence, on construit un nouveau chemin selon chacun des cas:

URL relative (suite)

- *L'URL relative est un nom de fichier:* On l'ajoute à la fin du chemin de référence
- *L'URL relative commence par / :* On considère alors qu'il s'agit d'un chemin absolu et remplacera le chemin de référence
- *L'URL relative et un chemin relatif:* On l'ajoute au chemin de référence, tout en prenant soin de remonter d'un niveau pour chaque occurrence de « .. »

URL relative - Exemple

- Soit un page HTML dont l'URL est

```
http://exemple.ca/home/index.html
```

- Voici comment les URL relatives suivantes seront résolues:

URL relative	
image1.jpg	http://exemple.ca/home/image1.jpg
images/image1.jpg	http://exemple.ca/home/images/image1.jpg
../image1.jpg	http://exemple.ca/image1.jpg
/ress/images/image1.jpg	http://exemple.ca/ress/images/image1.jpg

Requête

- Une requête HTTP a la forme suivante

```
Méthode URI VersionHTTP
Attribut1: Valeur1
Attribut2: Valeur2
…
AttributN: ValeurN
(ligne vide)
Corps (peut être vide en fonction du type de requête et l'implémentation de la librairie HTTP utilisée)
```

Requête

- Pour la suite, on s'intéressera surtout à HTTP/1.1

```
Méthode URI HTTP/1.1
Attribut1: Valeur1
Attribut2: Valeur2
…
AttributN: ValeurN
(ligne vide)
Corps (peut être vide)
```

Requête

- Une requête HTTP a la forme suivante

```
Méthode URI VersionHTTP
Attribut1: Valeur1
Attribut2: Valeur2
…
AttributN: ValeurN
(ligne vide)
Corps (peut être vide)
```

GET
POST
HEAD
DELETE
OPTIONS
etc.

Requête

- Une requête HTTP a la forme suivante

```
Méthode URI VersionHTTP
Attribut1: Valeur1
Attribut2: Valeur2
…
AttributN: ValeurN
(ligne vide)
Corps (peut être vide)
```

HTTP/1.0 (pratiquement disparu)

HTTP/1.1 (le plus utilisé)

HTTP/2.0 (gagne en popularité)

Requête

- Une requête HTTP a la forme suivante

```
Méthode URI HTTP/1.1
Attribut1: Valeur1
Attribut2: Valeur2
…
AttributN: ValeurN
(ligne vide)
Corps (peut être vide)
```

Avec HTTP/1.1, l'URI est généralement une URI relative. Le nom de l'hôte sera fourni comme valeur de l'attribut **HOST**. Cette séparation facilite l'utilisation d'hôtes virtuels.

Requête

- Une requête HTTP a la forme suivante

```
Méthode URI HTTP/1.1
Attribut1: Valeur1
Attribut2: Valeur2
…
AttributN: ValeurN
(ligne vide)
Corps (peut être vide)
```

Ici on retrouve les en-têtes, une par ligne. Le protocole définit plusieurs attributs pouvant être utilisés dans une requête.
Un attributs peut avoir plusieurs valeurs, séparées par des virgules.

Requête

- Une requête HTTP a la forme suivante

```
Méthode URI HTTP/1.1
Attribut1: Valeur1
Attribut2: Valeur2
…
AttributN: ValeurN
(ligne vide)
Corps (peut être vide)
```

En général, quand la méthode GET est utilisée, cette section est vide (les attributs de la requêtes sont dans l'URI).

Si la méthode est POST, cette section contiendra les attributs de la requête.

Méthodes

- GET:
  - Corps généralement vide
  - Sert à obtenir l'information correspondant à une ressource (une page HTML, une image, un fichier javascript, etc.)
  - Par défaut, le contenu obtenu peut être mis en cache
  - Les attributs de la requête sont fournis directement dans l'URI
  - À noter que la taille de l'URL est limitée, ce qui implique que GET doit être utilisé pour des requêtes courtes
    - La limite dépend des technologies utilisés : limite générale : 2000 caractères, CloudFlare accèpte jusqu'à 32KB, mais la spécification de HTTP ne donne pas un chiffre exact

Méthodes (suite)

- HEAD:
  - Comme GET, sauf que le serveur n'enverra que des en-têtes (pas de corps envoyé)
  - Utilisé en général pour obtenir des informations au sujet de la ressource (par exemple, pour savoir si elle a été modifiée depuis le dernier accès)

Méthodes (suite)

- POST:
  - Sert en général à transmettre des informations au serveur (par exemple, des données fournies par le biais d'un formulaire HTML)
  - Sert aussi à la création de nouvelles ressources
  - Requête non-idempotente : deux requêtes POST identiques consécutives peuvent produire des résultats différents (ex: création de 2 ressources différentes avec la même information)
  - Par défaut, la réponse du serveur n'est pas mise en cache
  - Les attributs de la requête sont fournis dans le corps

Méthodes (suite)

- PUT:
  - Sert en général à mettre à jour une ressource déjà existante en fournissant une version modifiée de la ressource au complet.
  - Requête idempotente: deux requêtes PUT identiques consécutives devraient produire le même résultat (ex: mise à jour de la même information d'une ressource plusieurs fois)
  - Le serveur peut retourner la ressource mise à jour ou simplement avec le code 200 ou ne rien retourner avec le code 204
  - Les attributs de la requête sont fournis dans le corps

Méthodes (suite)

- PATCH:
  - Sert en général à mettre à jour une ressource déjà existante en modifiant des parties ciblés de la ressource.
  - La modification peut être faite en fournissant des nouvelles valeurs ou en fournissant des instructions au propriétaire de la ressource pour indiquer quels changements effectuer.
  - Requête non-idempotente: deux requêtes PATCH identiques consécutives peuvent produire des résultats différents (ex: incrémenter une variable *count* de 1 à chaque requête)
  - Les attributs de la requête sont fournis généralement dans le corps

POST vs PATCH vs PUT

- Le protocole HTTP définit des rôles précis pour chaque méthode, mais n'impose pas une implémentation stricte.
- Le comportement et la gestion des requêtes POST,PATCH et PUT varie en fonction de l'implémentation utilisée et le respect des définitions
- Cet aspect sera exploré plus lors de la présentation de l'architecture REST. Voici quelques règles générales :
  - PUT devrait garantir l'idempotence. POST et PATCH peuvent dépendre du contexte
  - POST est utilisé pour la **création** d'une ressource
  - PATCH est utilisé pour la **modification partielle** d'une ressource
  - PUT est utilisé pour le **remplacement** d'une ressource spécifiée par un **URI** par une ressource **fournie** par la requête. Si la ressource n'existe pas, elle doit être **crée**. La différence entre PUT et POST dans le 2ème cas est parfois floue : idéalement, on vérifie l'existence de la ressource avant.

Méthodes (suite)

- DELETE:
  - Sert en général à mettre à supprimer une ressource existante.
  - Devrait retourner 200 OK ou 204 No Content en cas de réussite, en fonction du contexte.
  - Requête idempotente: deux requêtes DELETE identiques consécutives doivent produire le même résultat : la ressource est éliminée (si le traitement du serveur a réussi).
  - Les attributs de la requête (souvent l'identifiant de la ressource) sont fournis généralement dans l'URL directement.
    - Certaines librairies (ex: HttpModule d'Angular) ne permettent pas à une requête DELETE d'avoir un corps (*body*)

Méthodes (suite)

- OPTIONS:
  - Sert en général à connaître les options de communication acceptées par un URL ou un serveur au complet.
  - Utilisé dans les requêtes de pré-vérification (*preflight request*) pour vérifier si la méthode qu'on s'apprête à envoyer est accepté par le destinataire.
  - Requête pour le chemin "/resource/foo":

```
OPTIONS /resource/foo
Access-Control-Request-Method: DELETE
Access-Control-Request-Headers: origin, x-requested-with
Origin: https://foo.bar.org
```

  - Réponse comme quoi ce chemin accepte 3 méthodes:

```
HTTP/1.1 200 OK
Content-Length: 0
Access-Control-Allow-Methods: GET, OPTIONS, DELETE
```

Réponse du serveur

- La réponse du serveur a la forme suivante:

```
HTTP/1.1 code description
Attribut1: Valeur1
Attribut2: Valeur2
…
AttributN: ValeurN
(ligne vide)
Corps (peut être vide)
```

Réponse du serveur

- La réponse du serveur a la forme suivante:

```
HTTP/1.1 code description
Attribut1: Valeur1
Attribut2: Valeur2
…
AttributN: ValeurN
(ligne vide)
Corps (peut être vide)
```

1xx: Information
2xx: Succès
3xx: Redirection
4xx: Erreur de requête
5xx: Erreur du serveur

Réponse du serveur

- La réponse du serveur a la forme suivante:

```
HTTP/1.1 code description
Attribut1: Valeur1
Attribut2: Valeur2
…
AttributN: ValeurN
(ligne vide)
Corps (peut être vide)
```

Une (très!) courte description qui explique le code.

Réponse du serveur

- La réponse du serveur a la forme suivante:

```
HTTP/1.1 code description
Attribut1: Valeur1
Attribut2: Valeur2
…
AttributN: ValeurN
(ligne vide)
Corps (peut être vide)
```

Tout comme la requête, la réponse contient elle aussi des en-têtes.

Codes fréquemment utilisés

Code	Description
100 Continue	Le serveur attend d'autres informations pour poursuivre le traitement.
118 Connection timed out	Le temps de connexion est expiré.
200 OK	La requête a été traitée avec succès.
201 Created	Une nouvelle ressource a été créée sur le serveur.
202 Accepted	La requête a été traitée sans garantie de résultat.
204 No content	Le requête est traitée avec succès, mais aucune information n'est envoyée.

Codes fréquemment utilisés

Code	Description
100 Continue	Le serveur attend d'autres informations pour poursuivre le traitement.
118 Connection time ut	L
200 OK	
201 Created	
202 Accepted	
204 No content	

Utilisé dans le contexte d'une requête qui aurait un corps très volumineux. Pour éviter de gaspiller de la bande passante, on n'envoie que les en-têtes, parmi lesquelles on retrouve **Expect: 100-Continue**. Si le serveur peut traiter la requête avec succès, il enverra alors une réponse avec ce code, afin que le client puisse envoyer le contenu dans une nouvelle requête.

Codes fréquemment utilisés

Code	Description
100 Continue	Le serveur attend d'autres informations pour poursuivre le traitement.
118 Connection timed out	L
200 OK	s.
201 Created	sur le
202 Accepted	tie de
204 No content	mais aucune information n'est envoyée.

Code de retour très fréquent. Utilisé lorsqu'on récupère une ressource avec succès. Peut être utilisé dans le cas de la modification d'une ressource qui a un impact sur la page présentée à l'utilisateur (contrairement aux codes 202 ou 204).

Codes fréquemment utilisés

Code	Description
100 Continue	Le serveur attend d'autres informations pour poursuivre le traitement.
118 Connection timed out	Le temps de connexion est expiré.
200 OK	
201 Created	
202 Accepted	
204 No content	

Comme nous le verrons plus tard, ce code est utilisé dans un contexte de service web. La requête envoyée au serveur implique qu'il doivent créer une nouvelle ressource (sur le serveur) et par cette réponse il indique que celle-ci a été créée avec succès.

Codes fréquemment utilisés

Code	Description
100 Continue	Le serveur attend d'autres informations pour poursuivre le traitement.
118 Connection timed out	Le temps de connexion est expiré.
200 OK	L
201 Created	U s
202 Accepted	L
204 No content	

Très similaire au code 200 OK.
Utilisé dans le cas ou une ressource est modifiée (avec une requête PUT par exemple) sans qu'on ait besoin de modifier la page affichée à l'utilisateur.

Codes fréquemment utilisés

Code	Description
300 Multiple choices	Plusieurs versions disponibles pour la ressource demandée.
301 Moved permanently	Ressource déplacée de manière permanente.
302 Moved temporarily	Ressource déplacée de manière temporaire.
303 See other	Redirection vers une autre ressource.
304 Not modified	La ressource n'a pas été modifiée depuis la dernière requête.
307 Temporary redirect	La requête doit être redirigée temporairement vers l'URI spécifiée.
308 Permanent redirect	La requête doit être redirigée de manière permanente vers l'URI spécifiée.

Codes fréquemment utilisés

Code	Description
300 Multiple choices	Plusieurs versions disponibles pour la ressource demandée.
301 Moved permanently	Resso ermanente.
302 Moved temporarily	Re mporaire.
303 See other	Redi ce.
304 Not modified	La re depuis la derr
307 Temporary redirect	La re orairement vers
308 Permanent redirect	La requête doit être redirigée de manière permanente vers l'URI spécifiée.

Très souvent le corps de la réponse est une page HTML avec tous les liens proposés pour obtenir les différentes versions de cette ressource.

Codes fréquemment utilisés

Code	Description
300 Multiple choices	Plusieurs versions disponibles pour la ressource demandée.
301 Moved permanently	Ressourc manente.
302 Moved temporarily	Re oraire.
303 See other	Redirec
304 Not modified	La ress uis la derniè
307 Temporary redirect	La requ irement vers l'U
308 Permanent redirect	La requête nière permanente vers l'URI spécifiée.

Dans ce cas, la réponse contient un attribut **Location**, qui indique le lien à suivre pour obtenir la ressource. Le client devrait mettre à jour le lien dans sa base de données.

Codes fréquemment utilisés

Code	Description
300 Multiple choices	Plusieurs versions disponibles pour la ressourc
301 Moved permanently	Ressou
302 Moved temporarily	
303 See other	Redir
304 Not modified	La ress dernièr
307 Temporary redirect	La requ vers l'U
308 Permanent redirect	La requ permar

Dans ce cas, la réponse contient un attribut **Location**, qui indique le lien à suivre pour obtenir la ressource, sans changer la nature de la requête. Comme il s'agit d'une redirection temporaire, le client ne doit pas mettre à jour sa base de données. Ce code a souvent été mal interprété par les navigateurs. Les codes 303 et 307 ont été proposés pour éliminer l'ambiguïté du code 302.

Codes fréquemment utilisés

Code	Description
300 Multiple choices	Plusieurs versions disponibles pour la ressource d
301 Moved permanently	Ressourc
302 Moved temporarily	
303 See other	Redirecti
304 Not modified	La ressou dernière r
307 Temporary redirect	La requêt vers l'URI
308 Permanent redirect	La requêt permanent

Indique un lien à une autre ressource qui est liée d'une certaine manière à celle qui a été spécifiée dans la requête. Le lien vers cette ressource est indiqué par l'attribut **Location**. On doit faire une nouvelle requête GET. Par exemple, si la requête était un POST, cette réponse indiquerait le lien vers la page résultat, qu'il faudrait alors récupérer par une nouvelle requête utilisant la méthode GET.

Codes fréquemment utilisés

Code	Description
300 Multiple choices	Plusieurs r… ressource…
301 Moved permanently	Ressourc…
302 Moved temporarily	Ressourc…
303 See other	Redir…
304 Not modified	… dernière r…
307 Temporary redirect	La requête d… …ent vers l'URI spécifiée.
308 Permanent redirect	La requête doit être redirigée de manière permanente vers l'URI spécifiée.

Comme nous le verrons plus loin, cette réponse est liée à un GET conditionnel, et indique que la ressource demandée n'a pas été modifiée depuis la dernière fois qu'elle a été récupérée par le client.

Codes fréquemment utilisés

Code	Description
300 Multiple choices	Plusieurs versions disponibles pour la ressource
301 Moved permanently	Ressourc
302 Moved temporarily	Ressourc
303 See other	Redirecti
304 Not modified	La ress
307 Temporary redirect	La requ vers l'URI
308 Permanent redirect	La requête doit être redirigée de manière permanente vers l'URI spécifiée.

Ne pas confondre avec 302, ni avec 303. Le code 307 exige qu'on suive le lien proposé (avec l'attribut **Location**) en utilisant la même méthode que celle de la requête originale.
Différence : Avec 303, on ne peut pas changer la méthode HTTP (ex. POST à GET)

Codes fréquemment utilisés

Code	Description
300 Multiple choices	Plusieurs versions disponibles pour la ressource demandée.
301 Moved permanently	Ressource déplacée de manière permanente.
302 Moved temporarily	Ressourc
303 See other	Redirecti
304 Not modified	La resso dernière
307 Temporary redirect	La requê vers l'URI spe
308 Permanent redirect	La requête doit être redirigée de manière permanente vers l'URI spécifiée.

Idem à 307, mais pour une redirection permanente.
Différence : Avec 308, on ne peut pas changer la méthode HTTP (ex. POST à GET)

Codes fréquemment utilisés

Code	Description
400 Bad request	La requête n'est pas bien formée.
401 Unauthorized	La ressource demandée nécessite une authentification qui n'a pas été fournie avec la requête.
403 Forbidden	L'accès à la ressource n'est pas autorisé.
404 Not found	La ressource demandée n'existe pas.
405 Method not allowed	La méthode utilisée n'est pas acceptée par le serveur.
406 Not acceptable	La ressource est demandée dans un format que le serveur ne peut pas fournir.
410 Gone	La ressource n'existe plus.
412 Precondition failed	Une précondition précisée dans la requête n'as pas pu être respectée.

Codes fréquemment utilisés

Code	Description
400 Bad request	La requête n'est pas bien formée.
401 Unauthorized	La ressource demandée nécessite un authentification qui n'a pas été fournie avec la requête.
403 Forbidden	L'accès à
404 Not found	La ressou
405 Method not allowed	La métho serveur
406 Not acceptable	que le ser
410 Gone	La ressource n'existe plus.
412 Precondition failed	Une précondition précisée dans la requête n'as pas pu être respectée.

Utilisé dans le contexte de négociation de contenu. Par exemple, la requête indique que le contenu envoyé ne peut être qu'en français, et la ressource n'est pas disponible en version française.

Codes fréquemment utilisés

Code	Description
400 Bad request	La requête n'est pas bien formée.
401 Unauthorized	La ressource demandée nécessite un authentification qui n'a pas été fournie avec la requête.
403 Forbidden	L'accès à
404 Not found	La ressou
405 Method not allowed	La métho serveur.
406 Not acceptable	La res que
410 Gone	L ressource n'existe plus.
412 Precondition failed	Une précondition précisée dans la requête n'as pas pu être respectée.

Supposons par exemple que le client a mis la condition suivante dans sa requête: **Expect: 100-Continue**. Si le serveur n'est pas en mesure de permettre au client d'envoyer le contenu désiré, il répondra par le code 412.

Codes fréquemment utilisés

Code	Description
500 Internal server error	Un problème dans le serveur l'empêche de répondre à la requête.
501 Not implemented	Le serveur ne comprend pas la méthode qui est spécifiée dans la requête.
503 Service unavailable	Le service est interrompu temporairement, possiblement pour la maintenance.

En-têtes

- Les en-têtes remplissent plusieurs fonctions:
  - Informations sur la requête elle-même
  - Informations sur le client ou le serveur
  - Négociation de contenu
  - Gestion des caches
  - Gestion des cookies
  - Contrôle d'accès
  - Gestion de la connexion
- Certains sont utilisés seulement par le client (C), d'autres seulement par le serveur (S) et, finalement, certains sont utilisés par les deux (CS)

Informations sur la requête

En –tête	Description
Date (CS)	Date de création de la requête ou de la réponse. Cet en-tête est obligatoire.
Via (S)	Informe le client de la liste des proxies par lesquels la requête est passée.
Content-Length (CS)	Longueur du corps en octets.
Host (C)	Nom de domaine du serveur du site auquel on veut accéder.

Informations sur le client ou le serveur

En –tête	Description
Referer (C)	Lorsque la requête est issue d'un lien dans une page, cet attribut contient l'URL de cette page.
User-Agent (C)	Description du client (contient entre autres la version du navigateur).
Server (S)	Description du serveur.

Négociation de contenu

En –tête	Description
Accept (C)	Spécifie le type MIME des contenus que le client peut accepter.
Accept-Charset (C)	Spécifie le code de caractères accepté (**utf-8**, **iso-8859-1**, etc). On utilise « * » pour accepter tous les jeux de caractères.
Accept-Encoding (C)	Liste des encodages acceptés (gzip, compress, etc.).
Accept-Language (C)	Liste des langues acceptées.
Content-Type (CS)	Type MIME du contenu envoyé.
Content-Language (S)	Langue du contenu envoyé.
Content-Encoding (S)	Encodage utilisé pour le contenu envoyé (gzip, deflate, compress, etc.)
Vary (S)	Indique quels en-têtes sont utilisés par le serveur pour déterminer le format du contenu envoyé.

Type MIME

- MIME signifie Multipurpose Internet Mail Extension
- Autrefois utilisé pour identifier le type de document attaché à un courriel, il est aujourd'hui devenu une norme pour définir n'importe quel type de ressource échangée sur le web
- Il est composé de deux attributs séparés par un « / »
- Le premier attribut identifie une classe générale: **application**, **image**, **audio**, **video**, **text**, **multipart**, etc.
- Le deuxième définit de manière plus précise le type de document. Par exemple, dans la catégorie **text**, on retrouve les sous-types suivants: **plain**, **html**, **xml**, etc.
- La catégorie **application** sert à identifier un type de document spécifique à une application, qui sera indiqué par le deuxième attribut: **application/msword**, **application/javascript**, **application/pdf**, etc.
- La catégorie **multipart** sert à indiquer que le corps est séparé en plusieurs morceaux

Négociation de contenu

- Il s'agit d'un échange avec le serveur afin de s'assurer, lorsque possible, que la ressource soit transmise dans le format et la langue désirée
- HTTP permet de classer les différentes alternatives avec des valeurs de priorité
- Essentiellement, trois approches possibles:
  - Négociation contrôlée par le serveur
  - Négociation contrôlée par le client
  - Négociation transparente, par le biais des caches

Négociation contrôlée par le serveur

- Le client spécifie ce qu'il veut, grâce aux attributs **Accept-***
- Pour chacun de ces attributs, le client peut spécifier plusieurs valeurs
- Les valeurs peuvent être un nombre entre 0 et 1, pour indiquer le degré de préférence (par défaut, la valeur est 1)
- Prenons, par exemple, l'en-tête suivant:

  Accept: text/html, application/xml;q=0.5, text/plain;q=0.4, */*; q=0.1

  Dans ce cas, le client désire un document HTML. Sinon, il acceptera un document XML, un texte simple, ou n'importe quoi d'autre, dans cet ordre
- Le serveur utilise un algorithme, qui utilise les valeurs de ces attributs, et d'autres informations s'il y a lieu, et décide de ce qui sera envoyé comme contenu
- Il est important de noter que lorsque le navigateur établit une requête pour une image, un vidéo, un fichier audio, il utilise cette technique pour amorcer une négociation de contenu (les valeurs utilisées dépendent du navigateur)

Remarques sur l'attribut **Vary**

- Cet attribut est utile pour les caches de proxy qui peuvent se trouver sur le chemin entre le client et le serveur
- En spécifiant les en-têtes utilisés pour déterminer le format du contenu envoyé, un proxy peut utiliser le même algorithme pour déterminer s'il peut renvoyer le contenu dans le format qui se trouve dans sa cache

Négociation contrôlée par le client

- La négociation contrôlée par le client sert à éviter les problème de la négociation contrôlée par le serveur:
  - Devient vite complexe et inefficace, dû au grand nombre d'attributs, auxquels pourraient s'en ajouter d'autres
  - Il faut envoyer les en-têtes pour chaque requête, ce qui peut devenir lourd lorsqu'il y a beaucoup de requêtes et/ou en-têtes
  - Beaucoup d'informations spécifiques permettant de caractériser le client, ce qui entraîne des problèmes de sécurité
- Le principe est simple: le serveur envoie l'ensemble des possibilités (un ensemble de liens HTML par exemple), parmi lesquelles le client choisit celle qui lui convient
- Peut être implémenté avec le code **300 Multiple Choices** avec l'en-tête **Location** pour le choix préféré (ou par défaut)

Négociation de contenu transparente

- Compromis entre les deux approches
- Lorsque le serveur n'arrive pas à déterminer dans quel format il doit envoyer le contenu, il choisit le plus probable
- Le serveur indique aussi qu'il y a des alternatives, en fournissant des liens sur celles-ci
- HTTP n'a pas de spécifications précises pour ce type de négociation
  - Le protocole WebDAV, une extension du HTTP faites pour la gestion de fichiers distribués, définit le code 207 Multi-Status pour tels cas.

Gestion des caches

En –tête	Description
Cache-Control (CS)	Détermine si la ressource doit être mise en cache, ou si elle doit est revalidée avant d'être extraite de la cache.
If-Match (C) If-Modified-Since (C) If-None-Match (C) If-Unmodified-Since (C)	Pour faire un GET conditionnel: on veut s'assurer que la ressource en cache est encore valide. Si c'est le cas, c'est elle qu'on prend. Sinon, on fait une requête au serveur pour l'obtenir.
Etag (S)	Une étiquette qui identifie de manière unique la ressource demandée.
Expires (S)	Indique à quel moment la ressource envoyée devra être considérée invalide dans la cache.
Last-Modified (S)	Spécifie la dernière date de modification de la ressource.

Gestion des caches

- Tout navigateur ou proxy possède une cache, dans laquelle une ressource qui a été obtenue en réponse à une requête peut être stockée
- Si la requête se présente à nouveau, on peut donc extraire la ressource de la cache, plutôt que de lancer une nouvelle requête
- Une question importante se pose alors: la ressource en cache est-elle encore valide?
- Si ce n'est pas le cas, il faudra lancer une requête au serveur pour obtenir une version à jour de la ressource

Gestions des caches

- Dans sa réponse à une requête, le serveur peut déterminer, avec l'attribut **Cache-Control**, si la ressource doit ou peut être mise en cache, selon la valeur spécifiée:
  - **private** : indique que la ressource peut être mise en cache seulement chez le client (le navigateur)
  - **public** : la ressource peut être mise en cache chez le client et dans les proxies
  - **no-store** : la ressource ne DOIT PAS être mise en cache
  - **no-cache** : la ressource sera mise en cache, mais on doit TOUJOURS revalider son contenu avec le serveur avant de l'utiliser. La ressource est mise en cache, mais elle est automatiquement expirée.
  - **must-revalidate**: la ressource doit toujours être revalidée avant de l'utiliser lorsqu'elle est expirée (plus sévère que **no-cache**: on ne doit pas utiliser le contenu en cache si le serveur n'a pas répondu)
  - **max-age = xxx** : indique le temps total que la ressource pourra rester en cache

Gestions des caches

Les contrôles de cache *must-revalide* et *no-cache* mentionnent la revalidation du contenu de la ressource. Supposons qu'un proxy contient la ressource en cache.

Comment revalide-t-il la ressource?

Gestions des caches

Les contrôles de cache *must-revalide* et *no-cache* mentionnent la revalidation du contenu de la ressource. Supposons qu'un proxy contient la ressource en cache.

Comment revalide-t-il la ressource?

HEAD

Gestions des caches

Quelle est la différence entre les deux en-têtes suivantes?

Cache-Control: max-age=0, must-revalidate

Cache-Control: no-cache

Normalement, “no-cache” permet de réutiliser la ressource en cache si le serveur ne répond pas tandis que “must-revalidate” ferait que le proxy/navigateur retourne une erreur 504.

Cependant, ce comportement varie selon la technologie utilisée : certaines versions de Chrome ne permettent pas la reutilisation de la ressource en cache non-validée (les 2 en-têtes sont fonctionnellement pareilles)

Gestions des caches

- Le serveur peut utiliser l'en-tête **Expires**, qui indique la date à partir de laquelle une ressource mise en cache ne sera plus valide

  Par contre, **max-age** a précédence sur **Expires**

- Le serveur peut aussi faire appel à deux techniques pour spécifier la dernière version de la ressource:
  - Ajouter un en-tête **last-modified** dans la réponse : dans ce cas, la date fournie est celle qui correspond à la dernière version de la ressource
  - Ajouter un en-tête **ETag** dans la réponse: la valeur fournie est une étiquette unique qui désigne la version actuelle de la ressource sur le serveur

- Ces méthodes permettent au client d'effectuer un GET conditionnel, s'il détecte que la ressource en cache est expirée

Requête conditionnelle

- On fait un GET conditionnel lorsque la ressource demandée se trouve en cache, mais qu'elle n'est plus valide (le temps spécifié par l'attribut **max-age** a été dépassé, ou la date d'expiration est dépassée):

- Si on a reçu un **ETag** pour cette ressource, on demande tout simplement au serveur s'il possède une ressource avec cette étiquette:

```
GET www.exemple.com/index.html HTTP/1.1
If-None-Match: "8eca4-205f-17b94c"
```

  Si la ressource **index.html** qui se trouve sur le serveur est associée à cette étiquette, cela signifie qu'elle n'a pas été modifiée. Le serveur enverra alors le code 304 comme réponse, avec les nouvelles informations requises pour l'expiration de la ressource en cache. Sinon, il enverra le code 200 avec le contenu de la ressource demandée. À noter qu'il peut y avoir plusieurs étiquettes pour une même ressource.

Requête conditionnelle

- Si on a reçu un en-tête **Last-modified** pour cette ressource, la dernière fois qu'on l'a demandée, le navigateur a mémorisé cette date
- On demande tout simplement au serveur si la ressource a été modifiée depuis cette date:

```
GET www.exemple.com/index.html HTTP/1.1
If-Modified-Since: Sat, 05 Oct 2020 19:43:31 GMT
```

  Le serveur enverra une nouvelle version de la ressource seulement si elle a été modifiée depuis cette date

Et si aucun en-tête de gestion de cache n'est envoyé?

- On utilise une heuristique pour « estimer » le temps de validité qu'il reste
- Exemple:
  - DR est la date où la ressource a été reçue et mise en cache
  - DL est la date de dernière modification de la ressource, telle que spécifiée par l'en-tête **Last-Modified**
  - On fixe l'âge maximal de la ressource à une fraction de la différence DR – DL (typiquement 10%)
  - Si la ressource est en cache depuis un temps supérieur à cet âge maximal, on considère qu'elle est expirée

Gestion des cookies

En –tête	Description
Cookie(C)	Contient l'information associée à un cookie.
Set-Cookie (S)	Spécifie qu'un cookie doit être stocké sur le client. L'information qui y est associée devra être ajoutée à toutes les prochaines requêtes. On peut spécifier de manière plus précise les domaines et chemins pour lesquels le cookie s'applique. On peut aussi indiquer la date d'expiration du cookie. L'attribut "**Secure**" oblige l'utilisation de HTTPS et l'attribut "**HttpOnly**" rend le cookie inaccessible à l'API **Document.cookie**, donc inchangeable par le JavaScript.

Gestion des cookies

- Quand utiliser le localStorage/sessionStorage vs les cookies?

Gestion des cookies

- *localStorage/sessionStorage* lorsque les données seront utilisées seulement par le client
  - LocalStorage peut garder jusqu'à 5MB de data (objets JS ou primitives) vs un cookie qui est limité à 4KB (string seulement)
- *cookies* lorsque les données doit être partagées entre le client et le serveur
  - Permet de se rappeler d'un état (*state*) contrairement à HTTP qui devrait être sans-état (*stateless*)

Exemple de Cookie

- Voici un exemple de cookies envoyés par le serveur

HTTP/1.1 200 OK
Cache-Control: no-cache
Set-Cookie: user_session=AJi4QfEa3D; expires = Wed, 29-Sep-2021 15:23:52 GTM; Secure;

Set-Cookie; first_login=expires=Wed, 29-Sep-2021 15:23:52 GMT; path=/; domain=.example.com; **Secure**; **HttpOnly**; priority=high;

Gestion d'accès

En –tête	Description
Origin (C)	Spécifie le domaine de la page d'origine. Donc, s'il s'agit d'un accès à un site d'une autre domaine, le serveur pourra décider s'il accepte de transmettre le contenu de la ressource demandée.
Access-Control-Allow-Origin (S)	Fournit la liste des domaines d'origine acceptés par le serveur. On utilise « * » pour spécifier un accès universel.

Partage de ressources d'origines multiples

- Pour des raisons de sécurité, un navigateur refuse d'accéder à un site dont le domaine diffère de celui de la page originalement chargée
- Pour contourner ce problème, on peut utiliser l'en-tête **Origin**
- Le serveur répondra en indiquant les origines qu'il accepte
- Si l'origine indiquée par le client fait partie de la liste, la ressource sera envoyée au client
- Cette méthode simple est utilisée pour des requêtes GET, HEAD ou POST pour l'envoi de formulaire
- Sinon, le protocole est plus complexe
  - Cross-Origin Resource Sharing (CORS)