R4.01-R4.A.10

HTTP et API rest

monnerat@u-pec.fr

IUT de Fontainebleau

Sommaire

Rappels réseau

API Rest

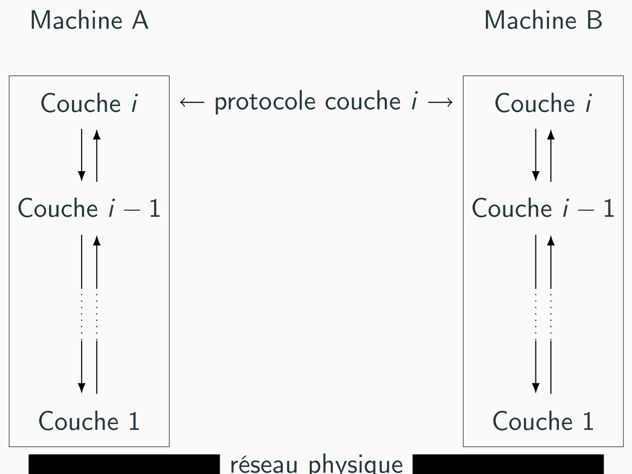
Exemple avec php

Firebase

Rappels réseau

Principe :

- la communication se fait entre deux couches de même niveau
- chaque couche offre des services à la couche immédiatement supérieure
- chaque couche s'appuie sur des services de la couche immédiatement inférieure.


Machine A
Machine B
Couche i
← protocole couche i →
Couche i
Couche $i-1$
Couche $i-1$
Couche 1
Couche 1
réseau physique

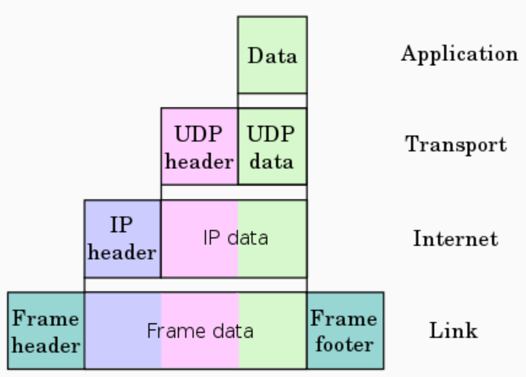
Modèle OSI

couche	7	Application	
	6	Présentation	
	5	Session	
	4	Transport	Segment/datagramme
	3	Réseau	Paquet
	2	Liaison	Trame
	1	Physique	bit

PDU : Protocal Data Unit

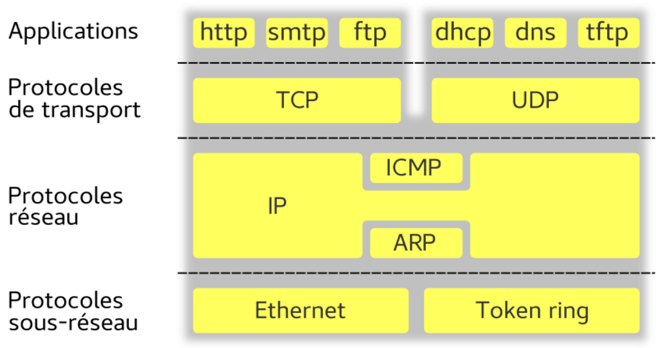
Couche	Protocoles	Unités de données
Application, Présentation, Session	ftp, telnet, http, smtp, etc.	Messages
Transport	TCP ou UDP	Segment TCP, Datagramme UDP
Réseau	Protocole de routage : IP	Paquets
Liaison	802.x,PPP,HDLC	Trames
Physique	Physique	Physique

Encapsulation



Modèle en couche d'Internet

Moins de couche qu'OSI



Les spécifications des protocoles d'internet sont publiées dans les RFC : documents textuels, numérotés, disponibles gratuitement sur internet :

- `http://www.ietf.org/rfc.html`
- `http://www.faqs.org/rfcs/`

Exemples :

UDP	:	RFC 768 (1980)
IP	:	RFC 791 (1981)
TCP	:	RFC 793 (1981)
HTTP	:	RFC 1630 (1996)

Protocole = format de message ou syntaxe + règles d'échange

Exemples : $\underbrace{\text{IP, ICMP, TCP, UDP}}_{\text{"binaire"}}$, $\underbrace{\text{HTTP,SMTP, POP3}}_{\text{"ascii"}}$

Protocles Binaires : (couches 3, 4)

- très compact
- non "lisible humainement"

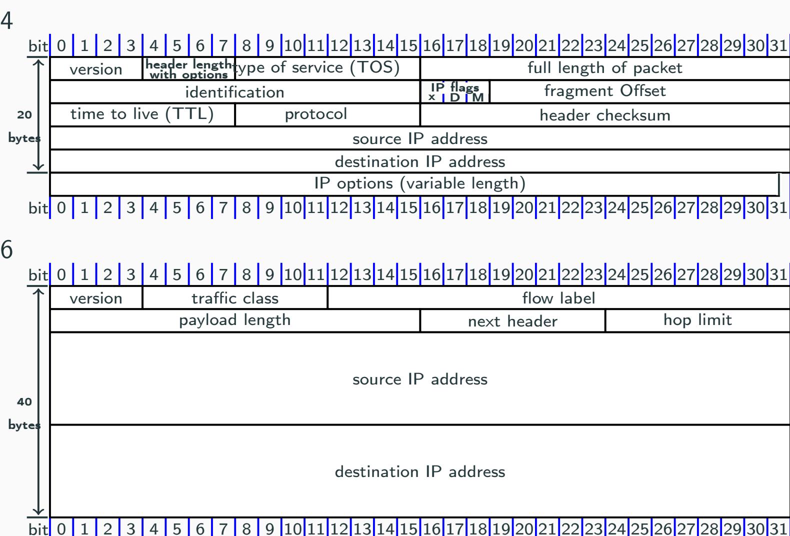
Protocles texte : (couche 7)

- lisisble et traçable
- on peut dialoguer directement au clavier avec netcat, etc.

Rappels réseau

IP

IP4/6 Headers



Rappels réseau

UDP

<table>
<tr><td>bit</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td rowspan="2">8 bytes</td><td colspan="16">source port</td><td colspan="16">destination port</td></tr>
<tr><td colspan="16">length</td><td colspan="16">checksum</td></tr>
<tr><td>bit</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
</table>

Question : quelle est la taille maximale d'un payload de datagramme udp avec ip v4 ? v6 ?

La taille maximale théorique du payload d'un datagramme UDP est $2^{16} - 1 - 8 = 65527$.

- Encapsulé dans IP v4 : taille payload = taille totale - entêtes = $2^{16} - 1 - 20 = 65515$.
  En retirant les entêtes UDP $\Rightarrow 65515 - 8 = 65507$.
- Encapsulé dans IP v6 : taille payload = $2^{16} - 1 = 65535$.
  En retirant les entêtes UDP $\Rightarrow 65535 - 8 = 65527$

Rappels réseau

TCP

Protocole fiable (couche 4), RFC 793, au dessus de IP. Permet de

- multiplexer les échanges (programmes $\leftrightarrow$ programmes) grâce au concept de port
- initialiser, maintenir et fermer une connexion
- remettre en ordre les segments TCP
- acquitter et réémettre les segments TCP
- adapter le flot de données à la charge du réseau

Segment TCP

Entête + Données

Entête binaire : entre 20 et 60 octets

- Port source et destination
- Numéro d'ordre (seq number)
- Numéro accusé de réception (ack number)
- Taille de l'entête en mutlitple de 32 bits (4 bits)
- Drapeaux (bits) : ACK, RST, SYN, FIN, etc. (12 bits)
- Fenêtre
- Somme de contrôle
- Options
- Bourrage (alignement 32 bits)

bit 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

source port | destination port

sequence number

acknowledgment number

offset | reserved | TCP flags (CWR ECE URG ACK PSH RST SYN FIN) | window

checksum | urgent pointer

TCP options (variable length, optional)

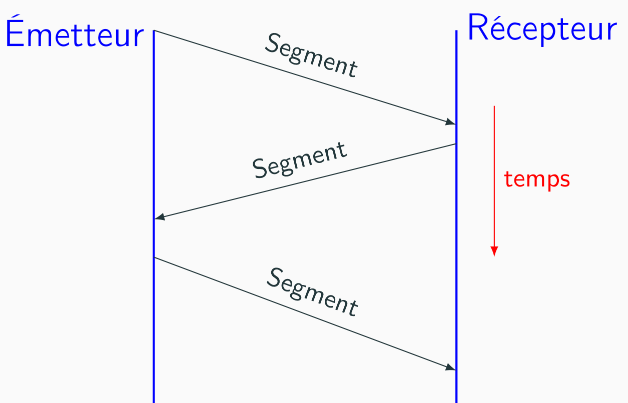
bit 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

20 bytes

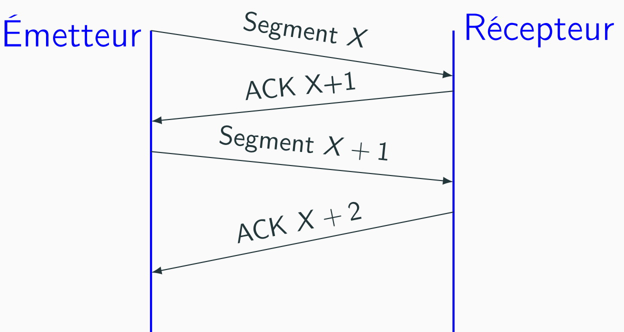
offset

Drapeaux = 9 flags sur 1 bit + 3 bits réservés

- ACK : accusé de reception (acknowledgement)
- RST : rupture anormale de connexion (reset)
- SYN : établissement de connexion (synchronisation)
- FIN : demande la fin de la connexion
- ECN : signale la présence de congestion
- etc.


Émetteur
Récepteur
Segment
Segment
temps
Segment

Acquittement



Principe

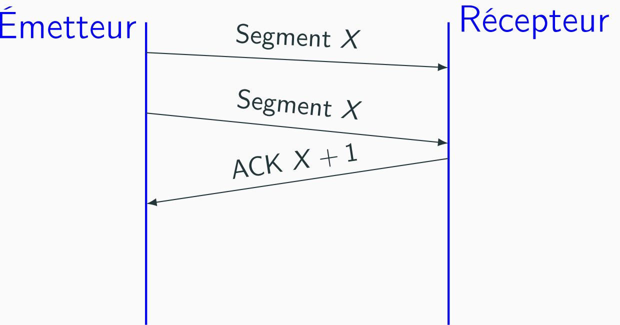
Émission d'un segment avec un numéro d'ordre X (numéro du premier octet du segment)

$\rightarrow$ Réponse avec drapeau ACK + numéro accusé de réception (NAR)

Interprétation : tous les segments $<$ NAR ont été reçus.

Remarque : NAR = numéro d'odre du prochain segment attendu

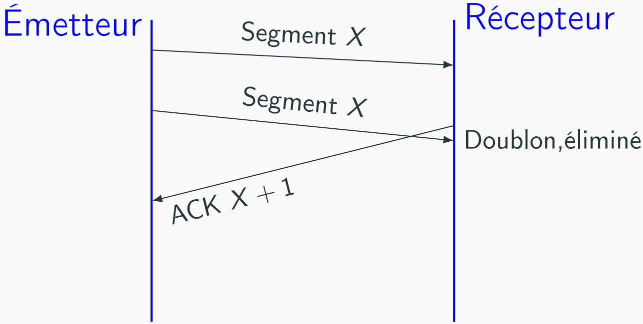
Réémission



Les segments émis sont mémorisés jusqu'à acquittement.

Au bout d'un timeout (RTO, dynamique), un segment non acquitté est réémis. (il y a un maximum ! `/proc/sys/net/ipv4/tcp_retries2`)

Doublon



Si le segment est réémis, cela peut créer un doublon.

Ils sont détectés au niveau du recepteur sur numéro d'ordre.

Connexion

Établissement d'une connexion : suppose

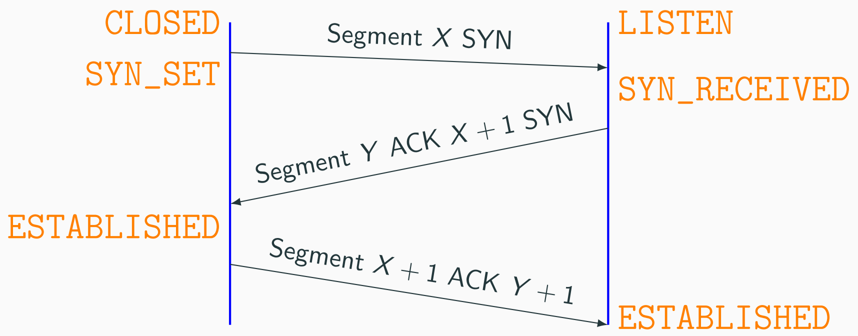
- une machine en demande de connexion : le client (état `CLOSED` $\rightarrow$ `SYN_SENT`)
- une machine en attente de connexion : le serveur (état `LISTEN`)

Chacune tire au sort un numéro d'ordre de départ X (client) et Y (serveur)

Connexion =

- se transmettre les numéros d'ordre (synchronisation) ;
- passer à l'état `ESTABLISHED`

Poignée de main (3 temps)



On utilise le flag SYN.

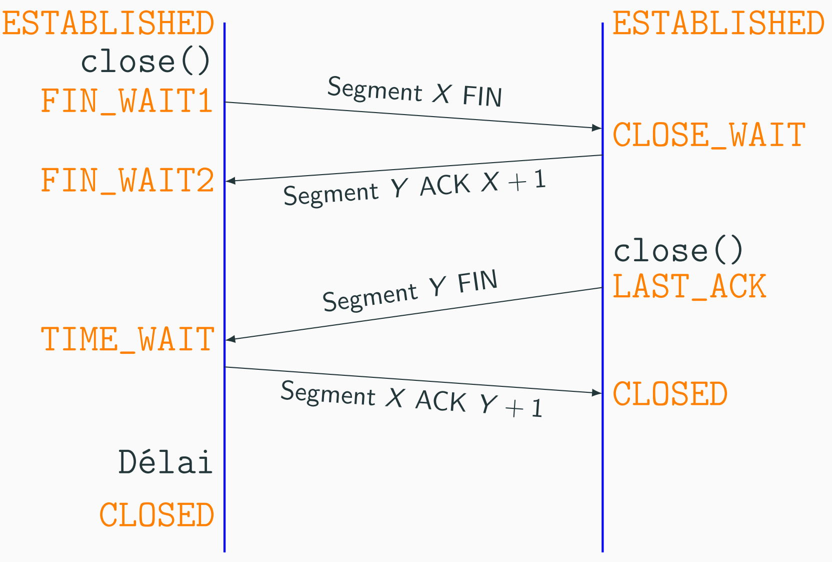
Déconnexion

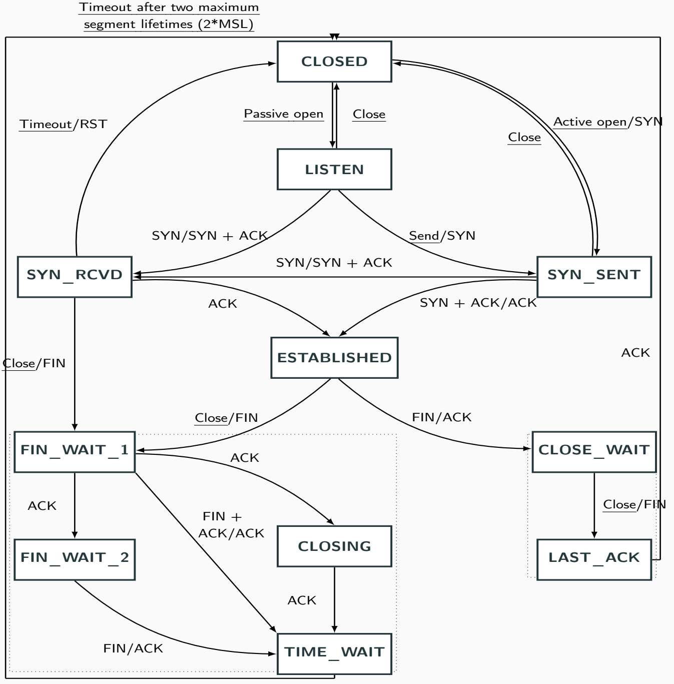
Plus complexe : ne doit pas être confondu avec une panne

Principe :

- je vais fermer / je sais que tu vas fermer
- je suis fermé / moi aussi
- je suis complètement fermé ...

Les numéros d'ordre ne changent pas pour éviter des problèmes "subtils".

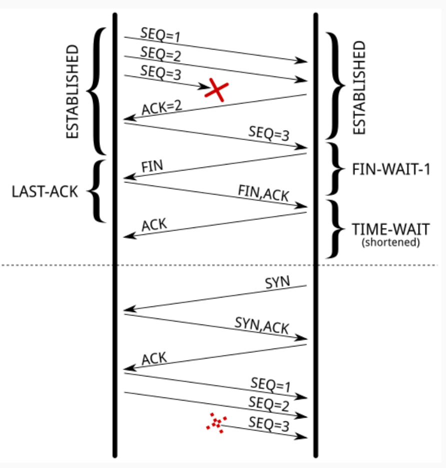

ESTABLISHED
close()
FIN_WAIT1
Segment X FIN
ESTABLISHED
CLOSE_WAIT
FIN_WAIT2
Segment Y ACK $X+1$
close()
LAST_ACK
Segment Y FIN
TIME_WAIT
Segment X ACK $Y+1$
CLOSED
Délai
CLOSED



Timeout after two maximum segment lifetimes (2*MSL)
CLOSED
Passive open
Close
Timeout/RST
Active open/SYN
Close
LISTEN
SYN/SYN + ACK
Send/SYN
SYN_RCVD
SYN/SYN + ACK
SYN_SENT
ACK
SYN + ACK/ACK
ESTABLISHED
ACK
Close/FIN
Close/FIN
FIN/ACK
FIN_WAIT_1
ACK
CLOSE_WAIT
ACK
FIN + ACK/ACK
Close/FIN
FIN_WAIT_2
CLOSING
LAST_ACK
ACK
FIN/ACK
TIME_WAIT

TIME_WAIT

Empêcher les segments en retard d'être acceptés dans une connexion utilisant le même quadruplet (adresse source, port source, adresse cible, port cible)



Rappels réseau

HTTP

HTTP

Créateur de HTTP : Tim Berners-Lee

Trois versions :

0.9	implantation originelle (1991)	abandonnée
1.0	RFC 1945 (1996)	obsolète
1.1	RFC 2616 (1999)	actuel

8 nouvelles RFC depuis 2014 (7230-7237)

Protocole Ascii 7 bits, sur TCP

Principe :

- le client se connecte, envoie une requête ;
- le serveur envoie une réponse puis déconnecte le client.

Version 1.1 : possibilité de maintenir la connexion (ouverte).

Structure d'une requête

```
Méthode URL HTTP/version <cr-lf>
Clé: valeur <cr-lf>
...
Clé: valeur <cr-lf>
<cr-lf>
Corps de la requête
```

Fin de l'entête : ligne vide (double `<cr-lf>`)

Retour chariot `<cr-lf>` : `\n` ou `\r\n`

Liste de (clé,valeur) : dictionnaire de propriétés

Version : 1.0 ou 1.1

Méthodes

- `GET` : demander une ressource
- `HEAD` : demander informations sur ressource
- `POST` : transmettre des informations
- `PUT` : transmettre une ressource par URL
- `PATCH` : modification partielle
- `OPTIONS` : obtenir les options de communication
- `CONNECT` : pour utiliser un proxy comme tunnel
- `TRACE` : écho de la requête
- `DELETE` : pour supprimer une ressource
- etc.

Propriétés

`Clé: valeur <cr-lf>`

- `Host` : site web demandé
- `Referer` : source du lien, fournie par le client
- `User-Agent` : navigateur utilisé
- `Date` : date génération réponse
- `Server` : serveur utilisé (Apache, etc.)
- `Content-Type` : type MIME (image/png, etc.)
- `Content-Length` : taille en octets
- `Expires` : date d'obsolescence, pour gestion du cache
- `Last-Modified` : date dernière modification
- etc.

Propriétés HTTP/1.1

`Clé: valeur <cr-lf>`

- `Connection` : la maintenir ou non
- `Accept` : type MIME acceptés
- `Accept-Charset` : encodages acceptés
- `Transfer-Encoding` : type d'encodage
- etc.

Méthode GET

```
GET URL HTTP/version <cr-lf>
Clé: valeur <cr-lf>
...
Clé: valeur <cr-lf>
<cr-lf>
```

Pas de corps de requête après l'entête.

Si la version est `1.1`, il doit y avoir la propriété

`Host:adresse-serveur[:port]` (par défaut 80)

Réponse HTTP

Une réponse HTTP est de la forme :

```
HTTP/version code explication <cr-lf>
Clé: valeur <cr-lf>
...
Clé: valeur <cr-lf>
<cr-lf>
Corps de la réponse
```

Codes :		
	20x	Succès
	30x	Redirection
	40x	Erreur du client
	50x	Erreur du serveur

```
$ netcat www.iut-fbleau.fr 80
TRACE /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Wed, 26 Feb 2020 17:44:02 GMT
Server: Apache/2.2.23 (Unix) mod_ssl/2.2.20 OpenSSL/1.0.0e DAV/2 SVN/1.7.6 PHP/5.4.8
Connection: close
Content-Type: message/http

TRACE /index.html HTTP/1.0
```

API Rest

Rest

Roy Fielding en 2000

- Representational State Transfer.
- Interface uniforme pour accéder à des ressources (n'importe quoi !).
  - Syntaxe universelle pour identifier des ressources : `URI` (endpoint)
  - Un ensemble bien défini d'opérations `GET, POST, PUT, DELETE` (fonctions basiques CRUD)

HTTP	CRUD	SQL
POST	Create	INSERT
GET	read	SELECT
PUT/PATCH	Update	UPDATE
DELETE	Delete	DELETE

  - Un ensemble de formats différents pour les données : `xml`, `json`, `csv`, `pdf`, ...

Avantages : abstraction, implantation et utilisation "simple" avec HTTP.

REST-ful vs REST-like

Une application est REST-ful (totalement REST) si les ressources sont accessibles par des représentions :

Restful

`http://airfrance.com/vols/123`

Cette ressource peut faire l'objet d'une requête `GET`, `PUT`, `PATCH` ou `DELETE`.

Certains services utilisent différentes urls pour des actions sur la même ressource (les actions sont dans l'url, approche RPC) :

Restlike

```
http://airfrance.com/vols/getvols?id=123
http://airfrance.com/vols/deletevols?id=123
```

- REST s'est imposait par rapport à XML-RPC, SOAP, etc.
- Beaucoup d'api de données, avec le format JSON.

Exemples

Création

Requête

```
POST /user
prenom=John&nom=Doe&age=25
```

Réponse du serveur

```
201 Created
Location: /user/123
```

Modification

Requête

```
PATCH /user/123
prenom=Johnny
```

```
PUT /user/123
prenom=Johnny&noDoe&age=25
```

Réponse du serveur

```
204 No Content
```

Exemples

Lecture

Requête

```
GET /user/123
```

Réponse du serveur

```
200 OK
<prenom>John</prenom>
<nom>Doe</nom>
<age>25</age>
```

Suppression

Requête

```
DELETE /usr/123
```

Réponse du serveur

```
204 No Content
```

Exemple avec php

URLS

Methode	URL	Action
GET	/todo	Récupére les todos
GET	/todo/id	Récupére le todo id
PUT	/todo/id	Modifie le todo id
DELETE	/todo/id	Efface le todo id
POST	/todo	Ajoute un todo

Format

Les Entrées/Sorties HTTP utiliseront le format JSON

table sql

Une seule table todo

name	type	NULL	default
id	int(11)	no	
title	varchar(256)	no	
done	tinyint(1)	no	0

Construction en PHP

But : fournir un service REST.

- souplesse
- abstraction par rapport au sgbd
- réutilisation
- http est ton ami (?)

Besoins techniques :

- Routage
- Filtrage des données
- entrées sorties HTTP
- Mise en cache
- etc.

Quelques micro-frameworks PHP

- `https://www.slimframework.com/`
- `http://flightphp.com/`

Certains framework sont capables de générer une api de données automatiquement.

Serveur http

Dans un premier temps, on va utliser le serveur apache (http) de dwarves.iut-fbleau.fr.

Dans votre répertoire, activer la réécriture d'urls :

```
Require method GET POST PUT PATCH DELETE OPTIONS
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php [QSA,L]
```

Utilisation de flight

Micro framework https://flightphp.com/

Pour le routage et les entrées/sorties http.

Utilisation comme une classe globale statique,

```php
<?php
require 'flight/Flight.php';

Flight::route('/', function(){
    echo 'hello world!';
});

Flight::start();
```

ou comme une instance d'objet

```php
require 'flight/autoload.php';

use flight\Engine;

$app = new Engine();

$app->route('/', function(){
    echo 'hello world!';
});

$app->start();
```

Enregistrement des routes, et handlers associés.

```php
<?php

Flight::route('GET /todo(/@id)','get_todo');
Flight::route('POST /todo','add_todo');
Flight::route('DELETE /todo/@id','delete_todo');
Flight::route('PUT /todo/@id','update_todo');
```

Partie Modèle

Utilisation de l'orm (object relational mapping) php RedBean :

https://www.redbeanphp.com/

```php
<?php
require 'rb-mysql.php';

$mysql = 'mysql:host=localhost;dbname=todo','user', 'mdp';

R::setup($mysql);
R::useFeatureSet( 'novice/latest' );
```

Et c'est tout ! On peut alors utiliser le modèle avec flight pour les entrées/sorties mysql.

Le code pour la route `get /todo` :

```php
<?php
require 'flight/Flight.php';
require 'model/model.php';
// classes static Flight et R
//
Flight::route('GET /todo(/@id)','get_todo');

function get_todo($id = null)
{

  if (!isset($id)){
    Flight::json(["results" => R::findAndExport('todo')]);
  }else{
    $todo = R::load( 'todo', $id  );
    if ($todo->id)
      Flight::json($todo->export());
    else
      Flight::halt(404);
  }
}
```

Application JS

Modification de la seule partie `model.js` pour utiliser l'api de données avac AJAX.

Firebase

Fonctionnalités

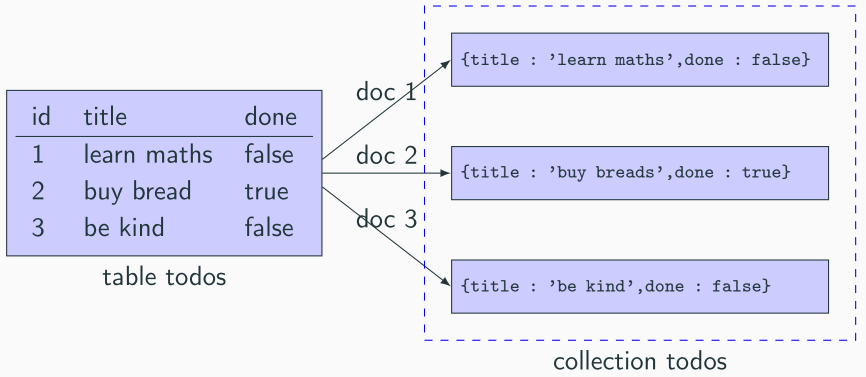
Firebase fournit, entre autres :

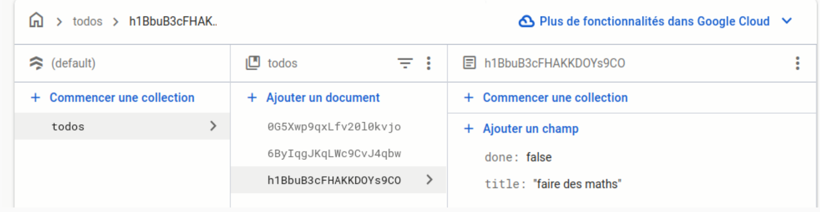
- un service d'authentification,
- des services de bases de données noSQL :
  - Cloud Firestore (collection de documents)
  - Realtime Database (un seul arbre JSON)
- Fonctions à distance,
- etc.

Tout le reste concerne les api web en javascript.

Firestore

Collections de documents.





On peut imbriquer des collections dans des documents.

ajout de données

```
import { doc, setDoc } from "firebase/firestore";

// Add a new document in collection "cities"
await setDoc(doc(db, "cities", "LA"), {
  name: "Los Angeles",
  state: "CA",
  country: "USA"
});

const cityRef = doc(db, 'cities', 'BJ');
await setDoc(cityRef, { capital: true }, { merge: true });
```

```javascript
import { doc, setDoc, Timestamp } from "firebase/firestore";

const docData = {
    stringExample: "Hello world!",
    booleanExample: true,
    numberExample: 3.14159265,
    dateExample: Timestamp.fromDate(new Date("December 10, 1815")),
    arrayExample: [5, true, "hello"],
    nullExample: null,
    objectExample: {
        a: 5,
        b: {
            nested: "foo"
        }
    }
};
await setDoc(doc(db, "data", "one"), docData);
```

Génération automatique d'un id pour un document.

```
import { collection, addDoc } from "firebase/firestore";

// Add a new document with a generated id.
const docRef = await addDoc(collection(db, "cities"), {
  name: "Tokyo",
  country: "Japan"
});
console.log("Document written with ID: ", docRef.id);
```

Mise à jour de données

```
import { doc, updateDoc } from "firebase/firestore";

const washingtonRef = doc(db, "cities", "DC");

// Set the "capital" field of the city 'DC'
await updateDoc(washingtonRef, {
  capital: true
});
```

```
import { doc, setDoc, updateDoc } from "firebase/firestore";

// Create an initial document to update.
const frankDocRef = doc(db, "users", "frank");
await setDoc(frankDocRef, {
    name: "Frank",
    favorites: { food: "Pizza", color: "Blue", subject: "recess" },
    age: 12
});

// To update age and favorite color:
await updateDoc(frankDocRef, {
    "age": 13,
    "favorites.color": "Red"
});
```

Mise à jour dans un tableau

```
import { doc, updateDoc, arrayUnion, arrayRemove } from "firebase/fir

const washingtonRef = doc(db, "cities", "DC");

// Atomically add a new region to the "regions" array field.
await updateDoc(washingtonRef, {
    regions: arrayUnion("greater_virginia")
});

// Atomically remove a region from the "regions" array field.
await updateDoc(washingtonRef, {
    regions: arrayRemove("east_coast")
});
```

Suppression

```
import { doc, deleteDoc } from "firebase/firestore";

await deleteDoc(doc(db, "cities", "DC"));
```

Lecture

```javascript
import { doc, getDoc } from "firebase/firestore";

const docRef = doc(db, "cities", "SF");
const docSnap = await getDoc(docRef);

if (docSnap.exists()) {
  console.log("Document data:", docSnap.data());
} else {
  // docSnap.data() will be undefined in this case
  console.log("No such document!");
}
```

Requêtes

```
import { collection, query, where, getDocs } from "firebase/firestore

const q = query(collection(db, "cities"), where("capital", "==", true

const querySnapshot = await getDocs(q);
querySnapshot.forEach((doc) => {
  // doc.data() is never undefined for query doc snapshots
  console.log(doc.id, " => ", doc.data());
});
```

filtres `where`, `orderBy`, `limit`

Observateurs

On peut écouter les modifications sur un ou plusieurs documents

```
import { doc, onSnapshot } from "firebase/firestore";

const unsub = onSnapshot(doc(db, "cities", "SF"), (doc) => {
    console.log("Current data: ", doc.data());
});
```