Conception Et Programmation Web

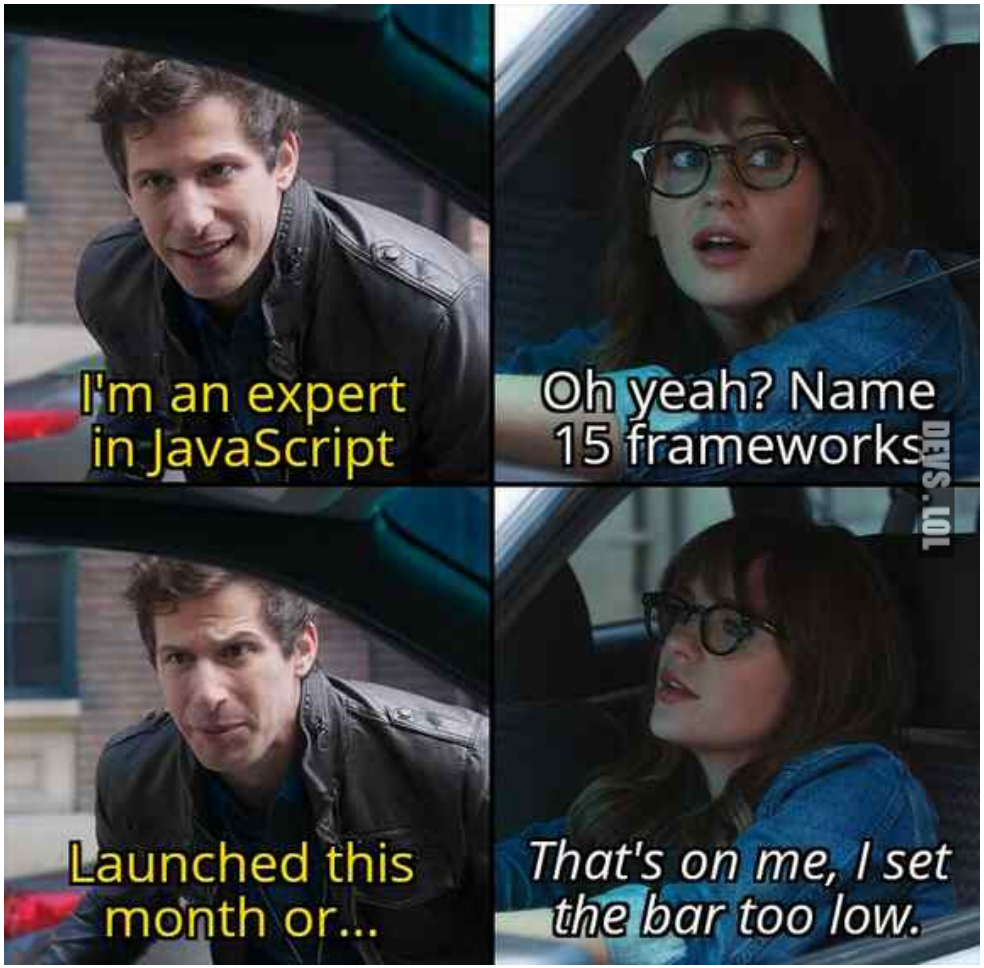
LIFWEB - http://lifweb.pages.univ-lyon1.fr/

Semestre printemps 2024-2025
L3 - UCBL

Aurélien Tabard - https://tabard.fr/, basé sur les cours de Romuald Thion

Applications et Frameworks Web

Aurélien Tabard - 2024-2025


I'm an expert in JavaScript
Oh yeah? Name 15 frameworks
Launched this month or...
That's on me, I set the bar too low.
DEVS.101

Source

Applications et Frameworks Web
Plan

1. Pourquoi des frameworks ?
2. Structure des frameworks
3. Ou placer la gestion de la Vue (du MVC ?)
4. Frameworks web Node.js

Que doit gérer une application Web ?

TCP ▶️ HTTP ▶️ APPLICATION ▶️ HTTP ▶️ TCP

Il faut implémenter (tout ou partie de) la spécification HTTP
-> https://httpwg.org/specs/

Le protocole de base est simple (non connecté, texte)

- mais les fonctionnalités sont nombreuses !

Les classes suivantes représentent les échanges :

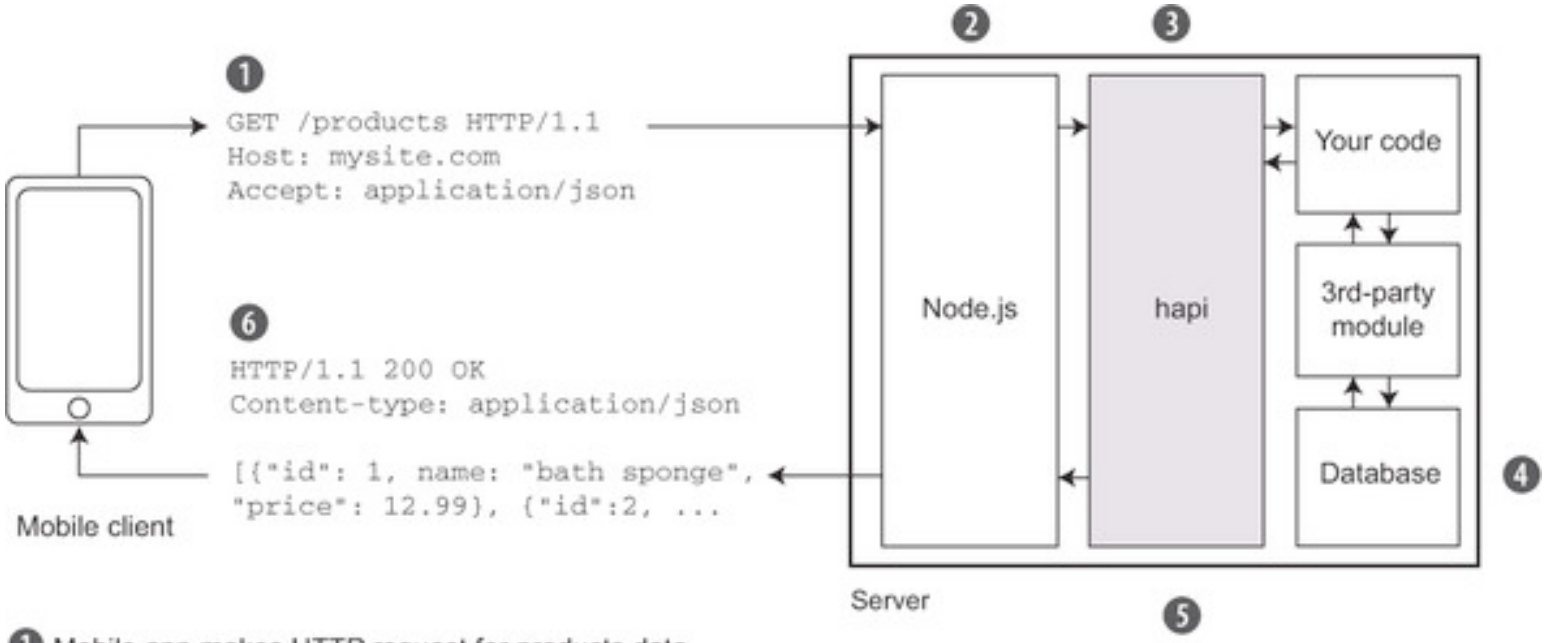
- http.ClientRequest
- http.ServerResponse

💡

Des frameworks d'applications existent dans tous les écosystèmes pour faciliter le développement d'appli Web :
Flask ou FastAPI en Python,
Jakarta/Tomcat en Java,
Dream en OCaml, Rust, etc.

Le framework des TP à venir

hapi

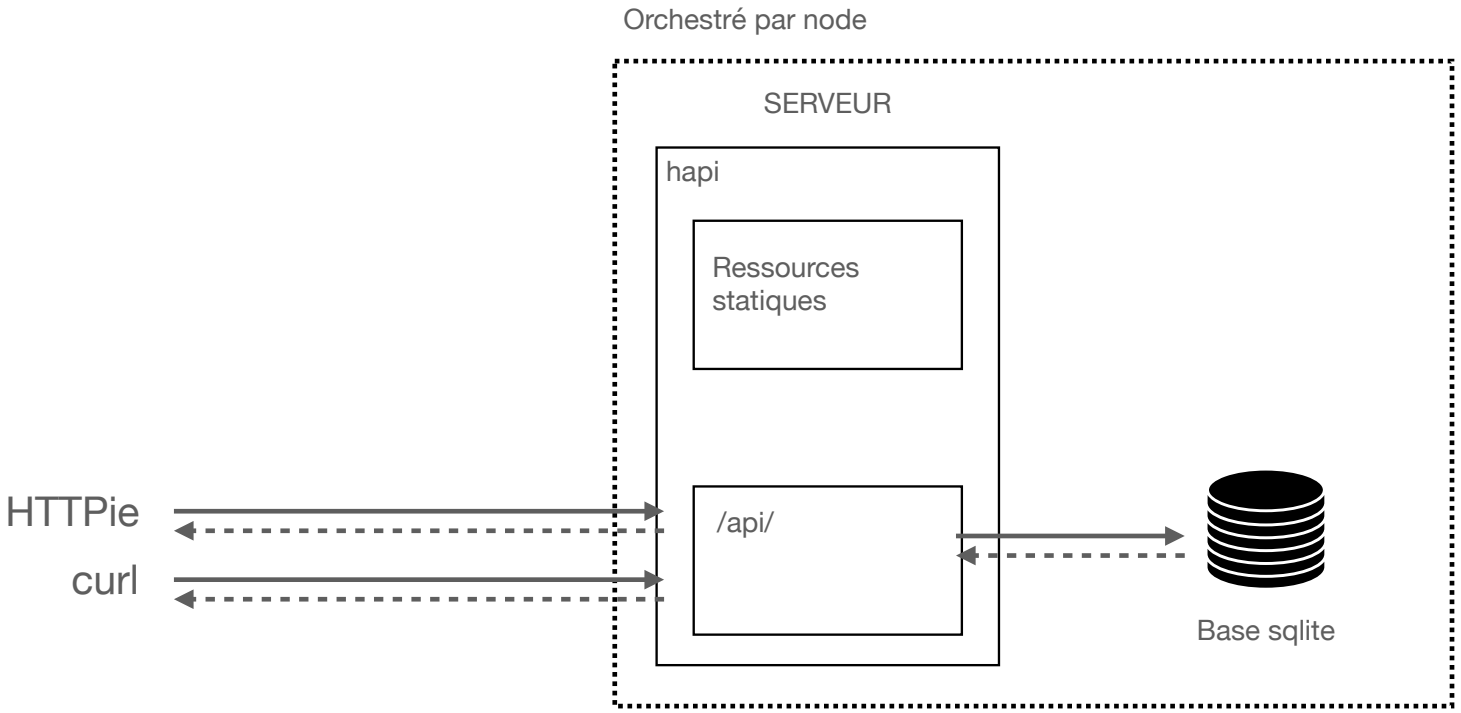


1. Mobile app makes HTTP request for products data.
2. Request is received by Node and forwarded to hapi.
3. hapi authenticates user and routes request to appropriate function in your code.
4. Your app gets products data from database.
5. Products data is given to hapi's `reply()` function. hapi validates and caches the output if configured to.
6. HTTP response is sent from Node to the mobile application.

https://livebook.manning.com/book/hapi-js-in-action/chapter-1/16

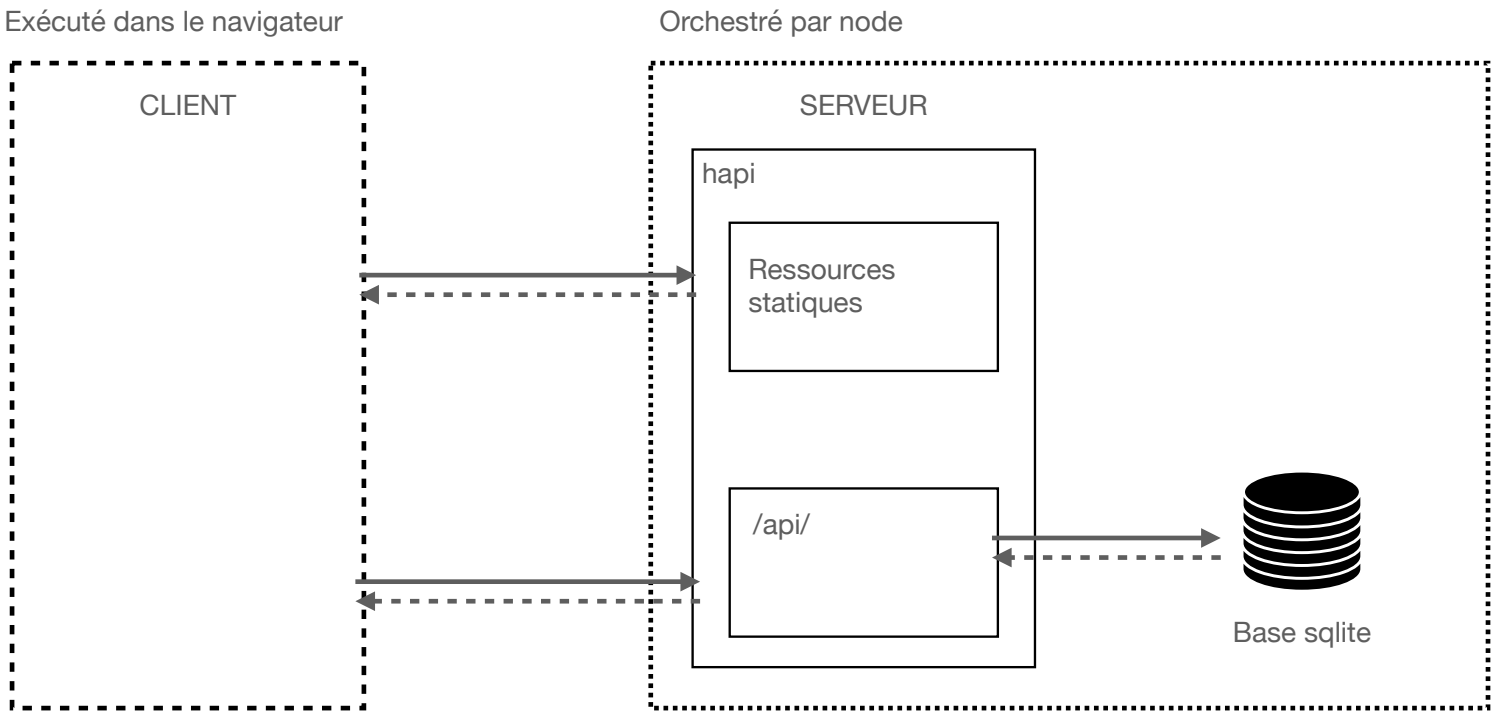
L’architecture du TP5

TP5a



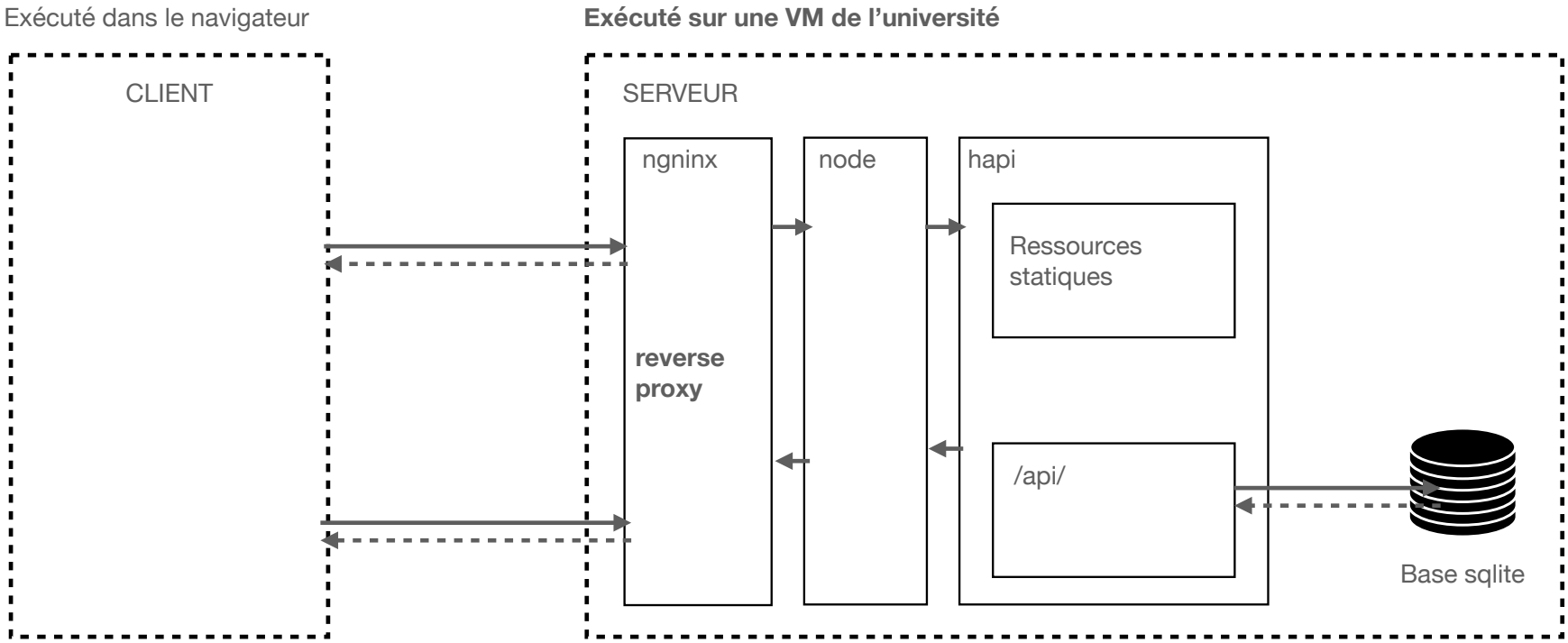
L’architecture du TP5

TP5a + TB5b



L’architecture du TP5

TP5a + TB5b + TP5c



Que doit gérér une application ?

HTTP ▶️ Application

```
GET /api HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

Analyser la requête : verbe, chemin et version.

Analyser les en-têtes :

- Host: pour multiplexer
- If-Modified-Since: pour HTTP 304

Désérialiser le body, etc.

📄 Opérations encapsulées dans un objet Request (celui de Node.js ou propre). 📄

Que fait un framework ?

HTTP ▶️ Application ▶️ HTTP

Il traite les requêtes HTTP pour produire les réponses HTTP associées.

```
// on étend (déclarativement) le serveur avec une route
server.route({
  // verbe et chemin extraits de la requête
  method: "GET",
  path: "/",
  // fonction de traitement qui prend la requête et calcule
  // une proto réponse qui sera complétée puis envoyée
  handler: (request, h) => {
    const name = request.query.name ?? "World";
    // dans ce framework on return simplement,
    // dans d'autre on configure un objet réponse
    return { message: `Hello ${name}!` };
  },
});
```

Exemple avec https://hapi.dev/ (source).

Que fait un framework ?

Application ▶️ HTTP (1/2)

Le framework construit l'objet `Response`, et conduit plusieurs opérations avant d'émettre sur le réseau :

- Positionner le code de retour (2xx, 3xx, 4xx, 5xx)
- Générer les *headers*
  - Calculer Content-Length et Etag (empreinte)
  - Positionner l'objet *cookie*
- **Sérialiser le body**, etc.

☣️ Erreur classique : essayer de modifier la réponse après envoi. ☣️

Que fait un framework ?

APPLICATION ▶️ HTTP (2/2)

👉 L'application calcule le code de retour, une partie des en-têtes et le contenu (body), le framework se charge de transformer en réponse HTTP.

```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html>… (here come the 29769 bytes of the requested web page)
```

Services offerts par les frameworks
1/2

🖐 Avec la gestion des requêtes et réponses HTTP, les frameworks proposent des services complémentaire pour assurer certaines fonctionnalités :

- routage (quasi systématique),
- logging et monitoring,
- validation des E/S,
- authentification et autorisation,
- persistance (e.g., ORM),

Services offerts par les frameworks
2/2

- cache,
- sessions et cookies,
- moteurs de vues,
- fichiers statiques,
- sécurité (e.g., CSRF).

👉 Les frameworks minimalistes et unopiniated proposent peu de services, les frameworks batteries included en proposent un plus grand nombre déjà intégrés.

Applications et Frameworks Web
Plan

1. Pourquoi des frameworks ?
2. Structure des frameworks
3. Ou placer la gestion de la Vue (du MVC ?)
4. Frameworks web Node.js

Structure des Frameworks

Source

Design patterns

Issus de la programmation orientée objet :

- Tendance à combiner les design patterns
  - e.g. en Jakarta EE Core J2EE Patterns.

Présents en programmation fonctionnelle :

- Plutôt sous forme algébrique
  - e.g., *monades* ou *gestionnaires d'effets*.

👉 Les *frameworks* s'appuient sur des principes de génie logiciel ou *patterns* assez caractéristiques.

👉 JavaScript, fonctionnel **et** objet, intègre des *design patterns* dans un style idiomatique. Voir Do you need Design Patterns in Functional Programming?.

Inversion de contrôle

👉 Les frameworks s’appuient en particulier sur l’inversion de contrôle.

- Ç’en est une caractéristique (par rapport aux bibliothèques)

⚙️ On paramètre le framework par différents points d’extensions

- Les traitements applicatifs sont “ancrés” à ces points.

🏁 Le programme principal (dit aussi point d’entrée, main en C/C++) laisse le contrôle du flot d’exécution au framework.

Inversion de controle

💡 C’est le framework qui appelle nos handlers…
…pas le contraire (approche bibliothèque) ! 💡

```
// création de l'application par le framework
const server = Hapi.server({ port });

/* configuration en utilisant server.route() */

// on laisse la main au flot d'exécution principal
await server.start();
```

Point d’entrée d’une application Hapi (source).

“One important characteristic of a framework is that the methods defined by the user to tailor the framework will often be ***called from within the framework itself***, rather than from the user’s application code.”

“The framework often ***plays the role of the main program*** in coordinating and sequencing application activity. This inversion of control gives frameworks the power to serve as extensible skeletons.

The methods supplied by the user ***tailor the generic algorithms*** defined in the framework for a particular application.”

Ralph Johnson and Brian Foote

Organisation de l’application

🤵 Le contrôle est inversé : c’est le framework qui contrôle l’appel des fonctions qu’on lui fournit.

❓ Comment définir et passer les traitements de l’application qu’on développe ?

- À travers des méthodes/constructeurs/interfaces
  - Par injection de dépendances,
- Via les listeners d’un design pattern Observer,
- Par style à passage de continuations (CPS),
  - Possiblement encapsulé par Promise
- Par configuration (e.g., fichier XML).

Organiser les traitements de l’application

💡 Quels que soient les langages et les architectures logicielles,
il faut pouvoir organiser les traitements de l’application,
notamment les séquencer :

1. Authentifier l’utilisateur (401 sinon)
2. Vérifier ses droits (403 sinon)
3. Vérifier la structure de la requête (400 sinon)
4. Calculer la réponse (500 possible)
5. Émettre la réponse HTTP (200)
6. Écrire dans le journal (log)

💡 Mais aussi factoriser les traitements,
e.g., authentification et autorisation sur admin/**.

Stratégie de séquencement

👉 Deux grandes stratégies de séquencement explicite et réutilisation :

1. Par chaînage de fonctions.
2. Selon le cycle de vie des requêtes et réponses HTTP.

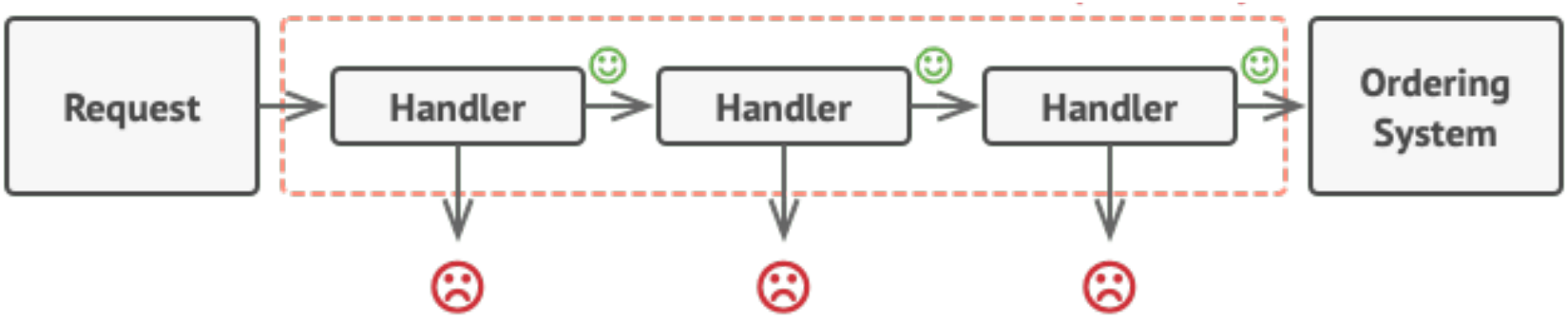
Stratégie chainage de fonctions

💡 Principe du design pattern chain of responsabilities pour regrouper les traitements en unités indépendantes et réutilisables.

Typique de Express et Koa

Les fonctions sont appelées *middlewares* et sont exécutées en séquence **dans l'ordre de définition**.

Patron Chain of Responsibility



Problème : l'ordre des déclarations impacte trop fortement le séquencement, elles sont difficiles à ordonner.

Composition de fonction — Définition

Soit une séquence **functs** $= [f_1, \ldots, f_n]$:

- de fonctions f_i de type $a \to (a \to r) \to r$
- a est le type du paramètre principal des f_i
  - exemple : `Request`, `Response`, `Toolkit` ou `Context` selon le *framework*
- $a \to r$ est le type du *callback*

Étant donnés un premier $x : a$ et un dernier *callback* **last** on veut chaîner les fonctions en calculant :

$$f_1(a, x_1 \mapsto f_2(x_1, x_2 \mapsto \ldots x_{n-1} \mapsto f_n(n-1, \textsf{last})))$$

Composition de fonction CPS

Style impératif

```
function composeCPS(functs) {
  return function (x, last) {
    let next = last;
    // right to left
    for (let index = functs.length - 1; index >= 0; index--) {
      const previous = next; //closure
      next = (x) => functs[index](x, previous);
    }
    return last(next(x));
  };
}
```

Composition de fonction CPS

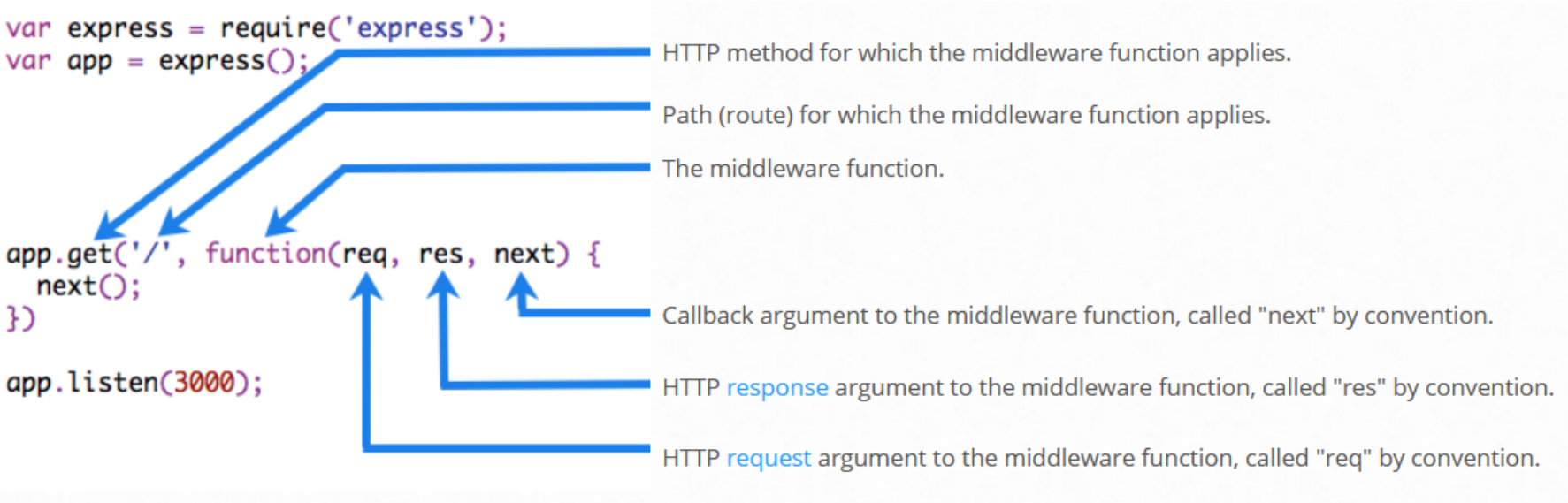
Style fonctionnel

```
// addition en CPS avec "trace" des valeurs
const add = (k) => (x, next) => (console.log(x, next), next(k + x));
const composeCPS = (functs) => (x, last) =>
  functs.reduceRight((next, f) => (x) => f(x, next), last)(x);
```

```
composeCPS([add(3), add(2), add(1)])(12, console.log);
12 [Function: next]
15 [Function: next]
17 [Function: log]
18
```

📄 Voir chain-of-functions.js. 📄

Continuation next() à la express



`app.get()` ajoute à la séquence de handlers, `next()` est le callback CPS de chaque handler. L'application Express va les chaîner (Writing middlewares).

Exemple configuration Tomcat

👉 On *déclare* un filtre (basé sur la classe `jakarta.servlet.GenericFilter`) et où l'appliquer (source).

```
<filter>
    <filter-name>RateLimitFilter Global</filter-name>
    <filter-class>org.apache.catalina.filters.RateLimitFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>RateLimitFilter Global</filter-name>
    <url-pattern>*</url-pattern>
</filter-mapping>
```

Exemple: filtre Tomcat

👉 La méthode `FilterChain.doFilter()` est l'équivalent objet du *callback* `next()` d'Express (source).

```
public class VisitorCounterFilter implements Filter {
    private static Set<String> users = new HashSet<>();
    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
      FilterChain chain) {
        HttpSession session = ((HttpServletRequest) request).getSession(false);
        Optional.ofNullable(session.getAttribute("username"))
          .map(Object::toString)
          .ifPresent(users::add);
        request.setAttribute("counter", users.size());
        chain.doFilter(request, response);
    }
}
```

Stratégie « cycle de vie »

Utilisé par Fastify et Hapi.

Le flot HTTP suit une série d'étapes fixes :

- Déclenchement géré par le framework
- Ces étapes constituent le cycle de vie.

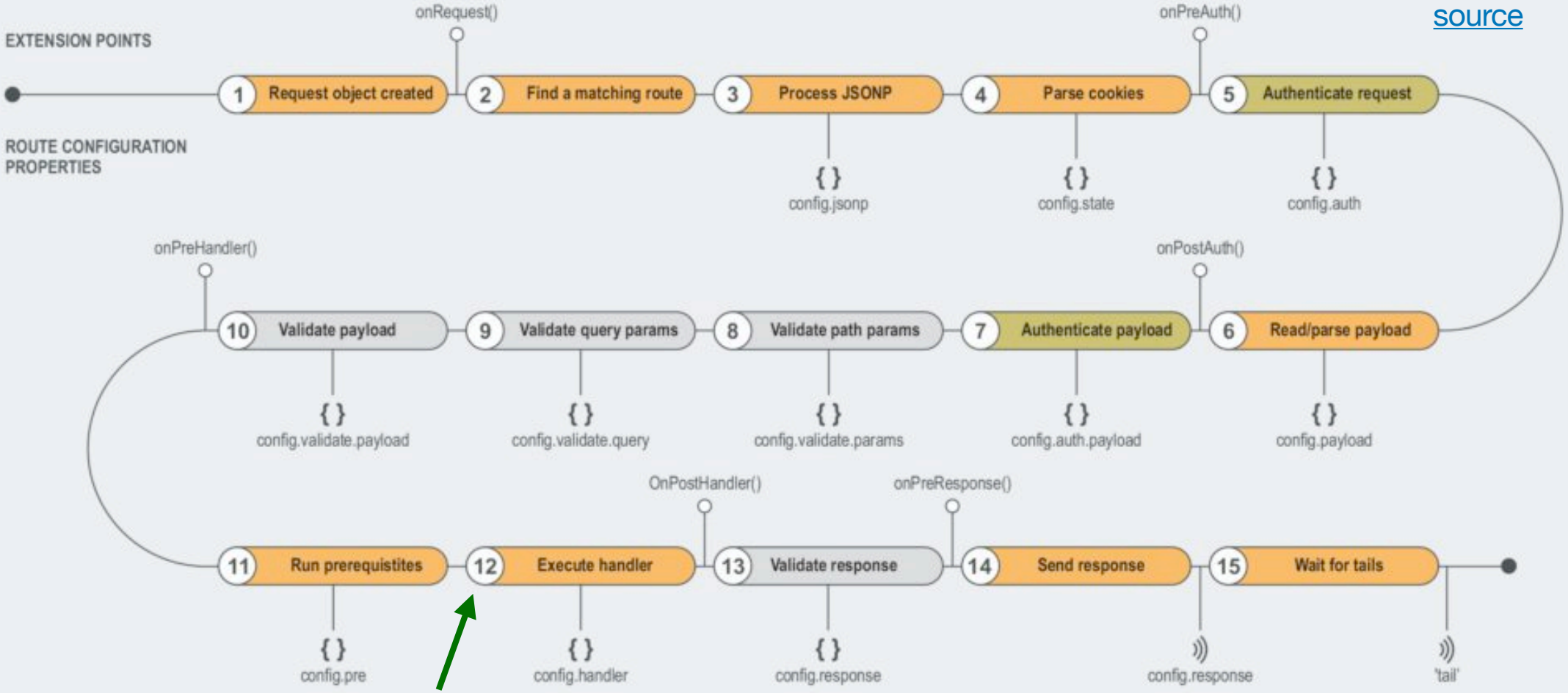
On associe des traitements aux étapes :

- On appelle parfois ces handlers des hooks (voir SO).

💡 C'est une instance du design pattern Observer, comme les EventEmitter, avec un ordre de déclenchement fixé. 💡

Cycle de vie en Hapi

source



Là où le principal *handler* utilisateur est exécuté.

Exemple d’extension hapi

Calcul le temps de traitement à l’échelle du serveur

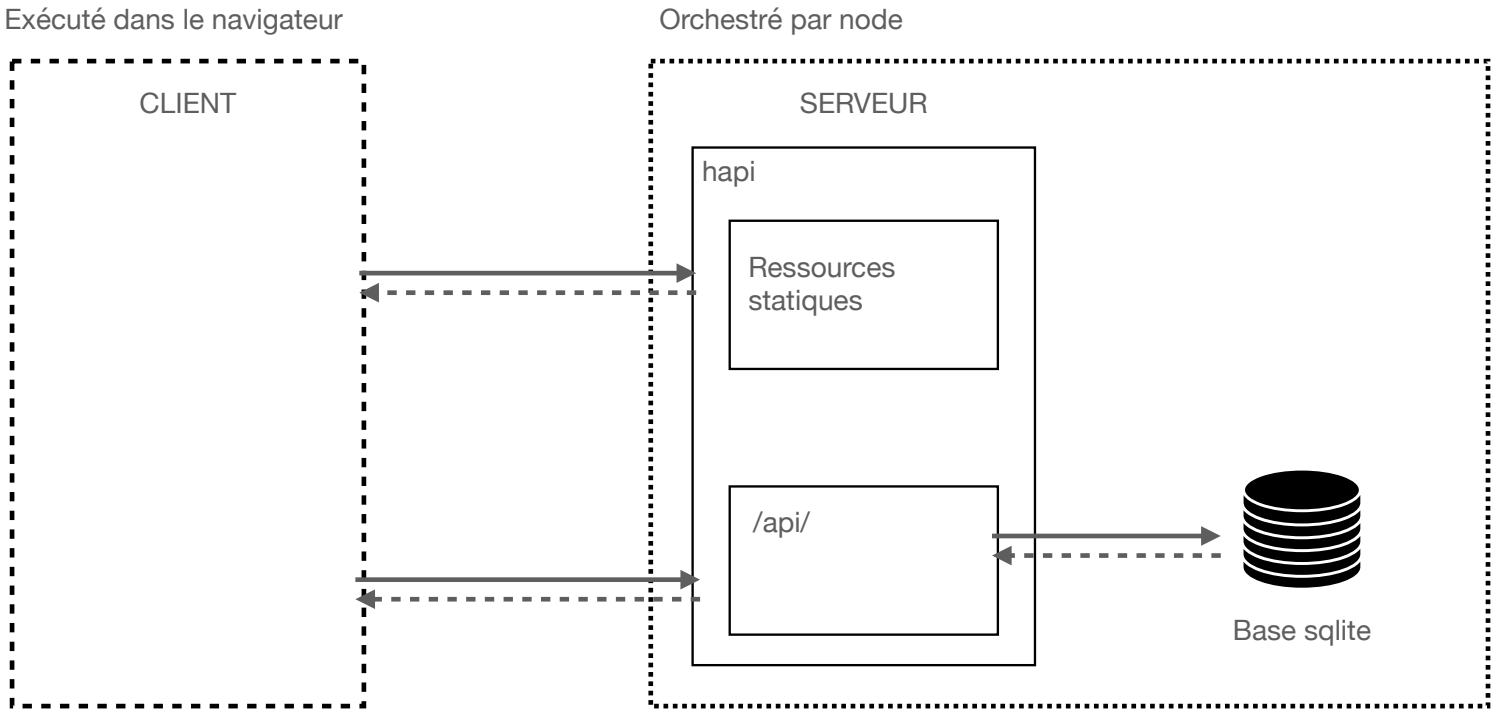
```
server.ext("onRequest", (request, h) => {
  request.app.startTime = process.hrtime.bigint();
  return h.continue;
});

server.ext("onPreResponse", (request, h) => {
  request.app.endTime = process.hrtime.bigint();
  request.response.header(
    "X-Response-Time",
    `${(request.app.endTime - request.app.startTime) / 1000n}`,
  );
  return h.continue;
});
```

Applications et Frameworks Web
Plan

1. Pourquoi des frameworks ?
2. Structure des frameworks
3. Ou placer la gestion de la Vue (du MVC ?)
4. Frameworks web node.js

Quand et où générer la vue ? i.e. le document HTML

Architecture des TP5a et TB5b



Server Side Rendering

🏛️ Applications Web dites classiques

Rendu côté serveur :

- Utilise un moteur de template (voir CM2).
- Générer le HTML a un coût.

Code client simplifié :

- Interaction utilisateur principalement.
- Peu de modifications du DOM

👍 Avantage pour le référencement (SEO – Search Engine Optimization, voir Google's SEO starter guid), pour le chargement de la première page, sur les vieux téléphones…

Client side rendering

 Applications dites AJAX, ou Web 2.0 ou SPA (Single Page Applications)

Rendu côté client

- Construit le DOM grâce aux JSON du serveur,
- Le HTML est un squelette
  - Potentiellement <body id="app"></body>

Découple complètement le rendu du serveur.

- Le serveur est réduit à une API Web.
- Code et charge réduite.

Voir, par exemple Client-side Vs. Server-side Rendering: What to choose when?

Client side rendering

Avantages/inconvénients

👍 Applications utilisables offline, limite les transferts HTTP, permet le développement de clients riches.

👍 Peuvent être servies par un autre serveur que celui qui fourni les données (voir autorisation CORS).

👎 La logique métier est fractionnée, voire dupliquée, entre client et serveur.

👉 Des frameworks MVC JS souvent utilisés pour gérer la complexité.

Applications et Frameworks Web
Plan

1. Pourquoi des frameworks ?
2. Structure des frameworks
3. Ou placer la gestion de la Vue (du MVC ?)
4. Frameworks web Node.js

Les frameworks Node

Framework	GitHub	Stars	Down/week	Année
Express	GitHub	62.5k	31.4M	2009
Koa	GitHub	34.5k	1.6M	2013
Fastify	GitHub	29.4k	1.8M	2016
Hapi	GitHub	14.4k	0.8M	2011

Voir par exemple Choosing the right Node.js Framework: Express, Koa, or Hapi?

Express

Voir server-express.js

“Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.”

✅ Référence historique (👴 API historique aussi)

✅ Documentation et communauté

❎ Configuration des chaînes de middlewares

❎ Problèmes de maintenance dans le passé

Bench : 2 ms (97.5%) et 0.83 ms (x̄), 7,850 R/S (x̄)

Koa

Voir server-koa.js

“Koa is developed by the same team behind Express.js. Referred to as a lighter version of Express.”

✅ API Express, rafraîchie, voir Koa vs Express.

❎ Minimaliste, pas de routage :

- https://github.com/koajs/router (822 stars)
- https://github.com/koajs/route (441 stars)

❎ Communauté réduite

Bench : 0 ms (97.5%) et 0.01 ms (x̄), 29,769 R/S (x̄).

Fastify

Voir server-fastify.js

"Fastify is a web framework highly focused on providing the best developer experience […]. It is inspired by Hapi and Express and […] is one of the fastest web frameworks in town."

✅ Orienté vers la performance

✅ Extension par hooks, all included (e.g., validation)

❌ Trop dynamique, architecture de plugins complexe

❌ Documentation aride, variable selon les plugins

Bench : 0 ms (97.5%) et 0.01 ms ($\bar{x}$), 32,181 R/S ($\bar{x}$)

hapi.dev

Voir server-hapi.js

“Build powerful, scalable applications, with minimal overhead and full out-of-the-box functionality - your code, your way Originally developed to handle Walmart’s Black Friday scale,[…]”

✅ Extension par hooks

✅ Écosystème complet et homogène

❎ Prisonnier de l’écosystème ad hoc

Voir Express Migration

Bench : 0 ms (97.5%) et 0.03 ms (x̄), 22,091 R/S (x̄)

Applications et Frameworks Web
Plan

1. Pourquoi des frameworks ?
2. Structure des frameworks
3. Ou placer la gestion de la Vue (du MVC ?)
4. Frameworks web Node.js

TP5
Plan

TP en 3 parties (sur 4 séances)

- 2 séances hapi -> serveur local
- 1 séance interfaçage client-serveur
- 1 séance déploiement

Démo lors de la séance du 15 avril

CC3

Mardi prochain 1/04 - salle à venir (probablement Thémis) -> email avec détails

8h30 (installation) - 8h45 (démarrage) -> 9h15 (fin) - 9h30 (tiers-temps)

Sujets couverts

- Bonnes pratiques JavaScript
- Programmation fonctionnelle
- JavaScript asynchrone
- Manipulation du DOM
- Node.js / hapi