

Ecole Centrale de Lyon

UE INF tc3

Projet d'Application Web

*Applications web,
contexte, principes et architecture*

René Chalon
Rene.Chalon@ec-lyon.fr

Daniel Muller
Daniel.Muller@ec-lyon.fr



2. Applications web, contexte, principes et architecture

Plan de la séance

- 2.1 Éléments d'architecture
 - 2.1.1 Architecture client-serveur
 - 2.1.2 Architecture Web
 - 2.1.3 Architecture trois tiers
 - 2.1.4 Interfaces et interactions utilisateur
 - 2.1.5 Données persistantes, stockage et accès
 - 2.1.6 Client léger
 - 2.1.7 Client riche
 - 2.1.8 Application isomorphique
 - 2.1.9 AJAX
 - 2.1.10 Contenu de la réponse
 - 2.1.11 Principes du protocole HTTP
 - 2.1.12 Architecture REST
- 2.2 Contexte
 - 2.2.1 Diversité des terminaux
 - 2.2.2 Mobilité
 - 2.2.3 Contenus multimédia
 - 2.2.4 Fonctionnalités des applications natives
- 2.3 Comment aborder cette complexité ?
 - 2.3.1 Décomposition des tâches
 - 2.3.2 Les métiers du Web
 - 2.3.3 Frameworks
 - 2.3.4 Gestion de projet informatique



2.1 Éléments d'architecture

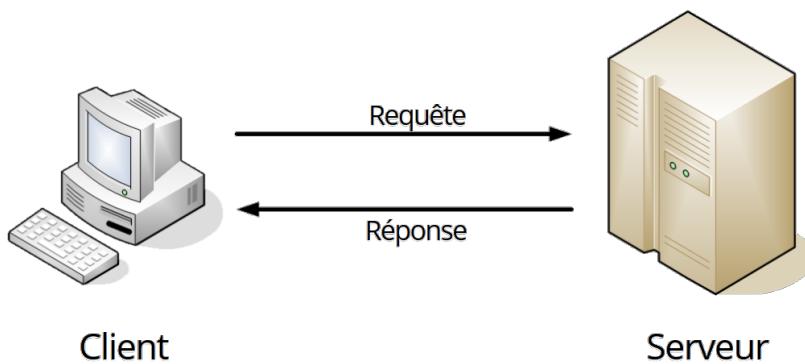
2.1.1 Architecture client-serveur

Rappel :

« Une **Application Web** (aussi appelée WebApp) est un logiciel applicatif manipulable grâce à un navigateur Web. » [\[Wikipédia\]](#)

Cette définition sous-entend une **architecture distribuée** qui met en oeuvre :

- d'un côté un navigateur web (*ou client*),
- de l'autre un serveur,

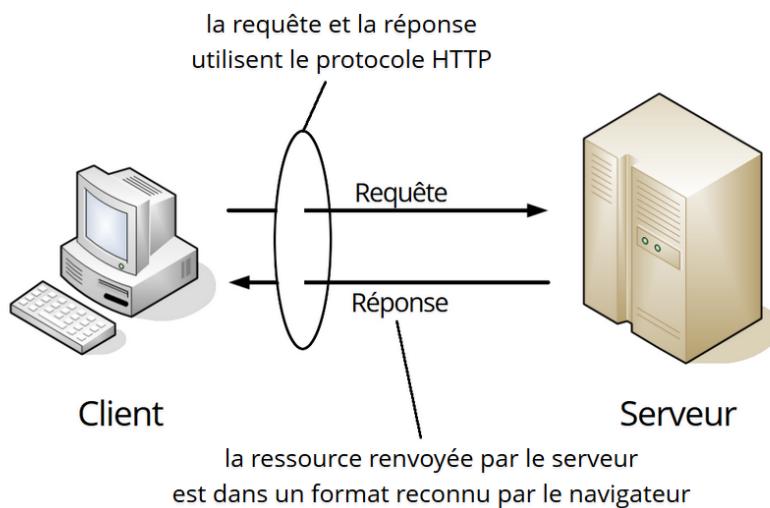


encore appelée **client-serveur**.

2.1.2 Architecture Web

Comme il s'agit d'une **application Web**, cela signifie que :

- le client utilise le **protocole** du Web (*HTTP - HyperText Transfer Protocol*) pour demander une **ressource** au serveur.
- Le serveur renvoie une représentation de la ressource dans l'un des nombreux formats reconnus sur le Web (*HTML, CSS, PNG, JPEG, GIF, SVG, MP3, MP4, Javascript, JSON, XML...*).



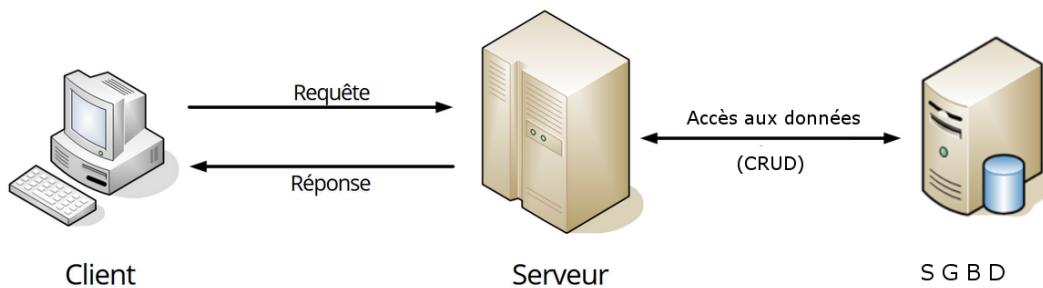


2.1.3 Architecture trois tiers

Outre :

- le client qui assure les interactions avec l'utilisateur,
- et le serveur web (*ou serveur d'applications*) qui contrôle habituellement l'enchaînement des opérations,

les applications s'appuient souvent sur un système de gestion de base de données (*relationnelle ou non*) pour le stockage des données persistantes.



On parle dans ce cas d'une **architecture trois tiers** ou à trois niveaux, appelés :

- couche présentation,
- couche métier,
- et couche d'accès aux données.

2.1.4 Interfaces et interactions utilisateur

Le client :

- présente à l'utilisateur les interfaces (*textuelles ou graphiques*),
- et gère ses interactions (*saisie clavier, souris, tactile...*) avec l'application.

En termes techniques on dira que le client abrite les **IHM** (*Interactions Humain-Machine*).



Source : Charles Martin (Own work) [GFDL (<http://www.gnu.org/copyleft/fdl.html>)], via Wikimedia Commons



Exemples d'interfaces

The screenshot shows a product page for the book "JavaScript: The Good Parts" by D. Crockford. The page includes the book cover, a brief description, customer reviews, and purchase options. A sidebar on the right shows a "Buy New" option at EUR 22,92 and a "Buy Used" option at EUR 18,72.

Recopie d'écran: amazon.com

The screenshot shows the "Comptes et contrats" section of the Banque Populaire website. It displays a table of accounts with columns for N° compte, Type, Intitulé personnalisable, Solde (EUR), and Actions. The table shows one account with a balance of 0.00.

Recopie d'écran: Banque Populaire

The screenshot shows a login form for the Central Authentication Service of Ecole Centrale Lyon. It requires a Username and Password, and includes a checkbox for "Warn me before logging me into other sites." and a "Login" button. A note at the bottom asks users to log out and exit their browser after use.

Recopie d'écran: Centrale Lyon



The screenshot shows the Zimbra Webmail interface. The left sidebar includes sections for 'Dossiers de mails', 'Reception (1325)', 'Trash (12)', and 'Tags (libellés)'. A calendar for October 2015 is also visible. The main area displays an inbox with several messages. One message from 'Younes Gonzalez' is selected, showing a preview of a document titled 'facture_csc.admin-58926058.doc' (40,8 ko). The message content discusses a bill for 512,36 euros.

Recopie d'écran: zimbra webmail

The screenshot shows the Dropbox interface. On the left, there's a sidebar with links like 'Fichiers', 'Équipe', 'Paper', 'Photos', 'Partage', 'Liens', 'Événements', 'Demandes de fichiers', 'Prise en main (4)', and 'Fichiers supprimés'. The main area lists shared folders with details such as 'Nom', 'Modifié', and 'Partagé avec'. Examples include 'BE-Symfony2', 'CLEM', 'INF_tc4', and 'PE Application Challenge 2016'. Each folder entry shows a list of users who have access to it.

Recopie d'écran: dropbox.com

On peut à titre d'exercice identifier les interfaces qui proposent :

- des zones de texte, icônes ou boutons cliquables,
- des listes de choix, des cases à cocher,
- des menus popup, des zones à survoler, des barres de menu, des onglets,
- des zones de texte à remplir, du glisser-déposer...

2.1.5 Données persistantes, stockage et accès

La plupart des applications s'appuient côté serveur sur un mécanisme de stockage de données. La nature des données stockées est très variable d'une application à l'autre. Quelques exemples :

- paramètres liés à chaque utilisateur (*profil, historique, actions en cours...*),
- statistiques et éléments d'analyse concernant l'utilisation de l'application automatiquement renseignées au fil de l'eau,
- données métiers (*horaires des trains pour la SNCF, état des stocks pour un magasin en ligne...*).



D'autre part, plusieurs solutions techniques existent pour le stockage des données (*cités par ordre de complexité croissante*) :

- fichiers plats (*formats texte* : **CSV**, **JSON**, **XML**...) lorsque le volume des données n'est pas trop important,
- base de données relationnelle, parfois objet (**SQLite**, **MySQL**, **SQL Server**), ou relationnelle objet (**PostgreSQL**, **Oracle**...),
- base de type **NoSQL** (*Not Only SQL*) initialement conçues pour de très gros volumes...



Exemples de solutions de stockage

(Parenthèse) Le principe KISS

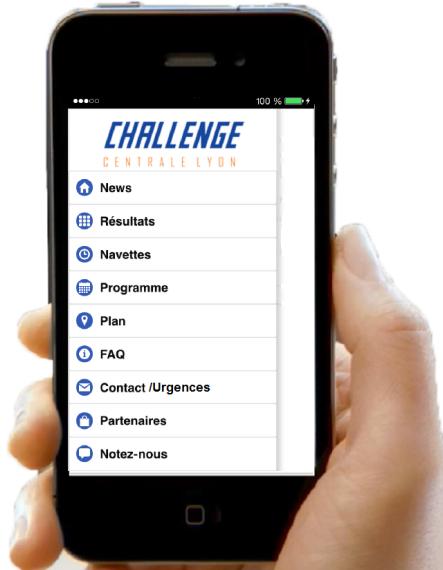
KISS est un acronyme pour :

« *Keep It Simple and Stupid* »

qui peut librement se traduire en français par :

« *Quand on peut, il vaut mieux faire simple que compliqué.* »

Appliqué au problème du stockage de données, le principe KISS suggère de ne pas surdimensionner la solution retenue.



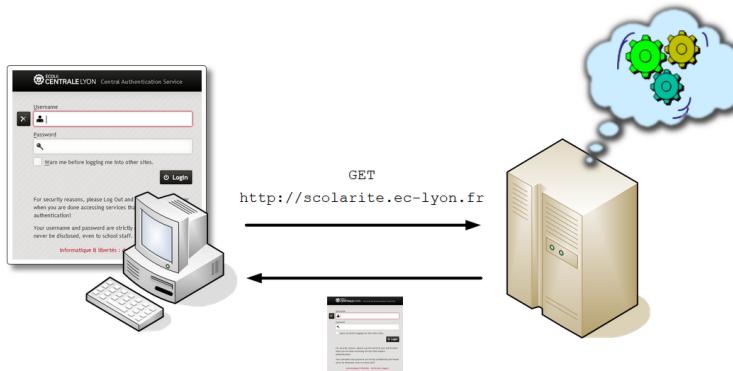
A titre d'exemple, l'*appli du week-end* de l'édition 2015 du **Challenge Centrale Lyon** (*anecdotiquement réalisée par un PE*) était une application web qui reposait sur des **fichiers au format JSON** pour stocker l'ensemble de ses informations :

- données relatives aux compétitions sportives (*localisation, horaires, équipes...*),
- aux navettes (*arrêts, horaires*),
- aux résultats des matches...

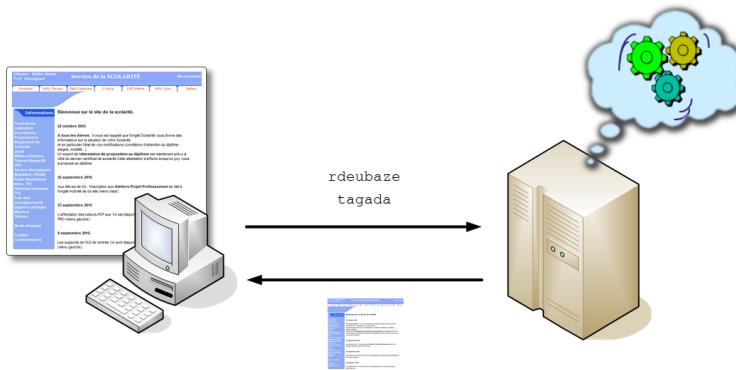


2.1.6 Principe du client léger

Historiquement, et c'est le cas le plus simple, le client se contente d'afficher les interfaces envoyées par le serveur, et de recueillir les informations entrées par l'utilisateur pour les renvoyer au serveur :



Exemple : le client demande une ressource, le serveur renvoie un formulaire de login

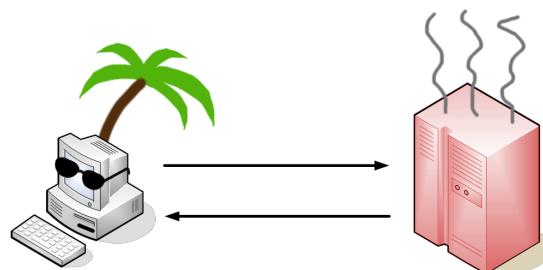


L'utilisateur s'authentifie, le serveur renvoie la ressource demandée

L'ensemble des traitements s'effectuent sur le serveur et le rôle du client est réduit au minimum, on parle dans ce cas de **client léger**.

Faire exécuter la logique applicative par le serveur présente quelques avantages :

- on a la maîtrise de l'environnement d'exécution (*le serveur*) sans avoir à trop se préoccuper de la diversité des terminaux de consultation (*navigateurs, versions, plateformes*),
- le code source de l'application est hébergé sur le serveur et n'est pas visible par les utilisateurs, mais possède aussi des inconvénients :
- comme l'ensemble du code est exécuté par le serveur, la charge de ce dernier croît très rapidement lorsque le nombre d'utilisateurs augmente,
- le potentiel des clients est en général largement sous-exploité.



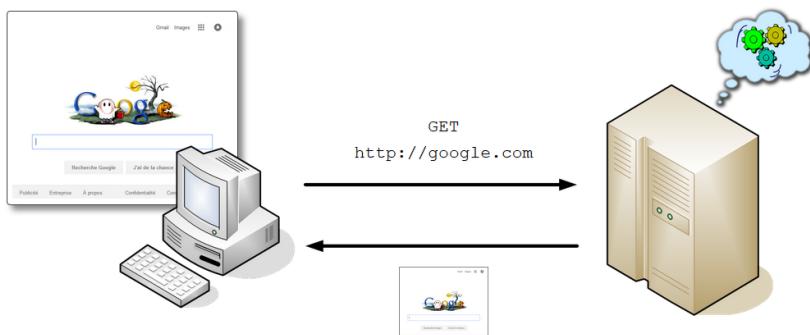


2.1.7 Cas du client riche

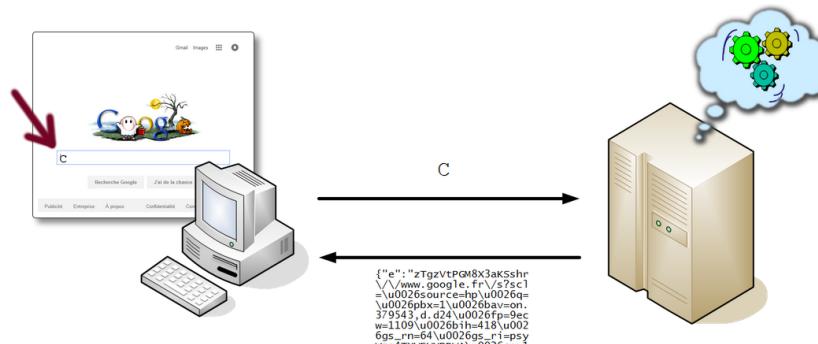
La réponse aux inconvénients du client léger consiste à développer des applications qui déportent tout ou partie de la logique applicative (*i.e.* "les *calculs*") sur le client. On parlera dans ce cas de **client riche**.

Dans ce cas, le serveur ne sert plus qu'à :

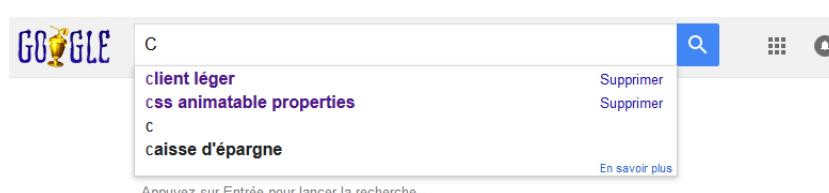
- initialiser le client en lui transmettant l'ensemble des interfaces,
- récupérer les informations issues des interactions avec l'utilisateur aux fins de stockage (*ou de traitement statistique*),
- transmettre au client les données brutes pour la mise à jour de l'interface.



Le client demande une ressource, le serveur renvoie l'interface et la logique.



L'utilisateur débute la saisie, le client communique avec le serveur qui renvoie des données brutes.



Appuyez sur Entrée pour lancer la recherche.

Le client met à jour l'interface en fonction des données reçues.

Les inconvénients des applications qui fonctionnent en « *client riche* » sont étroitement liés au principe de fonctionnement de cette architecture :

- le temps de chargement initial peut être relativement important, car le client reçoit l'ensemble des interfaces de l'application et tout le code nécessaire à son fonctionnement,
- le code source de l'application (*ou en tous cas une grande partie*) est lisible par l'utilisateur, puisque transmis au client.

On verra par la suite que certaines applications vont très loin dans cette direction, puisqu'une fois chargées elles peuvent même fonctionner en mode déconnecté (*i.e. sans le réseau*).



On parle dans ce cas de SPA - Single-Page Application ou application monopage en français.

L'inconvénient du code source lisible peut être minimisé en **obfuscant** le code. L'**obfuscation** est un procédé qui consiste à rendre le code illisible pour un humain, tout en conservant toutes ses fonctionnalités lorsqu'il est interprété par une machine.

En général, les logiciels d'obfuscation en profitent également pour essayer de minimiser la taille du code, en supprimant notamment tous les espaces inutiles...

2.1.8 Application isomorphique

Pour minimiser le temps de chargement initial des applications à client riche et le temps d'attente de la réponse du serveur, on peut penser à :

- exécuter sur le serveur tout le code d'initialisation nécessaire à l'obtention de la première interface,
- envoyer au client l'interface initiale qui s'affiche immédiatement, puis le code nécessaire à son fonctionnement.

Les interactions avec l'utilisateur sont communiquées au serveur qui met à jour ses données (*on dit aussi le modèle*), **et traitées en parallèle par le client** qui ajuste son interface en temps réel (*i.e. sans délai*) quitte à la remodifier en cas de différence avérée lors de la réponse du serveur.

Dans cette architecture le code s'exécute en parallèle sur le client et le serveur. Ceci pose un inconvénient majeur dans le cas où on n'utilise pas le même langage de programmation des deux côtés : il faut dupliquer le code.

La solution la plus élégante et la plus moderne consiste à s'appuyer sur un serveur qui fonctionne en Javascript (*c'est le même langage que sur le client*) et à utiliser un environnement (*framework*) qui permet d'écrire du code qui s'exécutera de la même façon sur le client que sur le serveur.

On parle alors d'**application isomorphe**.



Exemple de frameworks isomorphiques

2.1.9 AJAX

Comme on a pu le voir dans le cadre de l'architecture RIA (*client riche*) un client est capable d'émettre des requêtes de deux manières très différentes :

- lors du chargement d'une ressource pour
 - un changement de page,
 - le chargement d'une image ou autre ressource liée à une page (*CSS, javascript, fonte...*),
 - la soumission d'un formulaire (*cf. formulaire de login dans l'exemple du client léger*),
- suite à une interaction de l'utilisateur (*cf. exemple RIA lors de la frappe d'un caractère*).

Ce dernier type de requête est initié par le programme javascript qui s'exécute sur le client, et correspond à l'appellation **AJAX** (*Asynchronous Javascript And XML*).

Il n'y a aucune différence entre ces deux types de requêtes vu du serveur. La différence se situe côté client, dans la façon d'émettre la requête et dans la façon d'exploiter la réponse : utilisation directe de la ressource renvoyée dans le premier cas, traitement par programme dans le second.



2.1.10 Contenu de la réponse

Lors d'une requête classique le format de la ressource renvoyée par le serveur correspond au type de document attendu par le navigateur dans le contexte de la requête (*page html, image, vidéo, son, feuille de style CSS, fichier javascript...*)

Lors d'une requête AJAX par contre, le contenu et le format de la réponse dépendent de l'application et les possibilités sont variées :

- fragment de code HTML à placer tel quel dans la page,
- données formatées en XML (*d'où le X dans AJAX*),
- données formatées en JSON (*moins verbeux que XML et directement exploitable en Javascript*).

```
<suggestions>
  <item>client léger</item>
  <item>CSS animatable properties</item>
  <item>C language</item>
  <item>caisse d'épargne</item>
</suggestions>
```

```
{ "suggestions": [
  "client léger",
  "CSS animatable properties",
  "C language",
  "caisse d'épargne"
] }
```

Exemple de code XML (à gauche) et JSON (à droite)

2.1.11 Principes du protocole HTTP

Dans le cadre d'une architecture Web, le client et le serveur s'appuient sur le protocole **HTTP** pour échanger des informations (*requête et réponse*).

Même lorsque les données échangées possèdent un format binaire (*images, audio, vidéo*), les commandes du protocole HTTP lui-même correspondent à **du texte**, et sont par là-même directement lisibles et compréhensibles par des humains.

Le cas le plus simple est celui d'une requête **GET**, qui demande au serveur une ressource connaissant son adresse :

```
GET /index.html HTTP/1.1
Host: http://www.ec-lyon.fr
```

N.B. Le protocole HTTP a connu plusieurs versions depuis 1992 dont la dernière est **HTTP/2.0**, qui a été standardisée en février 2015.

N.B.2 - La requête ci-dessus correspond à la version **HTTP/1.1**. Dans la pratique, pour mentionner une requête on simplifiera souvent son écriture en :

```
GET http://www.ec-lyon.fr/index.html
```

La réponse du serveur comporte trois parties :

- la ligne de statut,
- la zone des entêtes, qui contient des informations au sujet du serveur, de l'échange et des données elles-mêmes (*on parle alors de métainformations*),
- le corps de la réponse, qui correspond en général à la ressource demandée.



```
HTTP/1.1 200 Ok // ligne de statut
Content-Length: 136 // .
Content-Type: text/html // .
Date: Fri, 30 Oct 2015 13:44:59 GMT // .
Etag: "94856-c0-5321c46d" // .
Last-Modified: Thu, 13 Mar 2014 14:45:01 GMT // .

<!DOCTYPE html> // corps de la réponse
<title>Exemple HTML</title>
<meta charset="utf-8">
<h1>Exemple</h1>
<p>Hello,<br>Ceci est un exemple de fichier HTML // .
```

Remarquer dans les entêtes le type de contenu de la réponse, sa taille, la date de dernière modification de la ressource...

N.B. Dans certaines circonstances Content-Type et Content-Length sont obligatoires.

Le protocole HTTP prévoit d'autres types de requêtes (*HTTP verbs*) que GET, que le client peut émettre pour indiquer l'action qu'on désire appliquer à la ressource (*liste non exhaustive*) :

Méthode	action
GET	On désire récupérer une représentation de la ressource. Utilisée uniquement pour lire des données.
HEAD	Comme GET, sauf qu'on désire uniquement les entêtes HTTP de la réponse (<i>pas le corps</i>). Utilisée par le client pour la gestion du cache.
POST	Demande que les données contenues dans la requête soient ajoutées à la ressource. Classiquement utilisée sur le web pour soumettre le contenu d'un formulaire.
PUT	Demande à ce que l'entité contenue dans la requête soit enregistrée et disponible ensuite à l'adresse utilisée. Si la ressource existe déjà elle doit être modifiée, sinon créée.
DELETE	Demande à ce que la ressource correspondante soit supprimée.
OPTIONS	Demande au serveur quelles sont les méthodes HTTP supportées.

Méthodes HTTP

N.B. Un serveur (*ou une application Web*) n'implémente pas forcément toutes ces méthodes, et encore moins toutes les méthodes pour toutes les ressources. On s'attend *a priori* à ce qu'il réponde au moins aux méthodes GET et HEAD et si possible OPTIONS.

Certaines requêtes ne sont pas anodines (*cf. DELETE*), d'autres ne doivent pas être répétées à la légère (*cf. POST*)... C'est pourquoi HTTP définit les notions suivantes.

Méthodes sûres

Une méthode sûre ne produit pas d'effets de bords côté serveur (*pas de modification de la ressource concernée*). GET, HEAD et OPTIONS doivent typiquement pouvoir être considérées comme sûres par le client et l'utilisateur.

A contrario PUT, POST, et DELETE ne sont pas sûres.

Méthodes idempotentes

Est idempotente une méthode qui produit invariablement les mêmes effets lorsqu'elle est appliquée plusieurs fois de suite. Les méthodes sûres (*i.e. sans effets de bords*) sont par définition idempotentes.

PUT et DELETE sont idempotentes : le résultat sera toujours le même après plusieurs requêtes identiques consécutives).



`POST` n'est pas idempotente : plusieurs requêtes consécutives peuvent amener à ajouter plusieurs fois le même article dans un blog par exemple, ou à répéter plusieurs fois une même transaction financière...

N.B. Il est de la responsabilité du concepteur d'une application de veiller à ce que ces principes soient respectés.

N.B.2 Les navigateurs affichent par défaut une fenêtre popup de confirmation lors de l'envoi d'un formulaire par `POST` car cette méthode n'est pas sûre...

2.1.12 L'architecture REST

Bien que la correspondance sémantique ne soit pas parfaite, les méthodes HTTP `PUT`, `GET`, `POST`, `DELETE` rappellent les opérations CRUD (*Create, Read, Update, Delete*) classiquement effectuées pour gérer le cycle de vie des objets en général, et des tables dans le contexte d'une base relationnelle en particulier.

Ce n'est pas un hasard.

Il existe une préconisation d'architecture nommée **REST** (*pour Representational State Transfer*) qui s'appuie sur le protocole HTTP et permet de manipuler les ressources, identifiées par leur adresse, via les « verbes » HTTP.

Exemple de requêtes d'une application « *RESTful* » fictive permettant d'effectuer diverses opérations sur un article de blog :

```
// Création d'un article
POST http://blogfictif.com/articles

// Récupération d'une représentation d'un article (pour affichage)
GET http://blogfictif.com/articles/133

// Modification d'un article particulier
PUT http://blogfictif.com/articles/133

// suppression d'un article
DELETE http://blogfictif.com/articles/133
```

N.B. Il faut recourir à AJAX pour émettre des requêtes `PUT` et `DELETE` depuis un navigateur.



2.2 Contexte

Le développement d'applications Web s'inscrit dans un contexte en pleine évolution.

- Les utilisateurs exigent des applications qui fonctionnent quel que soit le terminal de consultation :
 - Mac, PC, quelle que soit la taille et la résolution de l'écran,
 - tablettes quel que soit le système d'exploitation,
 - smartphone quels que soient la marque et l'opérateur...



- Certaines applications demandent à délivrer du contenu multimédia (*audio, vidéo, animations graphiques*) avec les contraintes ci-dessus.
- Dans l'idéal les applications doivent fonctionner dans un contexte de mobilité permanente, avec des moments où le réseau n'est pas disponible.
- Pour pouvoir concurrencer les applications natives, une application web doit offrir des fonctionnalités comparables à celles obtenues par les applications natives (*géolocalisation, GPS, appareil photo/caméra, carnet d'adresse...*).

2.2.1 Diversité des terminaux

Face à la diversité des terminaux, en particulier en ce qui concerne la taille des écrans, plusieurs stratégies sont possibles :

- **fixed design** – l'interface est construite avec des dimensions fixes, et son allure ne changera pas quelles que soient les caractéristiques de l'écran de consultation,
- **fluid (ou liquid) design** – la taille des différentes zones de l'interface est proportionnelle à la taille de l'écran,
- **adaptive design** – l'arrangement et la taille des zones de l'interface dépend de la taille de l'écran,
- **responsive design** – l'existence, le contenu, l'arrangement et la taille des zones de l'interface dépend de la taille de l'écran.

[article]



N.B. **Bootstrap** est un exemple de framework destiné au développement d'interfaces adaptatives et responsives. [démonstration]





2.2.2 Mobilité

La mobilité des utilisateurs, et en particulier l'absence temporaire de connexion réseau pose un autre défi à relever pour qui développe des applications Web.

Dans le cas d'une application isomorphe, il suffit de stocker localement sur le client les informations qui sont destinées au serveur lorsque celui-ci est absent.

Cela pose toutefois le problème de la re-synchronisation des données entre le client et le serveur au moment où celui-ci réapparaît, si d'autres utilisateur ou d'autres processus ont eu la possibilité de faire évoluer indépendamment les données du serveur.

Il existe diverses solutions dont certaines apparues récemment (*i.e. avec HTML5*) pour conserver des informations côté client :

- les cookies, qui existent depuis l'aube du Web, permettent de stocker sur le client une chaîne comportant jusqu'à 4000 caractères,
- [Web Storage](#) permet de stocker des paires clé-valeur à concurrence de plusieurs MB (*5 pour Chrome, FF et Opera, 10 pour IE, 25 sur BlackBerry*),
- [IndexDB](#) est une base NoSQL côté client, permettant de stocker des objets JSON.

2.2.3 Contenus multimédia

Pendant longtemps, les contenus audio et vidéo ont été les parents pauvres du Web : pour afficher une vidéo ou jouer une séquence audio on avait recours à des applications extérieures (*applet java d'abord, puis plugin Flash*)

Ce point a été largement amélioré avec HTML5, puisqu'on dispose maintenant des deux balises `<audio>` et `<video>` dédiées à ces usages.



Exemple de vidéo HTML5 stylée en CSS (cadre et bordure)

2.2.4 Fonctionnalités des applications natives

Le défi pour les applications web est d'arriver aux mêmes fonctionnalités que des applications natives développées spécifiquement pour une plateforme donnée (*Androïd ou iOS*).

L'intérêt est évidemment de ne développer qu'une seule version de l'application (*sous HTML5, CSS3, Javascript*) au lieu d'en développer plusieurs (*Androïd / Java, iOS / Objective-C, Swift...*).

Les inconvénients potentiels concernent les problèmes de sécurité qui se posent lorsqu'une application provenant du Web essaie d'accéder aux ressources matérielles d'un dispositif.



Voici quelques APIs (*Application Programming Interfaces*), toutes récentes et en cours de stabilisation, qui illustrent le champ des possibles :

- Géolocalisation [[Geolocation API](#)]
- Accès à la caméra et au micro [[Media Capture and Streams](#)]
- Accès à l'orientation et à l'accéléromètre [[DeviceOrientation Event Specification](#)]
- Accès à la carte graphique [[WebGL](#)]

2.3 Comment aborder cette complexité ?

On a vu que :

- une application web est basée sur le modèle client-serveur,
- le client est un navigateur ou un autre (*diversité des plateformes*),
- le serveur offre un contexte d'exécution (*scripts ou serveur d'application*), s'appuie sur un serveur de bases de données (*relationnel ou noSQL*) et synchronise éventuellement des données sur le client,
- l'interactivité des interfaces utilisateur est programmée en Javascript, et l'expérience utilisateur améliorée avec AJAX,
- on voudra des interfaces responsives adaptées aux terminaux mobiles et tactiles.

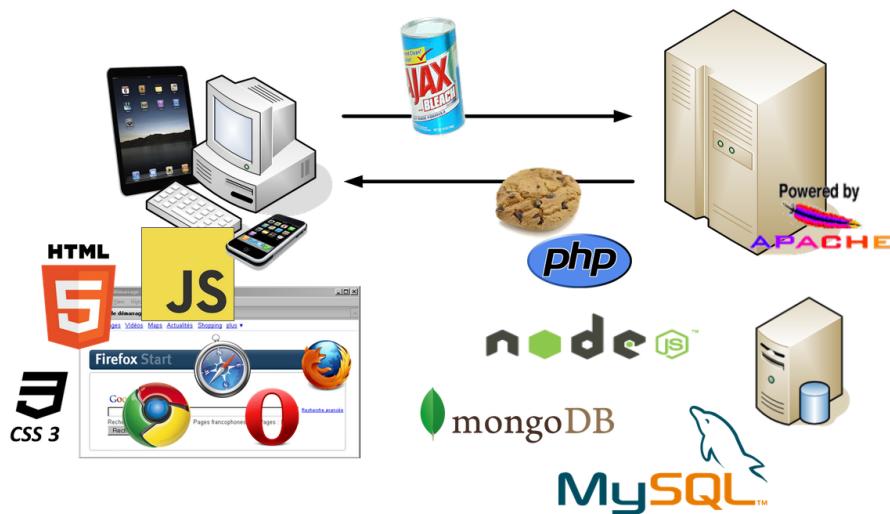


Illustration de la complexité de l'environnement d'une application web.

Tout cela constitue une **architecture complexe**, à laquelle pourront s'ajouter des connexions persistantes (*websockets*), des graphiques vectoriels (*SVG*), des graphiques 2D ou 3D (*canvas*), du multitouch, l'accès au matériel (*GPS, accéléromètre, micro, caméra*)...

Comment gérer tout cela lors du développement d'une application web ?

2.3.1 Décomposition des tâches

Diverses stratégies, souvent complémentaires, existent pour réduire ce problème et le décomposer en sous-tâches (*attention toutefois à ne pas négliger les interactions entre sous-tâches lors de ce découpage*).

- **Découpage métier** – Une première approche consiste à découper le problème en fonction des compétences nécessaires à la réalisation de chacune des tâches ainsi dégagées, et d'identifier les métiers correspondants.
- **Découpage topologique** – Un second découpage possible s'appuie sur l'architecture de l'application, et sépare les développements en termes d'interfaces, de code client (*frontend*), de programmation serveur, de modèle de données (*backend*)...



- **Découpage temporel** – Enfin, il est possible d'aborder le problème en découplant le travail en tranches temporelles réduites (*une semaine à 1 mois maximum*), qui permettent d'avoir rapidement des retours du commanditaire concernant des fonctionnalités réduites mais opérationnelles (*méthodes de gestion de projet agiles*).

2.3.2 Les métiers du Web

*[2018] N.B. les liens présents dans cette section ne fonctionnent plus... (serveur payant ?)

Le [portail des métiers de l'internet](#) du ministère de l'économie, de l'industrie, et du numérique recense plusieurs dizaines de métiers allant du marketing à la formation, en passant par la création de contenu et la gestion des infrastructures réseau. Tous ne concernent pas le développement d'applications web, mais en voici quelques-uns :

- architecte de l'information, architecte web, chef de projet web, correspondant informatique et libertés, juriste [[Métiers de la conception et gestion de projet](#)],
- designer d'interaction, designer web mobile, ergonome web, graphiste web, webdesigner [[Métiers "Interfaces et création numérique"](#)],
- développeur web, développeur web mobile, intégrateur web, gestionnaire de base de données [[Métiers "Programmation et développement"](#)],



Source : <http://www.metiers.internet.gouv.fr/>

2.3.3 Frameworks

L'arrivée du concept d'application isomorphe nous montre que le découpage des développements en termes de « *client d'un côté, serveur de l'autre* » n'est pas très pertinent.

Il existe des environnements de développement (*frameworks*) qui fournissent un cadre et tous les mécanismes pour rapidement mettre en place l'architecture d'une application web (*bidirectional binding, routing, composants réutilisables, isomorphisme...*).

En voici un échantillonnage purement arbitraire :

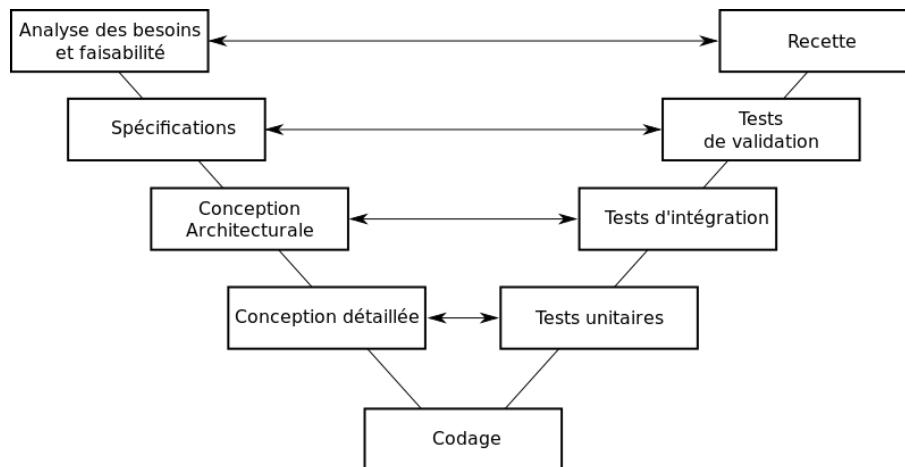


Framework	Licence	Lancement	Contributeur	Points forts
Angular	MIT	2016	Google	TypeScript, liaison de données bidirectionnelle, composants réutilisables, routage
React	BSD	2013	Facebook	liaison de données bidirectionnelle, composants réutilisables, isomorphisme
Meteor	MIT	2014	Meteor Dev Group	isomorphisme, liaison de données depuis la base jusqu'à l'interface
Aurelia	MIT	2015	Durandal Inc.	Javascript ES6/7, databinding bidirectionnel adaptatif, Web Components, routage

Frameworks Javascript

2.3.4 Gestion de projet informatique

La méthode de gestion de projet traditionnellement employée dans le domaine de l'informatique est modélisée par le fameux **cycle en V** :



by Christophe.moustier [CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0/>)], via Wikimedia Commons

Toutefois « *en pratique, il est difficile voire impossible de totalement détacher la phase de conception d'un projet de sa phase de réalisation. C'est souvent au cours de l'implémentation qu'on se rend compte que les spécifications initiales étaient incomplètes, fausses, ou irréalisables... »* — Wikipedia

Les aléas de la gestion de projet informatique (épisode 1)...



Comment le client a exprimé son besoin



Comment le chef de projet l'a compris



Comment l'architecte l'a conçu



Comment le programmeur l'a écrit



Comment le vendeur l'a décrit



Ce dont le client a réellement besoin

Les aléas de la gestion de projet informatique (épisode 2)...

2.3.5 Les méthodes agiles

Le terme « **méthodes agiles** » regroupe un ensemble de méthodes de gestion de projet qui cherchent à éviter les écueils des méthodes traditionnelles par une approche pragmatique et réactive qui implique le commanditaire tout au long du processus d'un développement en mode itératif.

L'ensemble des méthodes agiles se reconnaissent dans un texte fondateur appelé le « *Manifeste Agile* » :

Manifeste pour le développement Agile de logiciels

« Nous découvrons comment mieux développer des logiciels par la pratique et en aidant les autres à le faire. Ces expériences nous ont amenés à valoriser :

Les individus et leurs interactions plus que les processus et les outils.

Des logiciels opérationnels plus qu'une documentation exhaustive.

La collaboration avec les clients plus que la négociation contractuelle.

L'adaptation au changement plus que le suivi d'un plan.

Nous reconnaissons la valeur des seconds éléments, mais privilégions les premiers. »

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

© 2001, the above authors this declaration may be freely copied in any form, but only in its entirety through this notice.



Références

[Wikipédia], p.3

WIKIPÉDIA, "Application Web", .

En ligne, disponible sur https://fr.wikipedia.org/wiki/Application_web
[consulté le 4 nov. 2015]

[architecture distribuée], p.3

WIKIPÉDIA, "Architecture distribuée", .

En ligne, disponible sur https://fr.wikipedia.org/wiki/Architecture_distribu%C3%A9e
[consulté le 4 nov. 2015]

[client-serveur], p.3

WIKIPÉDIA, "Client-serveur", .

En ligne, disponible sur <https://fr.wikipedia.org/wiki/Client-serveur>
[consulté le 4 nov. 2015]

[protocole], p.3

WIKIPÉDIA, "Protocole de communication", .

En ligne, disponible sur https://fr.wikipedia.org/wiki/Protocole_de_communication
[consulté le 4 nov. 2015]

[ressource], p.3

WIKIPÉDIA, "Ressource du World Wide Web", .

En ligne, disponible sur https://fr.wikipedia.org/wiki/Ressource_du_World_Wide_Web
[consulté le 4 nov. 2015]

[architecture trois tiers], p.4

WIKIPÉDIA, "Architecture trois tiers", .

En ligne, disponible sur https://fr.wikipedia.org/wiki/Architecture_trois_tiers
[consulté le 4 nov. 2015]

[IHM], p.4

WIKIPÉDIA, "Interactions homme-machine", .

En ligne, disponible sur https://fr.wikipedia.org/wiki/Interactions_homme-machine
[consulté le 4 nov. 2015]

[CSV], p.7

WIKIPÉDIA, "Comma-separated values", .

En ligne, disponible sur https://fr.wikipedia.org/wiki/Comma-separated_values
[consulté le 4 nov. 2015]

[JSON], p.7

WIKIPÉDIA, "Javascript Object notation", .

En ligne, disponible sur https://fr.wikipedia.org/wiki/JavaScript_Object_Notation
[consulté le 4 nov. 2015]

[XML], p.7

WIKIPÉDIA, "Extensible Markup Language", .

En ligne, disponible sur https://fr.wikipedia.org/wiki/Extensible_Markup_Language
[consulté le 4 nov. 2015]

[SQLite], p.7

THE SQLITE CONSORTIUM, "SQLite", .

En ligne, disponible sur <https://www.sqlite.org/>
[consulté le 4 nov. 2015]



[MySQL], p.7

WIKIPÉDIA, "MySQL", .
En ligne, disponible sur <https://fr.wikipedia.org/wiki/MySQL>
[consulté le 4 nov. 2015]

[SQL Server], p.7

WIKIPÉDIA, "Microsoft SQL Server", .
En ligne, disponible sur https://fr.wikipedia.org/wiki/Microsoft_SQL_Server
[consulté le 4 nov. 2015]

[PostgreSQL], p.7

WIKIPÉDIA, "PostgreSQL", .
En ligne, disponible sur <https://fr.wikipedia.org/wiki/PostgreSQL>
[consulté le 4 nov. 2015]

[Oracle], p.7

WIKIPÉDIA, "Oracle Database", .
En ligne, disponible sur https://fr.wikipedia.org/wiki/Oracle_Database
[consulté le 4 nov. 2015]

[NoSQL], p.7

WIKIPÉDIA, "NoSQL", .
En ligne, disponible sur <https://fr.wikipedia.org/wiki/NoSQL>
[consulté le 4 nov. 2015]

[Challenge Centrale Lyon], p.7

ASSOCIATION CHALLENGE CENTRALE LYON, "Challenge Centrale Lyon", .
En ligne, disponible sur <http://www.challenge-grandes-ecoles.fr/>
[consulté le 4 nov. 2015]

[client léger], p.8

WIKIPÉDIA, "Client léger", .
En ligne, disponible sur https://fr.wikipedia.org/wiki/Client_l%C3%A9ger
[consulté le 4 nov. 2015]

[client riche], p.9

WIKIPÉDIA, "Rich Internet Application", .
En ligne, disponible sur https://fr.wikipedia.org/wiki/Rich_Internet_application
[consulté le 4 nov. 2015]

[Single-Page Application], p.10

WIKIPÉDIA, "Application Web monopage", .
En ligne, disponible sur https://fr.wikipedia.org/wiki/Application_web_monopage
[consulté le 4 nov. 2015]

[obfuscation], p.10

WIKIPÉDIA, "Obfuscation (software)", .
En ligne, disponible sur https://en.wikipedia.org/wiki/Obfuscation_%28software%29
[consulté le 4 nov. 2015]

[application isomorphe], p.10

J.R. BÉDARD, "Isomorphic Javascript", .
En ligne, disponible sur <http://isomorphic.net/javascript>
[consulté le 4 nov. 2015]



[protocole HTTP], p.11

WIKIPÉDIA, "Hypertext Transfer Protocol", .

En ligne, disponible sur https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol
[consulté le 4 nov. 2015]

[HTTP/2.0], p.11

THE INTERNET ENGINEERING TASK FORCE (IETF), "HTTP/2 Approved", .

En ligne, disponible sur <http://www.ietf.org/blog/2015/02/http2-approved/>
[consulté le 4 nov. 2015]

[REST], p.13

WIKIPÉDIA, "Representational State Transfer", .

En ligne, disponible sur https://fr.wikipedia.org/wiki/Representational_State_Transfer
[consulté le 4 nov. 2015]

[article], p.14

SMASHING MAGAZINE, "Fixed vs. Fluid vs. Elastic Layout: What's The Right One For You?", .

En ligne, disponible sur <http://www.smashingmagazine.com/2009/06/fixed-vs-fluid-vs-elastic-layout-whats-the-right-one-for-you/>
[consulté le 4 nov. 2015]

[Bootstrap], p.14

THE BOOTSTRAP CORE TEAM, "Get Bootstrap", .

En ligne, disponible sur <http://getbootstrap.com/>
[consulté le 4 nov. 2015]

[démonstration], p.14

THE BOOTSTRAP CORE TEAM, "Navbar example", .

En ligne, disponible sur <https://getbootstrap.com/docs/5.0/examples/navbar-static/>
[consulté le 28 apr. 2021]

[Web Storage], p.15

WIKIPÉDIA, "Web Storage", .

En ligne, disponible sur https://en.wikipedia.org/wiki/Web_storage
[consulté le 4 nov. 2015]

[IndexDB], p.15

WIKIPÉDIA, "Indexed Database API", .

En ligne, disponible sur https://fr.wikipedia.org/wiki/Indexed_Database_API
[consulté le 4 nov. 2015]

[Geolocation API], p.16

THE WORLD WIDE WEB CONSORTIUM (W3C), "Geolocation API Specification", .

En ligne, disponible sur <http://dev.w3.org/geo/api/spec-source.html>
[consulté le 4 nov. 2015]

[Media Capture and Streams], p.16

THE WORLD WIDE WEB CONSORTIUM (W3C), "Media Capture and Streams", .

En ligne, disponible sur <http://www.w3.org/TR/mediacapture-streams/>
[consulté le 4 nov. 2015]

[DeviceOrientation Event Specification], p.16

THE WORLD WIDE WEB CONSORTIUM (W3C), "DeviceOrientation Event Specification", .

En ligne, disponible sur <http://w3c.github.io/deviceorientation/spec-source-orientation.html>
[consulté le 4 nov. 2015]



[WebGL], p.16

THE KHRONOS GROUP, "WebGL - OpenGL ES 2.0 for the Web", .
En ligne, disponible sur <https://www.khronos.org/webgl/>
[consulté le 4 nov. 2015]

[portail des métiers de l'internet], p.17

MINISTÈRE DE L'ÉCONOMIE DE L'INDUSTRIE, ET DU NUMÉRIQUE, "Le portail des métiers de l'Internet", (via The Internet Archive).
En ligne, disponible sur <https://web.archive.org/web/20161215124340/http://metiers.internet.gouv.fr/>
[consulté le 15 déc. 2016]

[Métiers de la conception et gestion de projet], p.17

MINISTÈRE DE L'ÉCONOMIE DE L'INDUSTRIE, ET DU NUMÉRIQUE, "Famille de métiers : Conception et gestion de projet", (via The Internet Archive).
En ligne, disponible sur <https://web.archive.org/web/20170929144441/http://www.metiers.internet.gouv.fr/famille-metier/conception-et-gestion-de-projet>
[consulté le 29 sep. 2017]

[Métiers "Interfaces et création numérique"], p.17

MINISTÈRE DE L'ÉCONOMIE DE L'INDUSTRIE, ET DU NUMÉRIQUE, "Famille de métiers : interfaces et création numérique", (via The Internet Archive).
En ligne, disponible sur <https://web.archive.org/web/20171031235428/http://metiers.internet.gouv.fr/famille-metier/interface-et-creation-numerique>
[consulté le 31 oct. 2017]

[Métiers "Programmation et développement"], p.17

MINISTÈRE DE L'ÉCONOMIE DE L'INDUSTRIE, ET DU NUMÉRIQUE, "Famille de métiers : Programmation et développement", (via The Internet Archive).
En ligne, disponible sur <https://web.archive.org/web/20170103030431/http://www.metiers.internet.gouv.fr/famille-metier/programmation-et-developpement>
[consulté le 3 janv. 2017]

[Angular], p.18

WIKIPÉDIA, "Angular", .
En ligne, disponible sur <https://fr.wikipedia.org/wiki/Angular>
[consulté le 21 nov. 2018]

[React], p.18

WIKIPÉDIA, "React (Javascript Library)", .
En ligne, disponible sur https://en.wikipedia.org/wiki/React_%28JavaScript_library%29
[consulté le 4 nov. 2015]

[Meteor], p.18

WIKIPÉDIA, "Meteor (framework)", .
En ligne, disponible sur https://fr.wikipedia.org/wiki/Meteor_%28framework%29
[consulté le 4 nov. 2015]

[Aurelia], p.18

DURANDAL INC., "Aurelia", .
En ligne, disponible sur <http://aurelia.io/>
[consulté le 4 nov. 2015]

[cycle en V], p.18

WIKIPÉDIA, "Cycle en V", .
En ligne, disponible sur https://fr.wikipedia.org/wiki/Cycle_en_V
[consulté le 4 nov. 2015]



[méthodes agiles], p.19

WIKIPÉDIA, "Méthode agile", .

En ligne, disponible sur https://fr.wikipedia.org/wiki/M%C3%A9thode_agile
[consulté le 4 nov. 2015]

[Manifeste Agile], p.19

THE AGILE ALLIANCE, "Manifeste pour le développement Agile de logiciels", .

En ligne, disponible sur <http://agilemanifesto.org/iso/fr/>
[consulté le 4 nov. 2015]