

Les Nouvelles Technologies du Web

Angular, NestJS





HiveMind Solutions

Simon HAJEK



Programme des prochains cours



Programme des prochains cours

- Prérequis (1 cours)
 - Utilisation du JavaScript pour créer des applications
- Développement frontend : (1,5 cours)
 - Mise en place d'une SPA
 - Angular et quelques concepts
- Développement backend : (1,5 cours)
 - Plateforme Node.js
 - Création d'une API REST pour une architecture SOA
 - Un framework pour les API REST (mais pas que) : NestJS
 - Stockage des données avec MongoDB
 - Validation des entrées/sorties de votre API



Programme des prochains cours

- Evaluation frontend et backend : (1 cours)
 - Un sujet sera fourni à la fin du 4ème cours.
 - Vous devrez réaliser une application complète reprenant tout ce qu'on aura vu.
 - Vous serez en équipe de 2.
 - L'application devra être réalisée entre le 4ème cours et le dernier cours.
 - 20 Minutes de présentation par binôme.
 - Vous devrez fournir une présentation PPT et votre code source afin d'expliquer ce que vous avez fait et comment le tester.
 - J'exécuterai l'application devant tout le monde et je suivrai vos recommandations pour la tester.
 - Je noterai la présentation, le fonctionnement de l'application, les fonctionnalités implémentées et le code source.



Introduction



Introduction

Le développement web a beaucoup évolué ces dernières années :

- ▶ Des nouvelles méthodologies
 - Agile, Lean Startup, ...
- ▶ Des nouvelles plateformes
 - Explosion du cloud computing
- ▶ Des nouvelles technologies
 - Node.js, NoSQL, SPA, ...



Introduction

Il est important et nécessaire de comprendre ces évolutions !

Mais pour commencer ...

Qu'est ce que le développement web en 2024 ?



Historique



Historique

Au début (1990), le web était statique.

Les serveurs servaient des pages informatives, le contenu était fixé une fois pour toute.

Le navigateur était chargé de faire le rendu de ces pages : très peu d'interactions avec l'utilisateur.



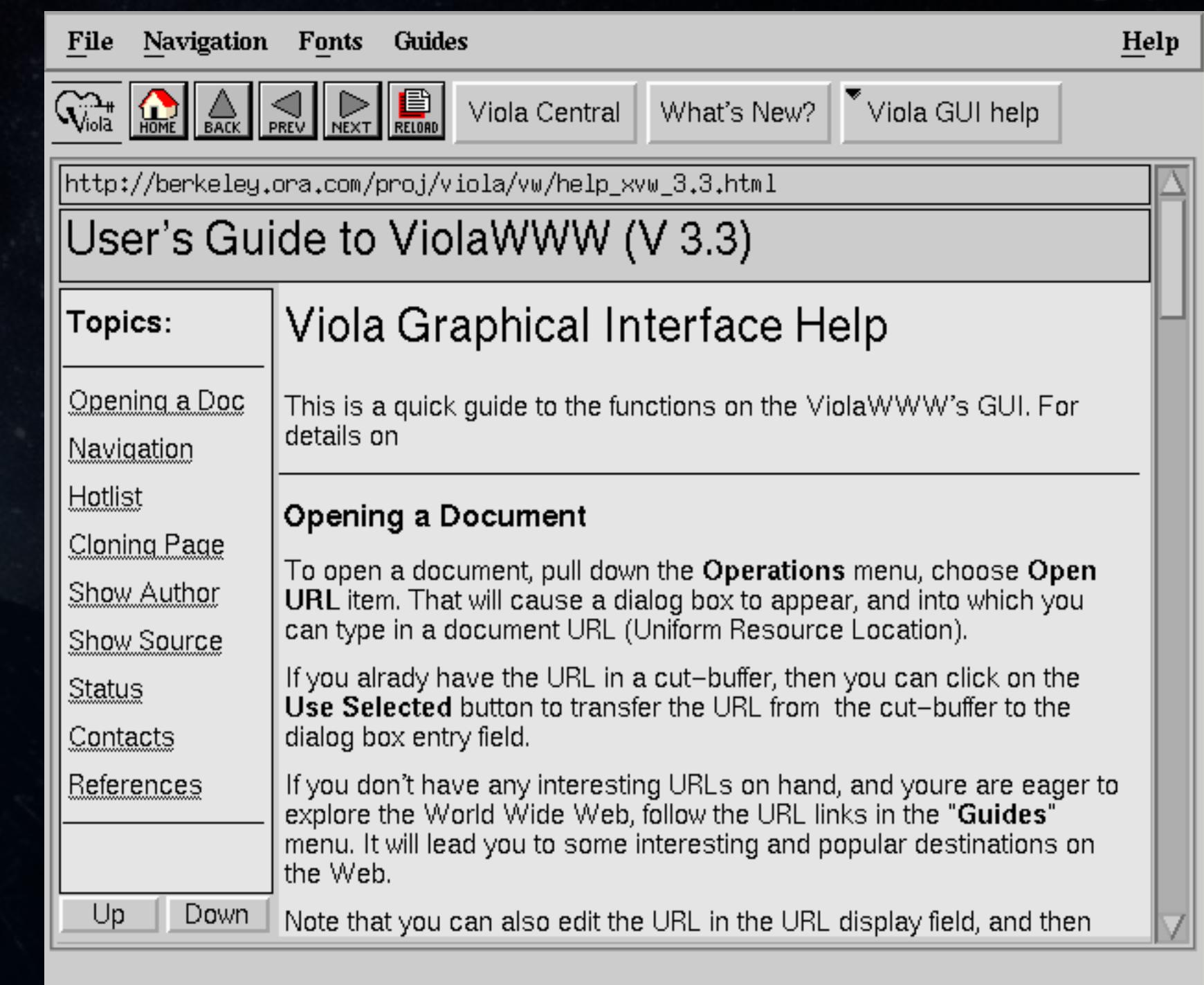
Historique

1991 / 1992

HTML 1.0 : « langage » permettant de structurer
le contenu d'une page web

ViolaWWW : un des premiers navigateurs

26 sites web dans le monde !



Historique

1993 / 1994

De nouveaux navigateurs : Mosaic, Netscape

Apparition des images dans les pages web !



10 022 sites web fin 1994



Historique

A partir de 1995, le Web commence à intéresser les médias et le grand public.

Et surtout, c'est l'apparition de PHP !

Cette fois, le serveur va exécuter des scripts en réponse aux requêtes des utilisateurs :
le contenu de la page est généré à la demande.



Historique

Personnalisation du contenu en fonction de l'utilisateur : c'est le début du web dynamique.

Le navigateur reste encore très basique : les interactions se limitent à cliquer sur des liens hypertextes ou des boutons.

JavaScript est créé en 1995, mais pour le moment ses capacités sont très limitées.



Historique

1996 / 1997

Flash



Applets Java



Plus de 1 000 000 de sites fin 1997



Historique

Les applets Java et les applications Flash ont permis d'amener du dynamisme sur le navigateur.

Interfaces utilisateurs plus riches, plus d'actions.

L'applet est “servie” par le serveur mais s'exécute localement dans le navigateur.

L'ancêtre des “Single Page Applications”.



Historique

Et puis...

1998 : Moteur de recherche Google

2000 : AJAX

2004 : Concept de “web 2.0”, Facebook

2008 : Google Chrome





Aujourd’hui



Aujourd'hui

Le navigateur est devenu aujourd'hui une plateforme d'exécution à part entière :

- ▶ JavaScript performant
- ▶ Chargement de données asynchrones ([AJAX](#))
- ▶ Sécurité ([SSL](#))
- ▶ Rendu graphique très évolué ([CSS3](#), [SVG](#), [Canvas](#), [WebGL](#))
- ▶ HTML 5 : API Filesystem, base de données embarquées, ...



Une première rupture :

Les « Single Page Applications »



Les architectures s'adaptent à la technologie

Changement de principe structurant dans le développement :

- ▶ A l'origine, les calculs étaient tous effectués sur le serveur (PHP)
- ▶ Les applets Java ont déporté une partie des tâches sur le client. Mais cela a été progressivement abandonné : **sandboxing contraignant, mauvaise compatibilité (mobile)**
- ▶ L'état de l'art est ensuite revenu aux traitements côtés serveurs : Java EE, .NET, ...



Single Page Application

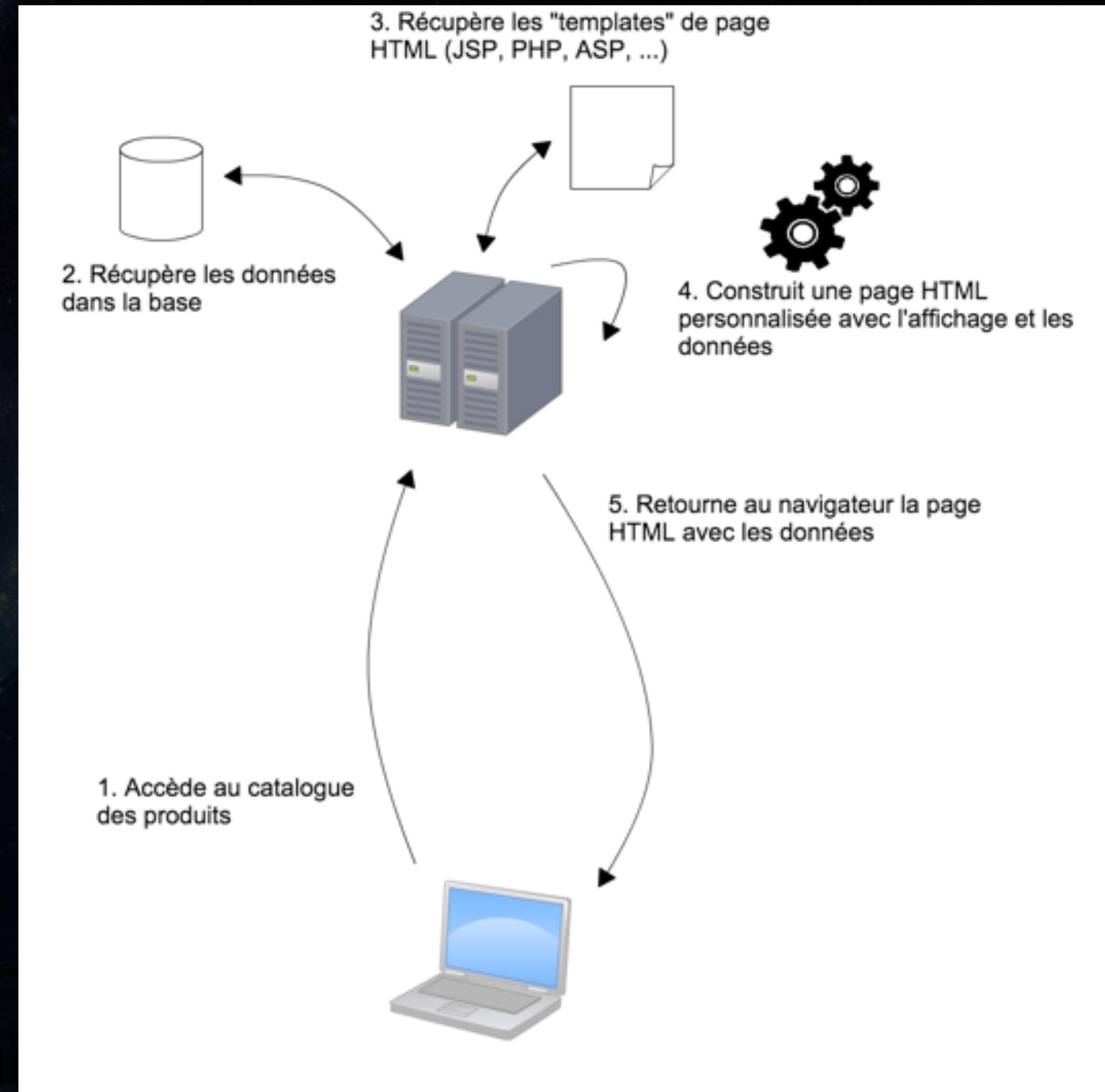
Maintenant, nouveau retour en force du navigateur :

- Concept de “SPA”
- Frameworks: Angular, VueJS, React
- Permet d'apporter plus de réactivité aux interfaces et de décharger les serveurs de certaines tâches
- Le **serveur** expose des APIs pour accéder aux données et l'affichage est construit par le navigateur



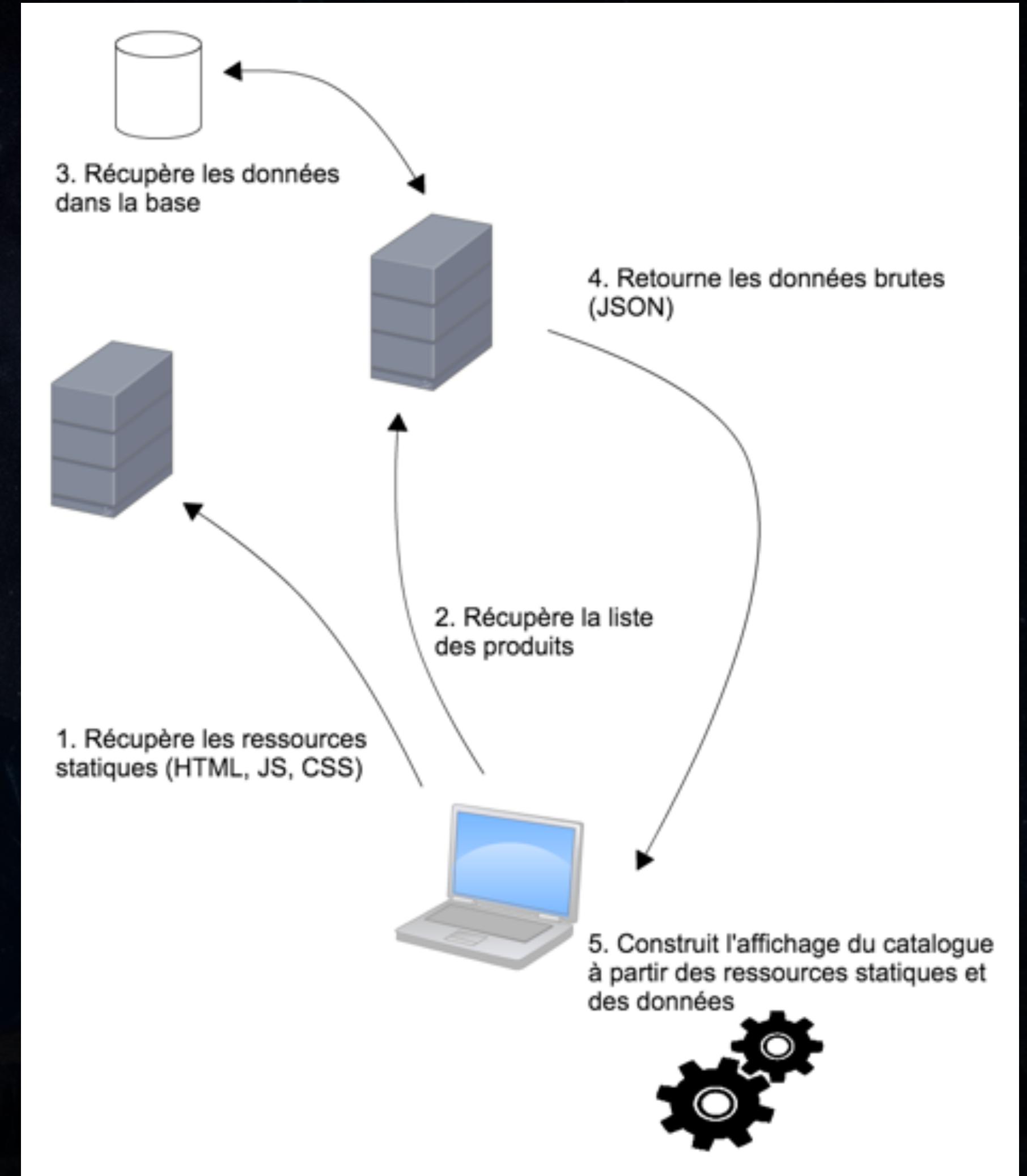
Site dynamique “classique” ...

- ▶ Les traitements sont effectués sur le serveur
- ▶ Le navigateur se limite à envoyer des requêtes et afficher les réponses
- ▶ ... comment faire pour intégrer un nouveau type de consommateur pour ces données (mobile, ...) ?



... vs API et SPA

- ▶ Les traitements sont effectués sur le navigateur
- ▶ Le serveur expose une API retournant les données dans un format standardisé
- ▶ Très facile d'intégrer de nouveaux consommateurs : mobile ou d'autres applications



Une deuxième rupture :

Les architectures distribuées



Le modèle historique

Avec l'explosion des usages de l'internet et des utilisateurs, la capacité des serveurs à gérer la charge est devenue un problème.

Comment gérer 3 000 utilisateurs simultanés ? 1000 requêtes par seconde ? Répondre aux utilisateurs en moins de 1 seconde ?

On a augmenté la puissance des serveurs : mainframes, +100 Go de RAM, ...



Mais ce modèle a des limites

Google : +80 000 requêtes par secondes.

Coût des mainframes très importants.

Augmentation des puissances de CPU limitée.

Changement de paradigme vers les architectures distribuées.



Super-computer vs commodity hardware

Google, Facebook, Twitter, ... font tourner leurs applications sur du “commodity hardware” :

- Des “petits” serveurs, mais en très grand nombre.
- Cela permet de diminuer les coûts.
- De distribuer géographiquement les serveurs au plus près des utilisateurs.
- Et d'avoir une bien meilleure tolérance aux pannes.



De nouvelles contraintes

L'architecture distribuée implique de nouvelles contraintes :

- Des standards pour faire communiquer des applications indépendantes entre elles (HTTP / REST, JSON, ...).
- Des nouveaux langages et paradigmes pour gérer la parallélisation (programmation fonctionnelle).
- De nouveaux modèles d'application (“stateless”).



L'explosion du cloud computing

Le principe du cloud computing : exploiter la puissance de calcul et de stockage de serveurs distants au travers d'Internet.

Les entreprises n'ont plus besoin de posséder les ressources informatiques, elles les louent à l'utilisation.

Par exemple : AWS, \$0.015 / heure pour une instance Linux.

Permet à une entreprise de **scaler** très rapidement ses services pour s'adapter aux pics d'utilisation.



Une troisième rupture :

Les systèmes NoSQL



Les modèles relationnels

Les bases de données relationnelles sont les modèles les plus courants aujourd'hui.

Elles offrent de nombreux avantages :

- Interface d'accès (presque) standard : SQL
- Respect des contraintes ACID :
 - Atomicité, cohérence, isolation, durabilité.
- Des connecteurs existent pour la plupart des langages et plateformes.



Ils ont aussi leurs limites

Cependant, elles ne sont pas adaptées à toutes les situations :

- Temps de réponses dégradés sur de très gros volumes.
- Nombre d'accès concurrents limité.
- Scalabilité moins évidente.
- Le modèle relationnel n'est pas adapté à toutes les problématiques !



De nouvelles solutions apparaissent

En conséquence à ces limites des systèmes traditionnels, de grands acteurs de l'Internet ont conçu de nouveaux systèmes, spécifiquement adaptés à leurs besoins :

- Constraintes en terme de temps de réponse.
- Plus adaptés au déploiement distribué (**scalabilité**).
- Nouvelle représentation des données ou de la façon de les interroger (bases orientées graphes, stockage orienté recherche, ...).



“NoSQL” : un nom unique - de nombreuses solutions

Derrière l'étiquette “NoSQL” existent différents outils, très différents, adaptés à des cas d'utilisations spécifiques.

A la différence des systèmes relationnels, les systèmes NoSQL se sont fortement spécialisés en fonction des usages.

Nous allons voir quelques exemples...



Stockage orienté document



- ▶ L'unité de stockage n'est plus une ligne composée de colonnes, mais un document.
- ▶ Peut être du JSON, de l'XML, ...
- ▶ Absence de schéma : tous les documents n'ont pas forcément la même structure.
- ▶ Avantages : simplicité, scalabilité, performances.
- ▶ Pas de notion de clé étrangère : on perd la consistance, mais on évite les jointures, coûteuses en performance.





Stockage clé-valeur

- ▶ Une sorte de “HashMap” distribuée : on associe à une clé une valeur qui peut être une simple chaîne de caractère ou un objet sérialisé.
- ▶ Très performants : conçus pour travailler principalement en mémoire.
- ▶ Le modèle de stockage / requêtage change fortement : stocker les mêmes données plusieurs fois avec des clés différentes.
- ▶ Adapté à du cache applicatif par exemple.



Base de données orientées graphe



- Ici l'objectif n'est pas d'optimiser les performances ou la scalabilité, mais de changer le modèle de requêtage.
- Stockage des données sous forme de noeuds et de liens entre ces noeuds.
- Requêtage par parcours de graphe.
- Adapté pour représenter certains domaines inadaptés aux bases relationnelles : les liens d'un réseau social par exemple.



Un changement pour les applications

Ces nouveaux systèmes apportent chacun des concepts nouveaux et des contraintes différentes sur le design des applications :

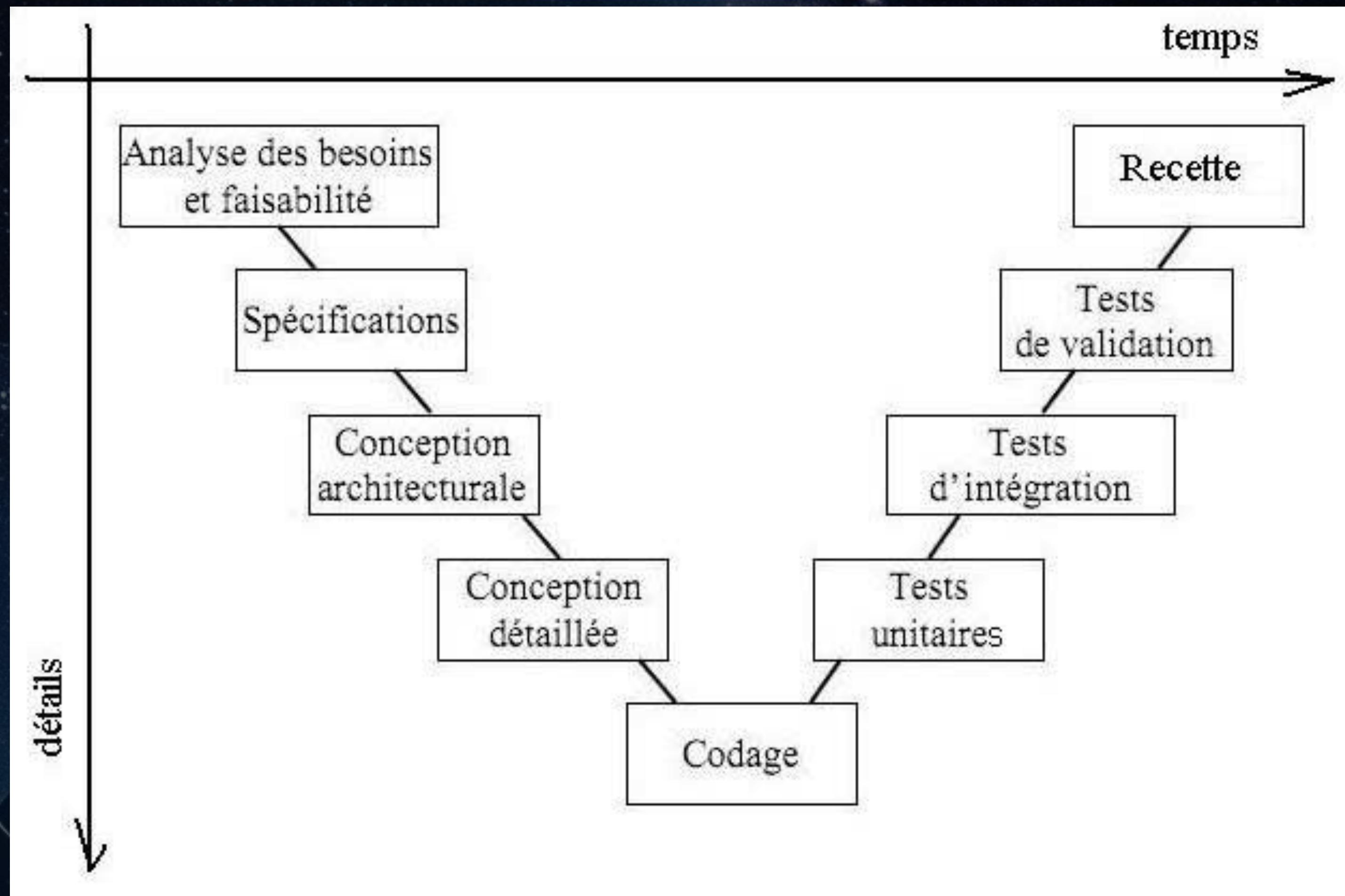
- Pas de langage unifié de requêtage mais des syntaxes spécifiques à chaque outil.
- Nécessite de repenser la façon de stocker les données : dénormalisation, choisir entre optimiser les écritures ou les lectures, ...
- Connecteurs spécifiques à chaque système (pas de JDBC !).



Une quatrième rupture :

Les nouvelles méthodologies





Cycle en V

La différence entre la théorie et la pratique...

- ▶ En théorie, une fois une phase terminée, elle est considérée comme définitive et ne sera plus remise en cause.
- ▶ En pratique... il est très fréquent de rencontrer des problèmes lors de la phase d'implémentation qui nécessite de revoir l'architecture technique ou fonctionnelle.
- ▶ De plus, **ce modèle augmente le risque** de décalage entre le besoin initial et l'application finale.
- ▶ Besoin de plus de flexibilité !



Les cycles courts et les méthodes agiles

La tendance est au développement avec des cycles très courts :

- Design itératif.
- Livrer peu à chaque fois mais livrer souvent.
- Obtenir du feedback plus rapidement.
- Permet de minimiser le décalage entre le besoin initial et l'application finale.



Déployer en continu

Les façons de livrer des nouvelles versions des applications évoluent en conséquence :

- Remplacer les “grosses” releases 3 fois par an par de petites releases tous les mois par exemple.
- Moins de fonctionnalités à chaque fois, mais plus souvent.
- Le risque est minimisé, la satisfaction des utilisateurs est améliorée.



Déployer en continu

Les “pure-players” vont jusqu’à déployer des nouvelles versions en production tous les jours, plusieurs fois par jours, sans que l’utilisateur ne s’en rendent compte !

Même sans aller à ces extrêmes, livrer régulièrement ce qui est terminé améliore la réactivité.

Cela nécessite des méthodologies, des outils, des architectures adaptées.



Quelques solutions

- Outils d'intégration continue pour compiler et tester automatiquement ce qui est développé.
- Les déploiements doivent être systématiquement automatisés.
- ... mais les rollbacks aussi ;-)
- Docker a fortement transformé la façon de livrer les applications.



Échauffement

Javascript, NodeJS, Typescript



Installation - 1ère étape



Installation de Node.js et Yarn

Pour les utilisateurs de MacOS, je vous conseille d'utiliser le gestionnaire de package [Homebrew](#) (<https://brew.sh/>)

Une fois que vous avez [Homebrew](#) sur votre machine, afin d'installer Node.js, Npm et Yarn, tapez dans votre [terminal](#)

```
$ brew install node
```

```
$ brew install yarn
```

Procédure expliquée ici => <https://classic.yarnpkg.com/en/docs/install#mac-stable>



Installation de Node.js et Yarn

Pour les utilisateurs de Ubuntu ou Debian, vous devez d'abord installer Node.js v20.x comme indiqué dans ce [lien](#) (<https://github.com/nodesource/distributions/blob/master/README.md#installation-instructions>)

Une fois que vous avez Node.js sur votre machine, afin d'installer Yarn, tapez dans votre terminal

```
$ curl -o- -L https://yarnpkg.com/install.sh | bash
```

Procédure expliquée ici => <https://classic.yarnpkg.com/en/docs/install/#alternatives-stable>



Alternative : Installation de nvm

Nvm permet d'installer rapidement plusieurs versions de NodeJS et la version nom correspondante.

<https://github.com/nvm-sh/nvm#installing-and-updating>





Npm, Yarn c'est quoi ?



Gestionnaire de paquets javascript

- ▶ Bibliothèque de programmation javascript pour NodeJS
- ▶ Similaire à Maven
- ▶ <https://www.npmjs.com/>



Différence entre Npm et Yarn

- Npm => Node Package Manager
- Installation des dépendances en séquentiel
- Génère un fichier package.lock.json
- Yarn => Yet Another Resource Negotiator
- Installation des dépendances parallèles
- Devenu plus une alternative après les progrès de Npm



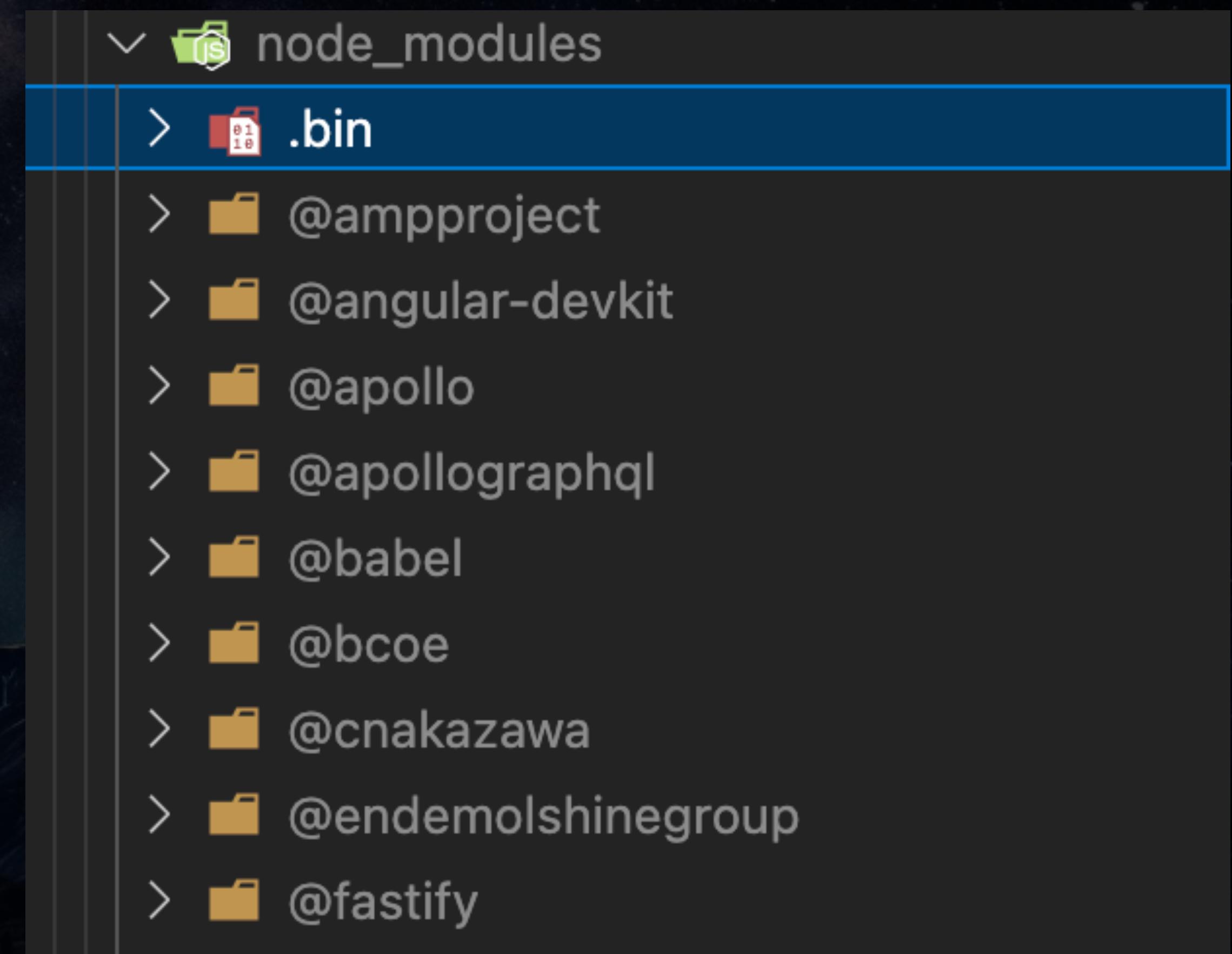
Liste des commandes

Command	NPM	Yarn
Initialize a project	<code>npm init</code>	<code>yarn init</code>
Run tests for the current package	<code>npm test</code>	<code>yarn test</code>
Check for outdated packages	<code>npm outdated</code>	<code>yarn outdated</code>
Publish a package	<code>npm publish</code>	<code>yarn publish</code>
Run a script	<code>npm run</code>	<code>yarn run</code>
Manage local package cache	<code>npm cache clean</code>	<code>yarn cache clean</code>
Log in or out	<code>npm login/logout</code>	<code>yarn login/logout</code>
Install dependencies	<code>npm install</code>	<code>yarn</code>
Install packages	<code>npm install [package name]</code>	<code>yarn add [package name]</code>
Uninstall packages	<code>npm uninstall [package name]</code>	<code>yarn remove [package name]</code>
Update manager	<code>npm update</code>	<code>yarn upgrade</code>
Update packages	<code>npm update [package name]</code>	<code>yarn upgrade [package name]</code>
Install packages globally	<code>npm install --global [package name]</code>	<code>yarn global add [package name]</code>
Uninstall packages globally	<code>npm uninstall --global [package name]</code>	<code>yarn global remove [package name]</code>
Interactive dependency update	<code>npm run upgrade-interactive</code>	<code>yarn upgrade-interactive</code>



Les node modules

- ▶ Dépendances installées dans votre projet
(\$ `yarn` ou \$ `npm i`)
- ▶ Ajouter des dépendances en local
 - ▶ \$ `yarn add rxjs`



Déroulement de la formation

Github de la formation

<https://www.github.com/simonh88/nwt-school-basics>



Déroulement de la formation

```
$ git clone https://www.github.com/simonh88/nwt-school-basics
```



Exercice 1 : Hello world

```
git checkout -f step-01
```

Initialiser un projet et executer votre javascript



Exercice 1 : Hello world

- Initialiser votre projet
 - `yarn init`
- Examiner les fichiers générés
- Créer un fichier `index.js` et faire en sorte qu'à l'execution de ce dernier `Hello World` soit affiché dans la console
 - Pour tester votre fichier `index.js` executer la commande suivante
 - `node index.js`
- Trouver un `autre moyen` de lancer le programme en modifiant le `package.json`
 - Aide ici ↗ et regarder du côté de la commande `$ yarn run nom_du_script`



Solution

```
git checkout -f step-01-solution
```



Exercice 2 : Fonctions magiques

`git checkout -f step-02`

Jouer un peu avec les fonctions de javascript



Exercice 2 : Fonctions magiques

- ▶ Modifier les trois fonctions et executer le test
- ▶ Il faudra obtenir en output ceci :

```
→ nwt-school-basics git:(step-02-solution) node index.js
++++++ sortingNumbers 0K
++++++ sortingObjects 0K
++++++ addPropertyToObjects 0K
```

- ▶ - - - - NOK signifie que la modification de votre fonction n'est pas suffisante
- ▶ Aide ici ➔ et bien sûr google

Solution

```
git checkout -f step-02-solution
```



Exercice 3 : Typescript

`git checkout -f step-03`

(Re)-Découvrir Typscript



Exercice 2 : Typescript

- Il y a maintenant un fichier `index.ts`
 - **Examiner** le fichier
 - Trouver un moyen d'**executer** ce fichier
-
- Ne pas hésiter à utiliser **Google**. Plusieurs solutions sont possibles



Solution

git checkout -f step-03-solution

- ▶ \$ yarn add typescript -D
- ▶ \$./node_modules/.bin/tsc index.ts
- ▶ ● → **nwt-school-basics** git:(step-03-solution) ✘ node index.js
Voilà les détails de Toto : {"nom":"TUTU","prenom":"Toto","age":25}



Fin de l'échauffement

- ▶ Ce n'était qu'une très rapide mise en jambes
- ▶ Quelques liens pour avoir les bases en [javascript](#), [typescript](#) et [nodeJS](#)
- ▶ [Javascript](#)
- ▶ [TypeScript](#)
- ▶ [NodeJS](#) ou [la partie 1 NodeJS tutorial](#)
- ▶ Si vous avez le temps voir [RxJS](#) => Au moins comprendre différence entre `map` et `mergeMap`, ce qu'est un [Observable](#)

