Développement WEB

Full Stack Web Development

Qui suis-je ?

Jordan SABLON *jordansablon.enseignement [at] gmail.com*

→ **8 ans** d'expérience dans le développement WEB

Développeur Full Stack chez catamania DIGITAL process for HUMAN progress depuis mai 2024

Développeur Full Stack chez DATA LEGAL DRIVE de septembre 2021 à mai 2024

Développeur JAVA/EE chez sopra steria de mars 2017 à septembre 2021

Ancien étudiant de **l'EIL Côte d'Opale**, Calais

Objectifs du module

→ Découvrir l'univers du développement WEB

→ Comprendre le rôle d'un développeur Full Stack

→ S'informer sur les langages et les outils existants

→ Aborder les principes de base d'architecture d'une application WEB

→ Créer une application WEB « *from scratch* » (en partant de rien)

Organisation du module

28 heures par étudiant découpées comme suit :

- 4 heures CM → présentation de notions génériques
- 16 heures TD → découverte et mise en application des différentes notions
- 8 heures TP → développement d'un projet complet en autonomie

Déroulement :

- 1 séance de CM d'initiation au développement WEB de 2h
- 2 séances de TD de 2h par notion à aborder
  - présentation théorique
  - exercices pratiques encadrés

  x 4 notions
- 2 séances de TP de 4h
- 1 évaluation intermédiaire
- 1 examen final sur machine

Les notions abordées

Partie n°1 – Backend avec Node.js et Express.js

Mise en place de la couche de traitements et d'accès aux données

Partie n°2 – Base de données

Sélection, configuration et interaction avec une base de données

Partie n°3 – Frontend avec React

Création d'une interface utilisateur

Partie n°4 – Interactions Frontend <> Backend

Réaliser des appels au backend depuis le frontend

01

Initiation au développement WEB

Sommaire

→ Qu’est-ce que le développement web ?

→ « Full Stack Web Developer », en quoi ça consiste ?

→ Adopter une architecture web

→ Choisir une stack technique par rapport à ses besoins

→ Quels outils utiliser ?

→ Installation & configuration de l’environnement de développement

Qu’est-ce que le développement WEB ?

« Discipline qui consiste, par l’usage de langages de programmation web, à programmer des sites web ou applications web destinées à être publiés sur des serveurs »

	Site Internet	Application Web
Objectif principal	Facilitez la navigation et l'extraction des informations pertinentes pour les utilisateurs qui répondent à leurs besoins	Être réactif aux actions des utilisateurs; être interactif et fournir aux utilisateurs la possibilité de manipuler des données et de faire des demandes pour différentes sorties
Principales caractéristiques et avantages	Accès facile, mise à jour facile, gain de temps et d'argent, publicité facile, satisfaction du client	Expérience personnalisée, évolutivité, optimisation des capacités des appareils, satisfaction client
Éléments principaux	Langage de balisage hypertexte (HTML), feuilles de style en cascade (CSS) et JavaScript	HTML, CSS et JavaScript; utilise en outre des langages de programmation tels que Ruby, PHP et des frameworks tels que Ruby on Rails, Scriptcase, Django et base de données

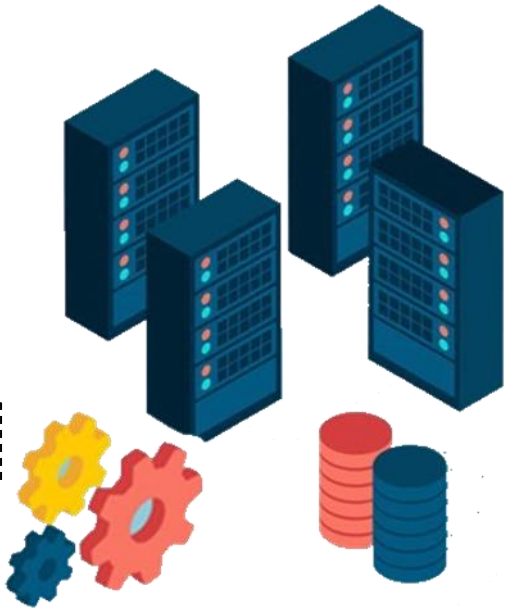
Qu’est-ce que le développement WEB ?

Saisie une adresse

Affiche le résultat

Requête envoyée

Contenu retourné

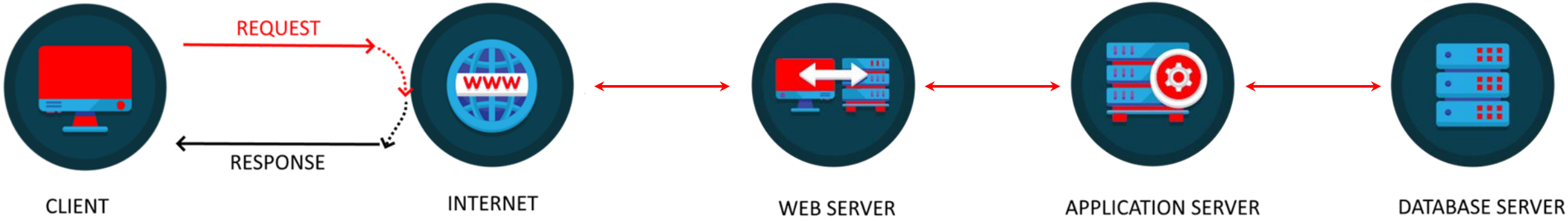


Utilisateur

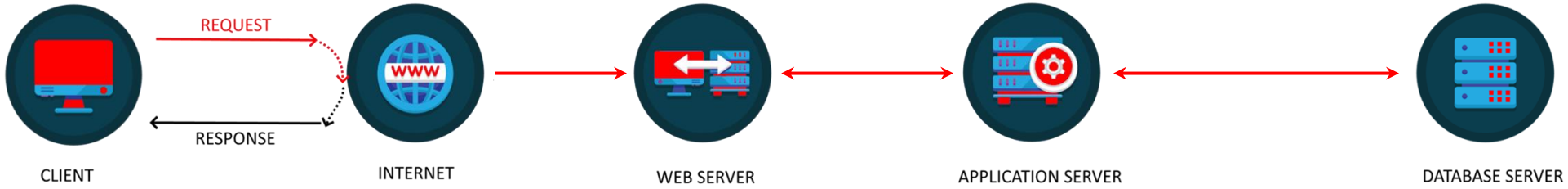
Navigateur

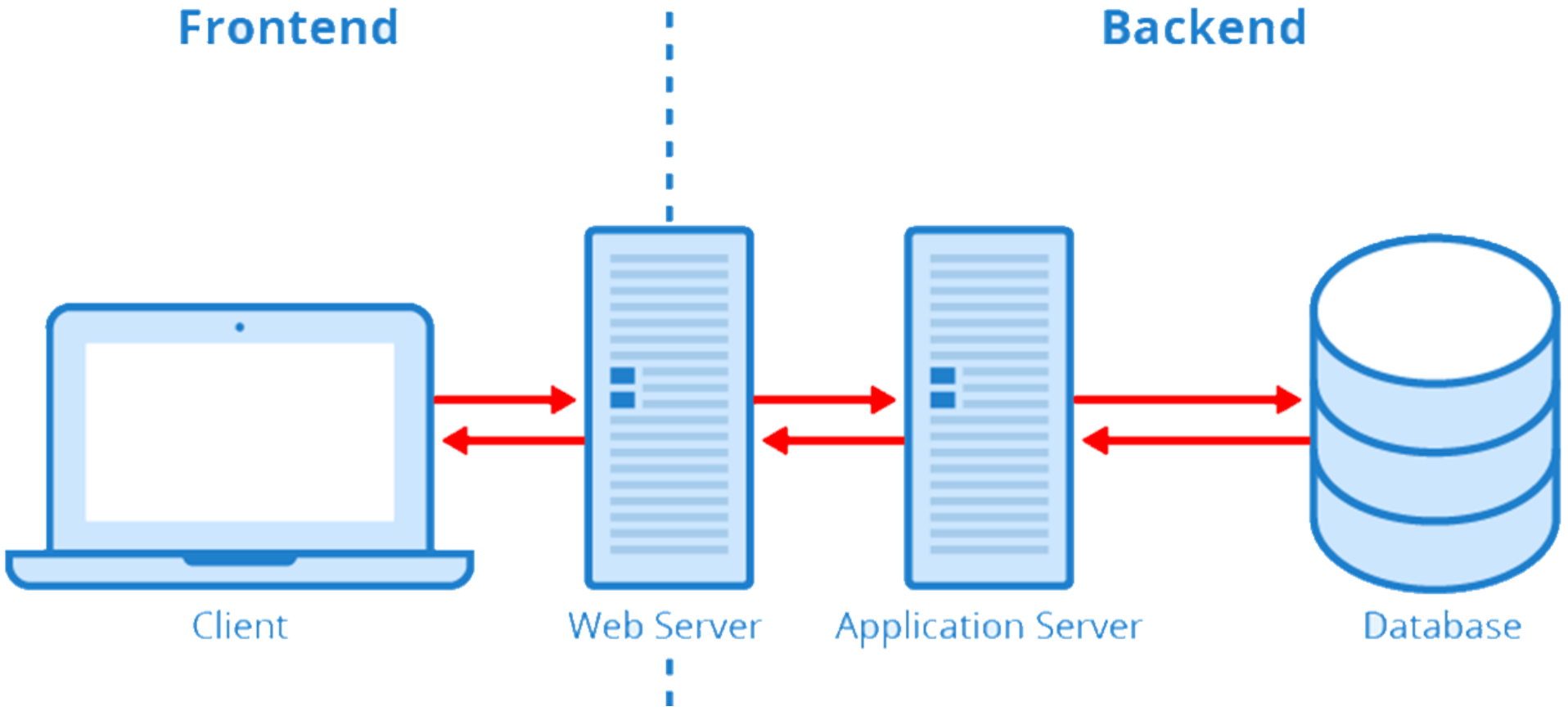
Serveur

Qu’est-ce que le développement WEB ?

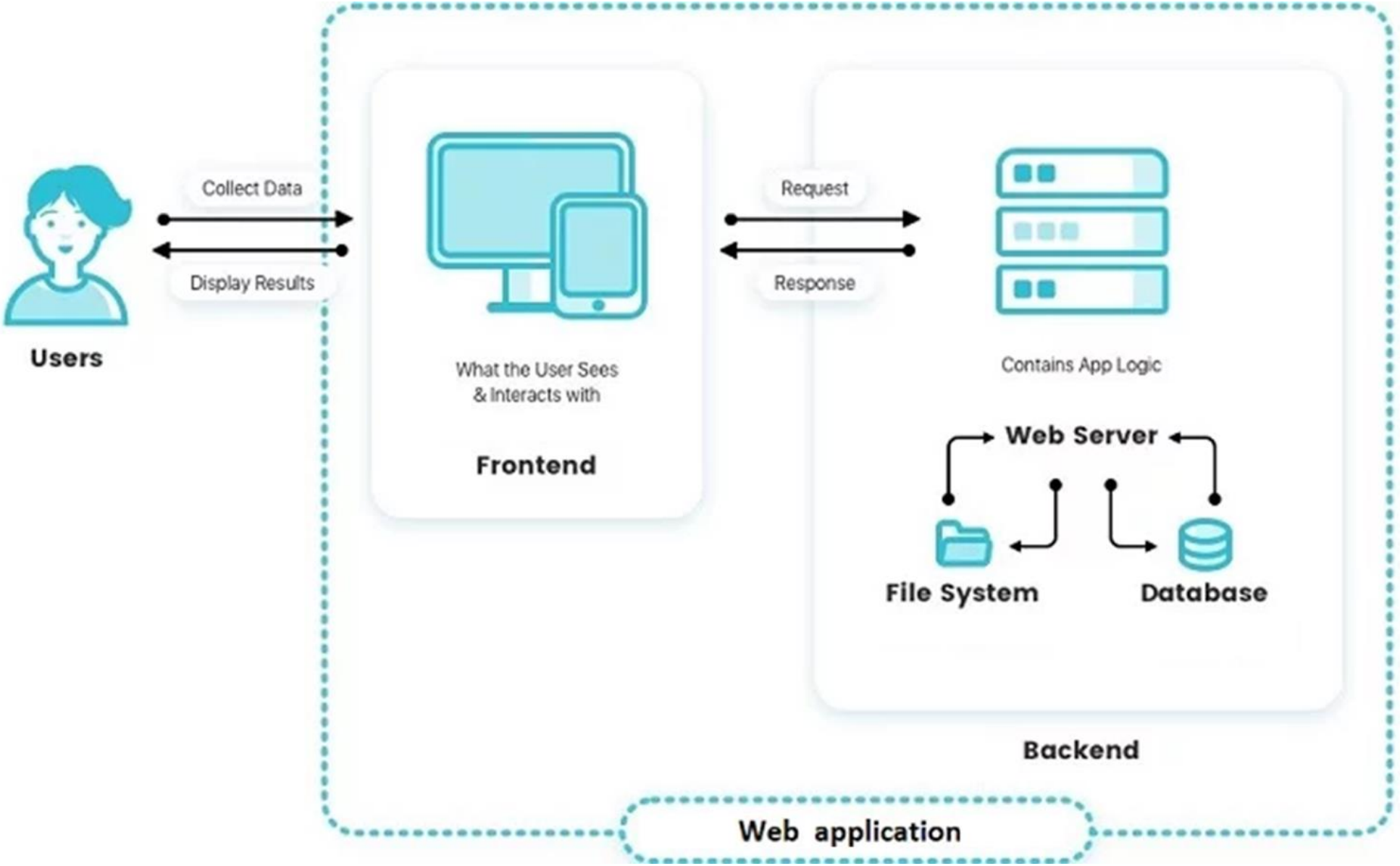


Qu’est-ce que le développement WEB ?





Qu’est-ce que le développement WEB ?



Frontend

« En développement web, la notion de « front end » fait référence à l’ensemble des éléments visibles et accessibles directement sur un site web (voire sur une application web ou une application web mobile) »

→ On parle d’**interface utilisateur** ou **IHM** (**I**nterface **H**omme-**M**achine)

→ Regroupe l’ensemble des contenus visibles

→ Permet d’interagir avec l’application (clics, navigations, saisies d’informations via des formulaires, etc.)

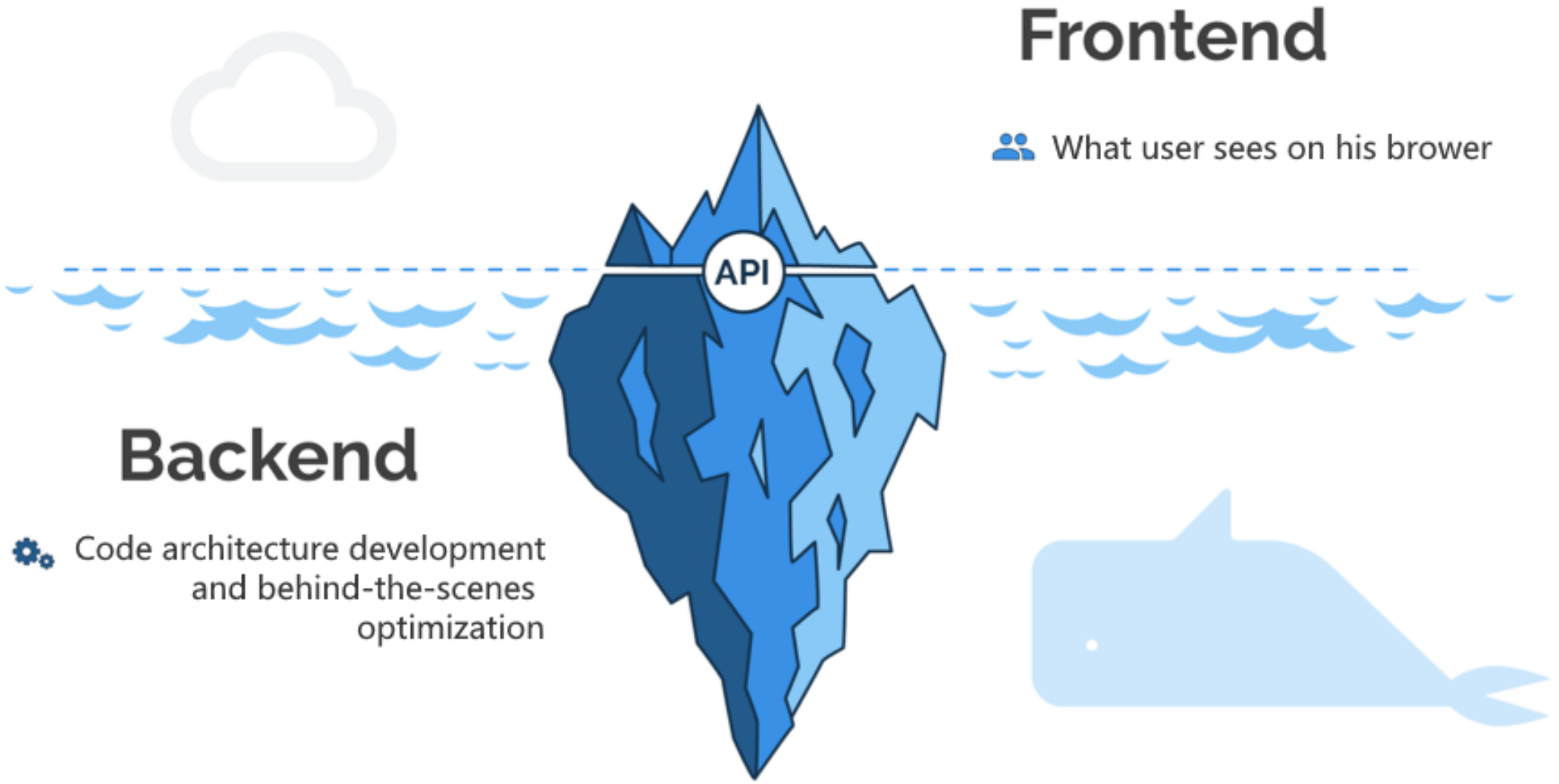
Backend

« En développement web, la notion de « back-end » se réfère à toute la partie invisible pour l’utilisateur, mais qui va permettre le bon fonctionnement d’une application web. Il s’agit donc de mettre en place la programmation au niveau du serveur, pour exécuter les requêtes qui sont réalisées sur le site web par les internautes. »

→ Regroupe l’ensemble de la logique métier (*Business Logic*)

→ Permet d’associer un traitement pour chaque action utilisateur

Frontend ⇔ Backend



Base de données

« Une base de données est une collection organisée d’informations structurées, généralement stockées électroniquement dans un système informatique.

Une base de données est généralement contrôlée par un système de gestion de base de données (SGBD).

L’ensemble que constituent les données et le SGBD, ainsi que les applications qui leur sont associées, est nommé système de base de données, ou simplement base de données.. »

Base de données

Bases de données relationnelles : ensemble de tables comportant des lignes et des colonnes offrant le moyen le plus efficace et flexible d’accéder à des informations structurées.

Bases de données orientées objet : les informations sont représentées sous forme d’objets

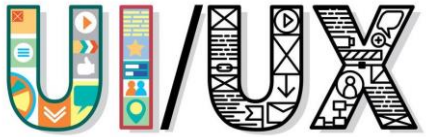
Bases de données distribuées : composé de fichiers ou plus, situés dans différents sites.

Data warehouses : référentiel central de données spécifiquement conçu pour permettre une interrogation et une analyse rapides des données.

Bases de données NoSQL : permet de stocker et de manipuler des données non structurées

Bases de données orientées graphe : stocke des données en termes d’entités et de relations entre les entités.

Quelques notions à prendre en compte



Je m’assure que mon application soit…

Responsive

- S’adapter à toutes les résolutions
- Un seul site pour plusieurs supports
- Trafic sur mobile **>** Trafic sur ordinateur
- Impact sur le référencement (Mobile First de Google)

Je m’assure que mon application soit…

Accessible

- Définit par les standards du web internationaux
- Application adaptée aux personnes en situation de handicap, mais pas que…
- Enrichie l’utilisation via des appareils connectés (montre, commande vocale, etc.)

Je m’assure que mon application soit…

Accessible

- **Quelques exemples**

✓ Alternative textuelle pour les images

✓ Saisie au clavier
(ex: navigation dans les formulaires)

✓ Transcription de l’audio

Je m’assure que mon application soit…

Évolutive

- Conception de l’application adaptée
- Minimiser la dette technique
- Choix technologiques pérennes
  - ✓ Adéquates aux besoins
  - ✓ Courbe d’apprentissage
  - ✓ Rapidité de mise en œuvre
  - ✓ Intégration dans l’existant
  - ✓ Importance de la communauté

Je m’assure que mon application soit…

Performante

- Améliorer l’utilisation de l’application
- Optimiser l’expérience utilisateur et donc sa satisfaction
- Doit être constamment surveillée lors des développements

Je m’assure que mon application soit…

Performante

- Anticipation de l’utilisation
  - ✓ Nombre de pages interrogées
  - ✓ Temps de réponse requis
  - ✓ Nombre de connexions simultanées
  - ✓ Les données transférées
  - ✓ Opérations par unité de temps

Je m’assure que mon application soit…

Performante

- Réalisation de tests avant déploiement
  - ✓ **Charge** : mesure de la performance pour la charge ciblée
  - ✓ **Stress** : comportement en cas de pic d’utilisation soudain
  - ✓ **Capacité** : sollicitation croissante sur une période donnée jusqu’à rupture
  - ✓ **Endurance** : activité importante pendant une durée prolongée
  - ✓ **Résilience** : fonctionnement en cas de panne sur l’infrastructure

Je m’assure que mon application soit…

Sécurisée

- Réduire les risques de piratage
- Protéger les données des utilisateurs
- Renforcer la confiance des utilisateurs envers l’application (crédibilité)
- Impact sur le référencement

Je m’assure que mon application soit…

Sécurisée

- Plusieurs risques possibles

✓ Injections SQL
✓ Cross-site scripting (XSS)
✓ Man-in-the-middle (man-in-the-browser)
✓ Détournement de session
✓ Empoisonnement de cookies
✓ Phishing

Je m’assure que mon application soit…

Sécurisée

- Sécuriser l’application

✓ Identifier et **authentifier** les utilisateurs
✓ Contrôler les accès
✓ Protéger et chiffrer les données
✓ Sécuriser les transactions (HTTPS)

Je m’assure que mon application soit…

Sécurisée

- Sécuriser l’infrastructure

✓ Présence de pare-feu pour filtrer le trafic entrant
✓ Protection contre les tentatives d’intrusion
✓ Protection contre les attaques par déni de service (DDoS)
✓ Monitoring
✓ Réaliser des backups réguliers

Je m’assure que mon application soit…

Compatible

- S’assurer que l’application fonctionne avec les principaux navigateurs
- Eviter le code spécifique
- Consulter les documentations techniques pour s’assurer de la compatibilité (ex: CSS)
- Tester l’application sous différents navigateurs

Je pense aussi à son…

Déploiement

- Choix de l’hébergement

✓ Compatible avec les technologies utilisées
✓ Tarif
✓ Disponibilité
✓ Bande passante
✓ Stockage
✓ Support
✓ Sécurité

Je pense aussi à son…

Déploiement

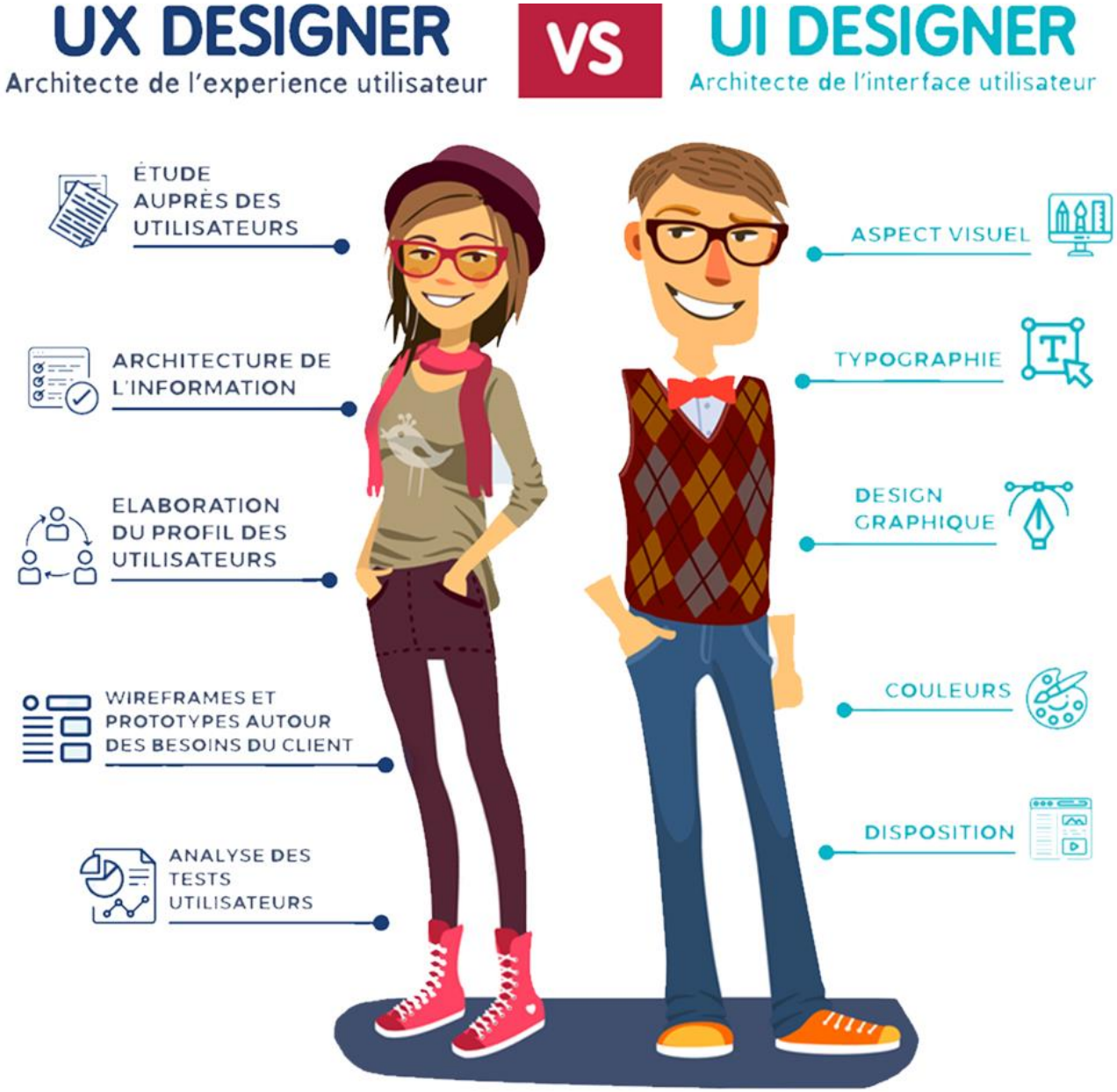
- Prendre en compte les contraintes de production lors du développement
- Automatisation du cycle de vie (tests, installation) via le CI/CD
- Gestion des versions du code et de la base de données
- Documentation des modifications

Je pense aussi à son…

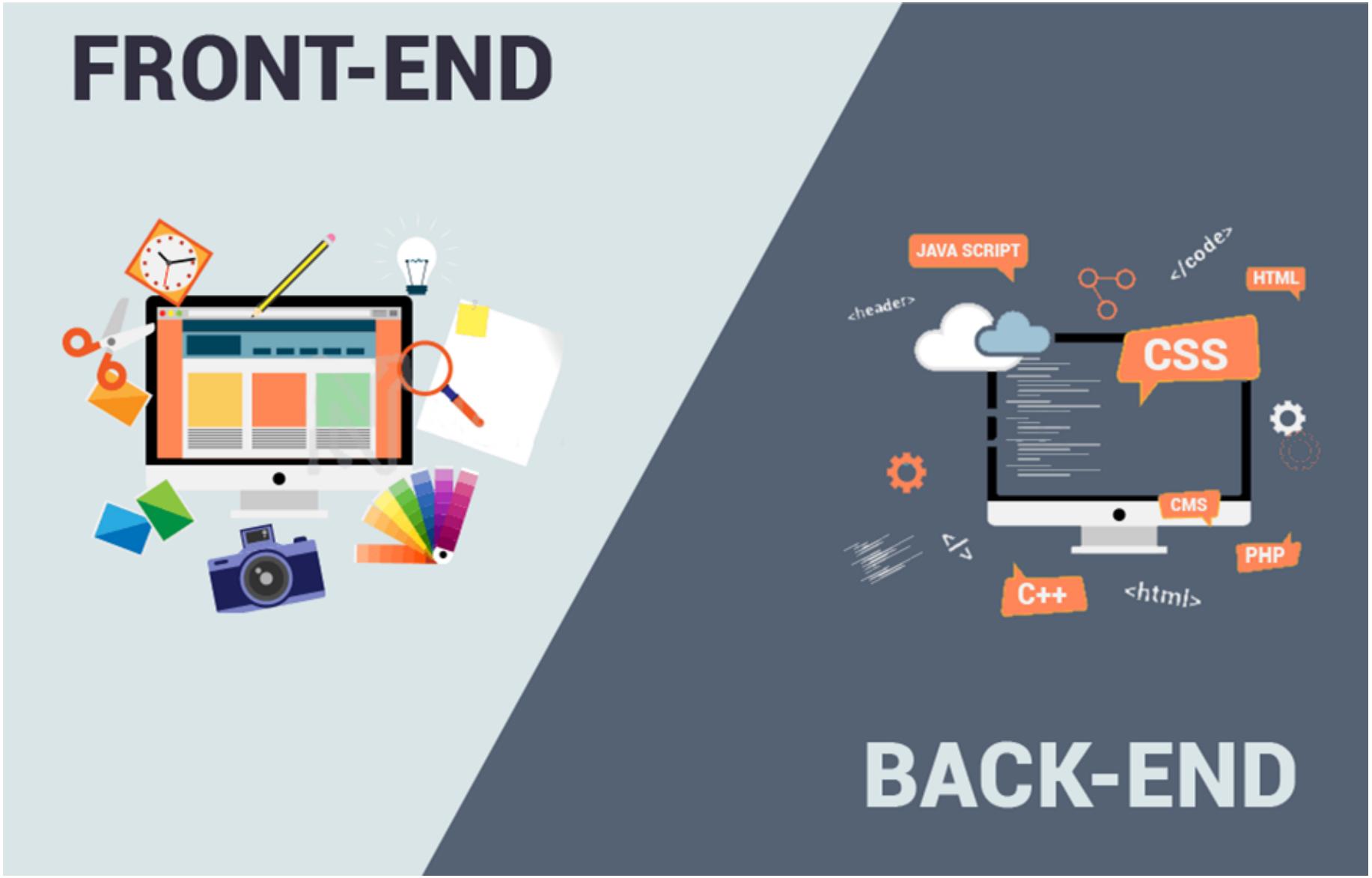
Utilisabilité

- Interface ergonomique, accessible et facilement utilisable
- Disposition intuitive des éléments
- Homogénéité des éléments graphiques
- Expérience utilisateur optimisée

Qu’est-ce que le développement WEB ?



« Full Stack Web Developer », kezako ?



« Full Stack Web Developer », kezako ?

Qu'est-ce qu'un développeur full stack ?

« Le Développeur Full-Stack travaille à la fois sur la partie visible de l'interface et avec laquelle l'utilisateur va interagir (Front-end) et sur la partie technique de l'interface (Back-end). »

→ Profil polyvalent capable de gérer un projet de développement web de A à Z

→ Double compétence frontend & backend

→ Développeur généraliste

→ Prisé par les entreprises

Quelles sont ses missions ?

Analyser le besoin

Etablit la faisabilité et la cohérence du développement en échangeant avec les personnes à l'origine de la demande et définit les besoins en termes de ressources techniques et humaines.

Concevoir et développer

Détermine la solution technique la mieux adapté au besoin énoncé, la documente et assure son implémentation sans oublier les tests.

Quelles sont ses missions ?

Faire évoluer le produit

Assure une veille technologique constante afin de sélectionner d'éventuelles nouvelles technologies permettant d'améliorer l'application.

Garantir sa performance

Contribue à la résolution d'éventuels bugs et anomalies techniques afin de garantir le fonctionnement de l'application.

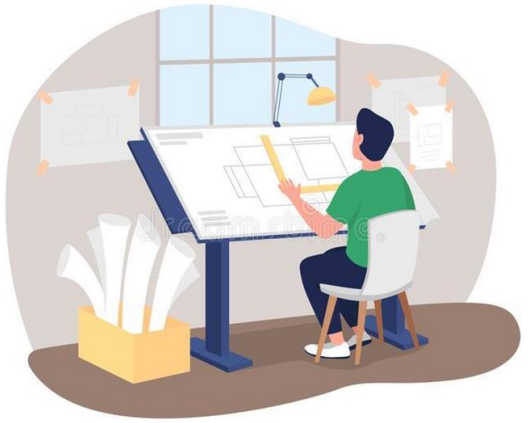
Maintient le code à jour afin de limiter la dette technique.

Qu’est-ce que l’architecture web ?

→ Décrit les interactions entre les applications, les bases de données et les systèmes middleware

→ Cadre technique qui permet d’établir des relations et des interactions entre tous les composants de l’application

→ Permet de traiter de l’efficacité, de la fiabilité, de l’évolutivité, de la sécurité et de la robustesse de l’application



Exemples de critères à prendre en compte

Testabilité de la logique métier

Si cette logique est liée à la base de données et au client, il sera alors très difficile d'écrire des tests automatisés qui puissent être exécutés en quelques secondes.

Si tous les développeurs perdent plusieurs minutes pour vérifier que la suite de tests ne retourne pas d'erreurs, c'est une perte importante de productivité.

Exemples de critères à prendre en compte

Paralysie technique

S'il est impossible de changer une partie de l'infrastructure sans avoir à réécrire une large portion du code, l'application n'évoluera pas et sera jetée à la poubelle pour en refaire une nouvelle.

Choix techniques erronés

Avec une mauvaise architecture il est impossible de revenir en arrière. Si un choix technique se trouve être bloquant durant le développement, revenir en arrière sera impossible et le projet devra être repris à partir de zéro.

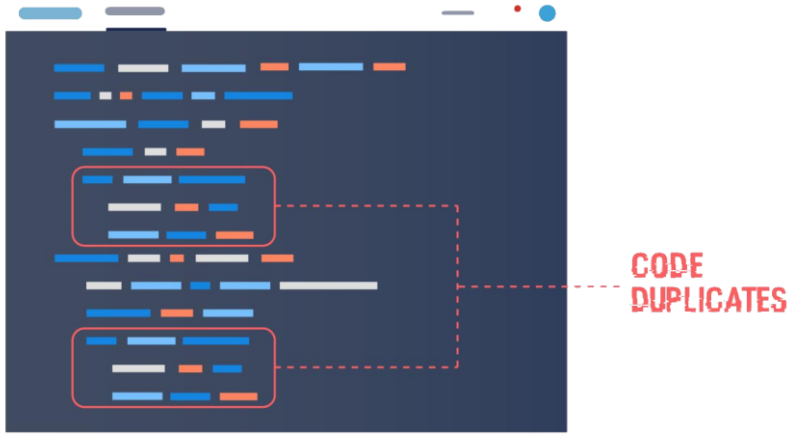
Exemples de critères à prendre en compte

Duplication du code

Sans architecture cohérente, la duplication de code sera inévitable.

Par exemple, deux éléments de l'application voudront faire la même chose mais l'architecture ne le permettra pas, la duplication sera la seule solution.

→ Il faut éviter cela à tout prix !



Quelques exemples d'architectures

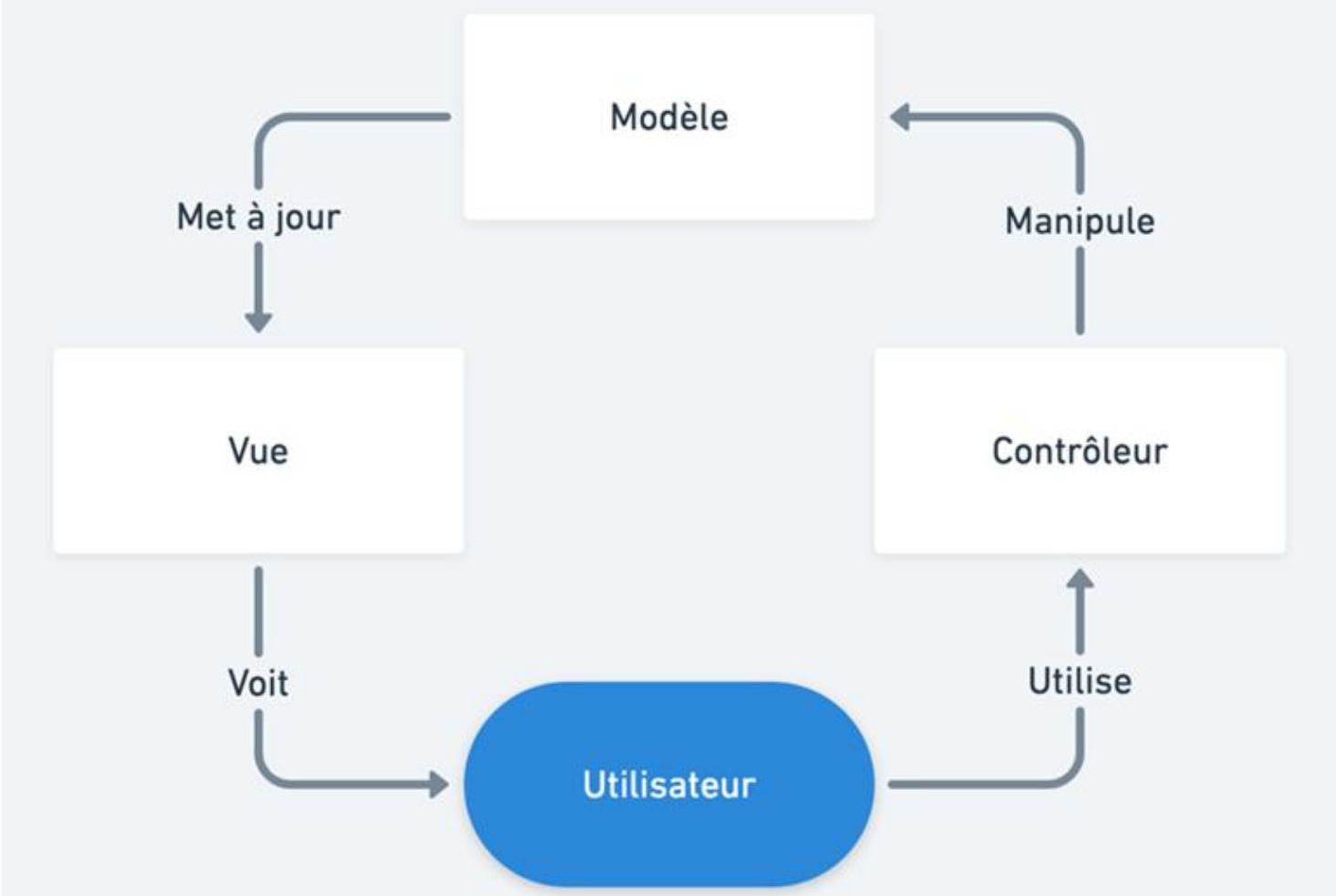
Modèle-Vue-Controller (MVC)

Cela se traduit par le motif d'architecture MVC, qui se décompose comme suit :

- Les **modèles** communiquent avec la base de données
- Les **vues** sont faites pour la présentation de l'interface utilisateur
- Les **contrôleurs** incluent les actions effectuées par les utilisateurs

Quelques exemples d'architectures

Modèle-Vue-Controller (MVC)



Quelques exemples d'architectures

Modèle-Vue-Controller (MVC)

→ Accès aux données totalement séparé du reste de l'application

→ Pas de dépendances entre l'interface utilisateur et la logique métier

→ Réduction de la duplication de code

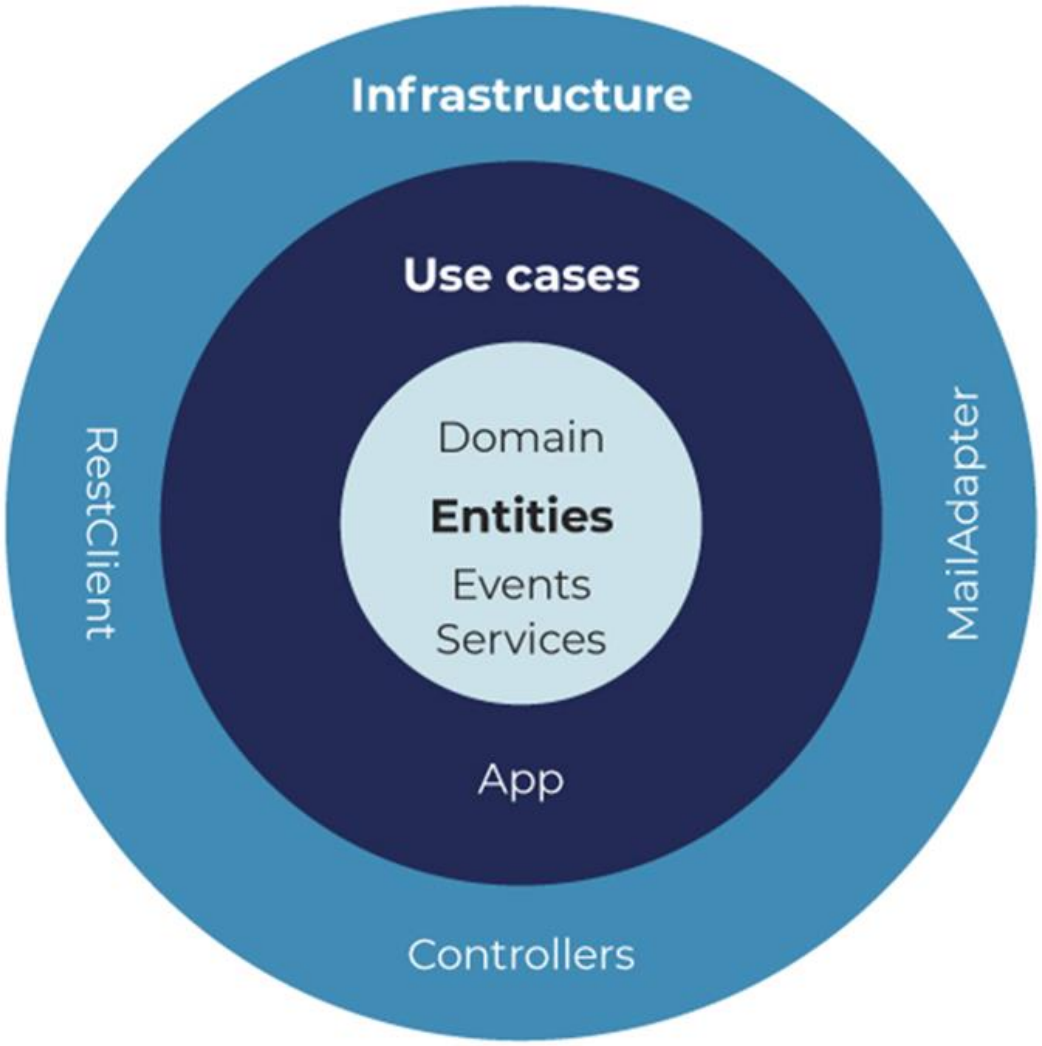
Quelques exemples d'architectures

Clean architecture

La Clean Architecture vise à réduire les dépendances de la logique métier avec les services que consommés (API, base de données, frameworks, librairies tierces) afin de maintenir une application stable au cours de ses évolutions, de ses tests mais également lors de changements ou mises à jour des ressources externes.

Quelques exemples d'architectures

Clean architecture



Quelques exemples d'architectures

Clean architecture

→ Supprime la dépendance de la logique métier avec les autres composants de l'application (interface utilisateur, frameworks, base de données, services externes)

→ Abstraction des services externes pour supprimer les dépendances via des interfaces

→ Simplicité des tests, réalisables couche par couche

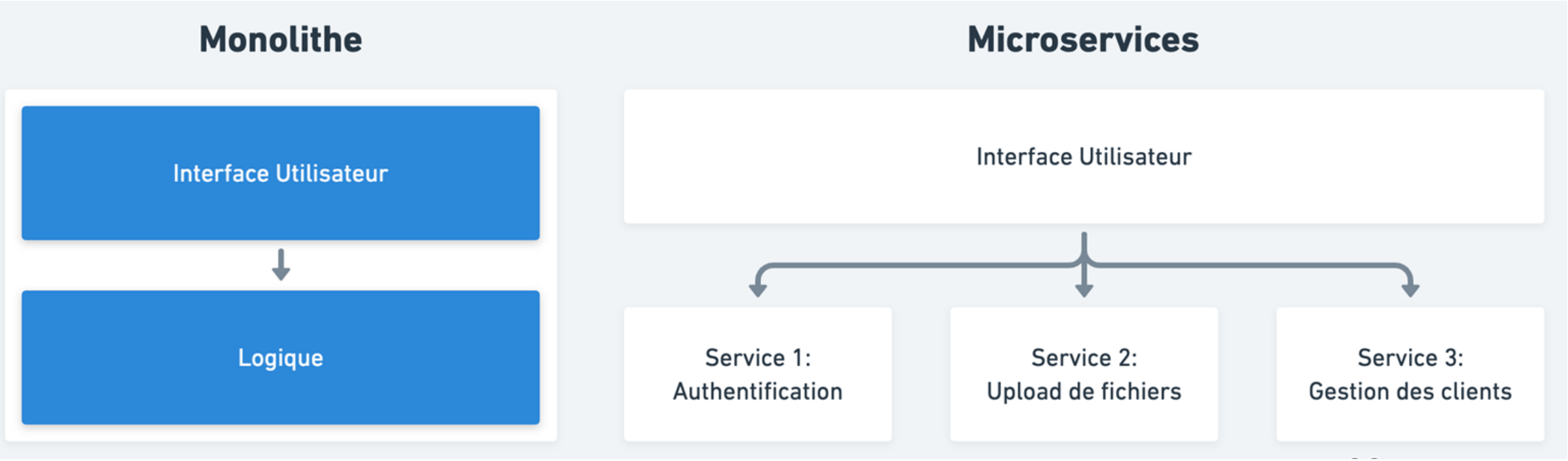
Quelques exemples d’architectures

Microservices

Le backend est découpé en plusieurs morceaux indépendants ce qui permet de faire évoluer un module sans toucher à un autre et d'allouer des ressources à certains services qui ont des demandes différentes du reste des services.

Quelques exemples d'architectures

Microservices



Quelques exemples d'architectures

Microservices

→ Résilience : aucun impact les uns sur les autres

→ Haute évolutivité : développement et déploiement modulaire = rapidité

→ Accessibilité / Maintenabilité : plus facilement compréhensible grâce à la décomposition structurée

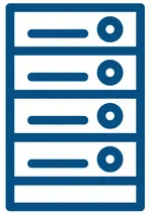
→ Scalabilité : possibilité d'étendre les ressources au fur et à mesure que la demande augmente pour certains services

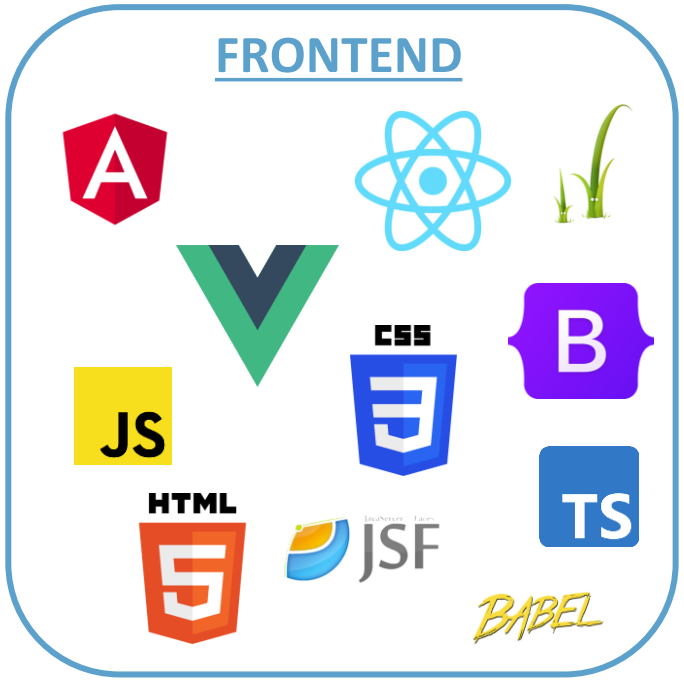
Qu’est-ce qu’une stack technique ?

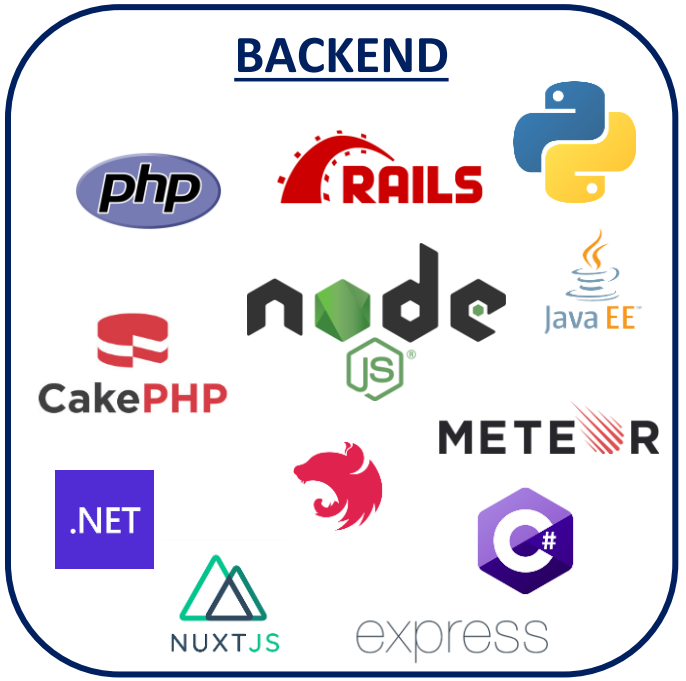
« Une stack technique, également appelée « tech stack », « pile de technologies » ou « écosystème de données », est une liste de tous les outils technologiques utilisés pour développer et faire fonctionner un programme. »

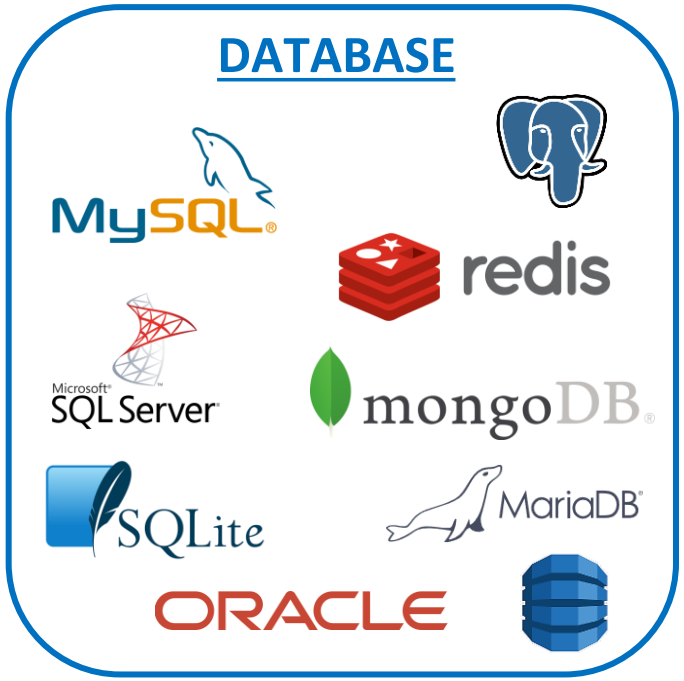
Choisir une stack technique par rapport à ses besoins

Quelques exemples de technologies









Quels critères ?

La maturité des technologies

→ S'orienter vers une technologie mature

→ Vérifier le pourcentage d'utilisation au fil du temps

→ Préférer une technologie avec une communauté active

→ S'assurer que la technologie est maintenue (ex: correction de bugs)

Quels critères ?

L'obsolescence

→ Ne pas choisir une technologie en déclin

→ Assurer une veille technologique pour s'informer sur les actualités

Selon les compétences de l'équipe de développement

→ Tenir compte des technologies déjà utilisées / connues

→ Prendre en considération la courbe d'apprentissage

Quels critères ?

Les besoins du projet

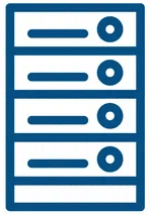
→ Adapter le choix aux besoins (performance, fiabilité, etc.)

→ Être pertinent selon l'ampleur du projet (POC, MVP, projet d'envergure, etc.)

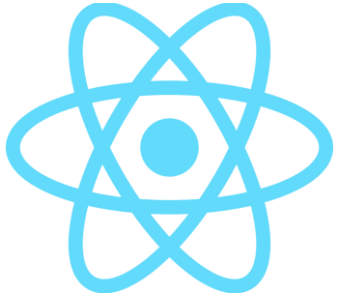
L'infrastructure

→ Prendre en compte les contraintes techniques liées à l'infrastructure (hébergement, complexité de déploiement, etc.)

La stack technique retenue



FRONTEND



React

BACKEND

Node.js

express

DATABASE

MongoDB

Quels outils utiliser ?

IDE

VCS

DB manager

Design

Others

02

Backend with Node.js and Express.js

Expres
s

Sommaire

Qu’est-ce que Node.js ?

« Node.js est un environnement d'exécution JavaScript asynchrone et orienté événement, conçu pour générer des applications extensibles. »

✓ Environnement bas niveau permettant l’exécution de JavaScript côté serveur

✓ Utilisé notamment comme plateforme de serveur Web

✓ Repose sur le moteur V8 de Google

Qu’est-ce que Node.js ?

→ Fréquemment utilisé pour écrire des services côté serveur (appelés API)

→ Permet de créer des traitements asynchrones

→ Alternative à d’autres langages serveur (PHP, Java, etc.)

→ Permet d’utiliser sur un seul langage pour le développement d’une application

Caractéristiques de Node.js?

Real Time Application (RTA)

→ Permet de créer des applications en temps réel, qui nécessite de se mettre à jour très fréquemment (WhatsApp, Facebook, etc.)

Single Page Application (SPA)

→ L'application ne contient qu'une seule page dont le contenu change en fonction des actions de l'utilisateur

Caractéristiques de Node.js?

Single Thread

→ Peut être un inconvénient dans certain cas d'usage (traitement lourd)

→ Complexité généralement faible côté serveur

→ Consiste généralement à procurer de la donnée pour alimenter les vues

Caractéristiques de Node.js?

Non-Blocking

→ Système non bloquant offrant la capacité de lancer une tâche sans attendre qu'elle se finisse pour passer à la suivante

→ Si une requête est demandée, elle est lancée sans attendre le résultat

→ Si une autre requête arrive, elle peut immédiatement être traitée

Caractéristiques de Node.js?

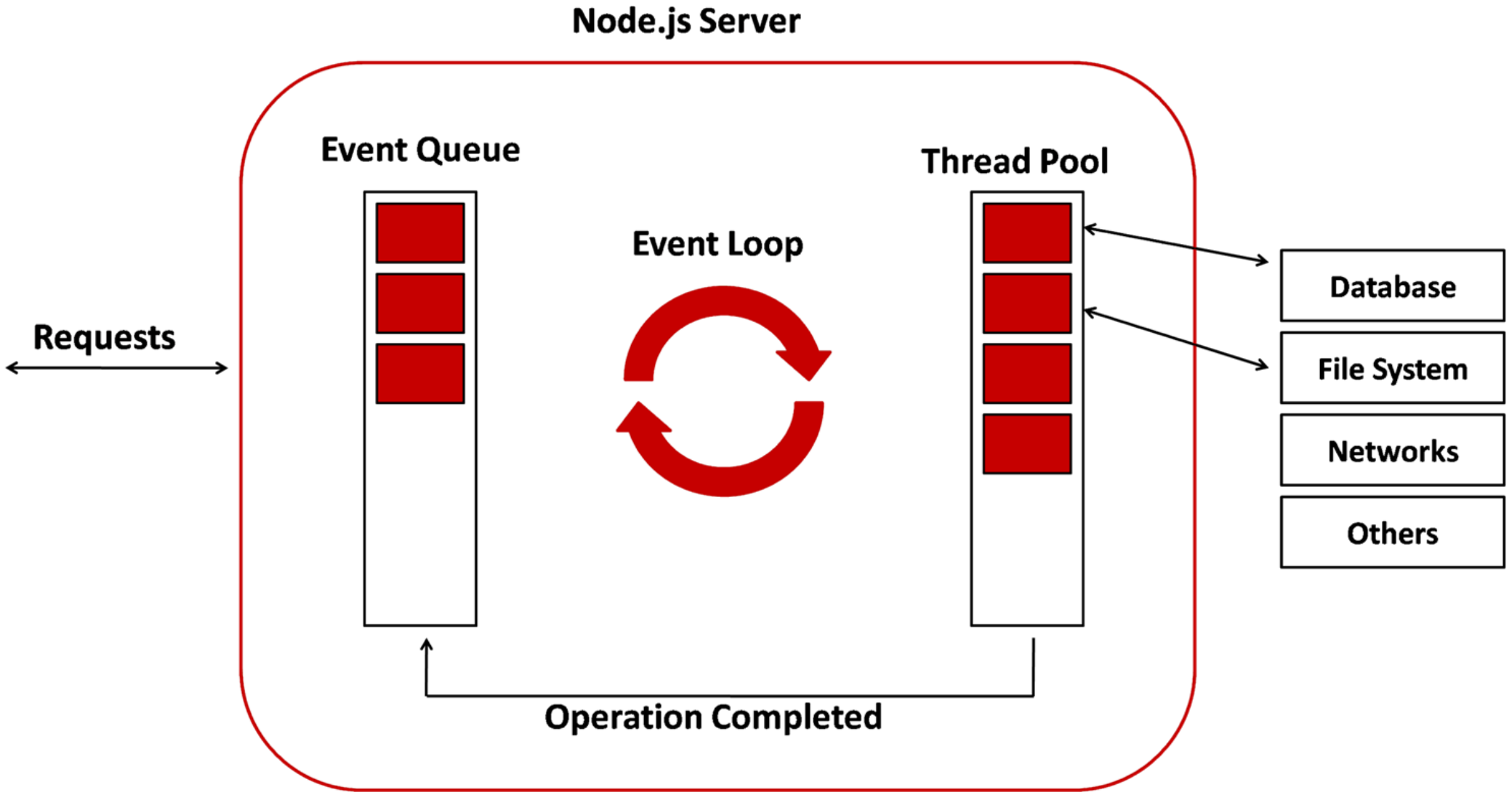
Flexible

→ Livré avec peu de fonctionnalités embarquées

→ Possibilité de choisir quels modules lui ajouter via NPM

→ Pas de convention d'écriture stricte

Comment fonctionne Node.js?

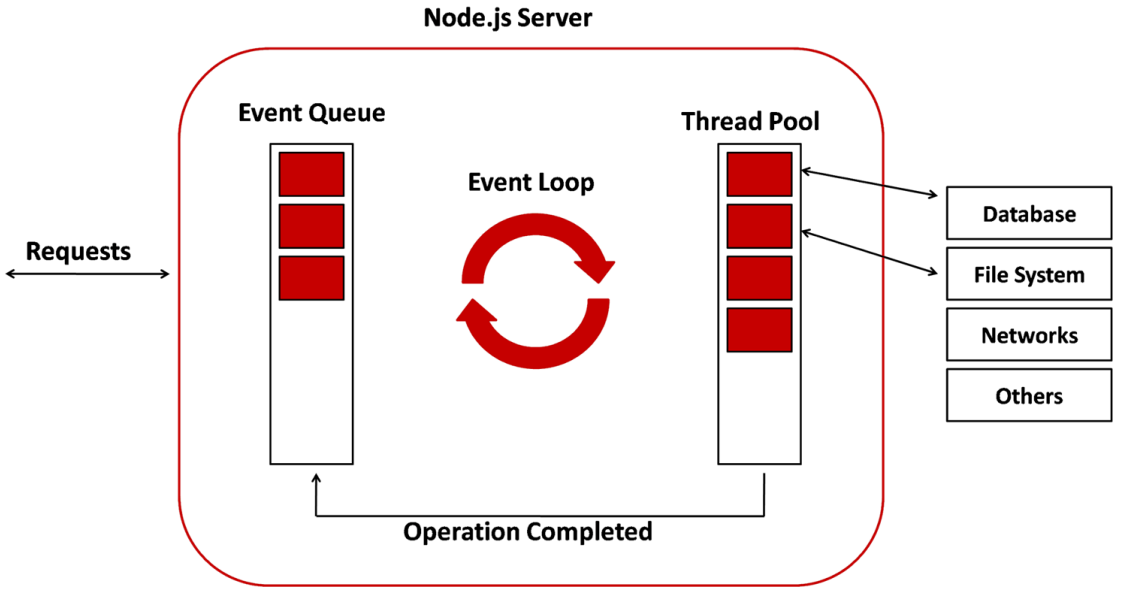


Comment fonctionne Node.js?

Requests

→ Ensemble des requêtes adressées au serveur Node.js

→ Les requêtes arrivent dans un certain ordre

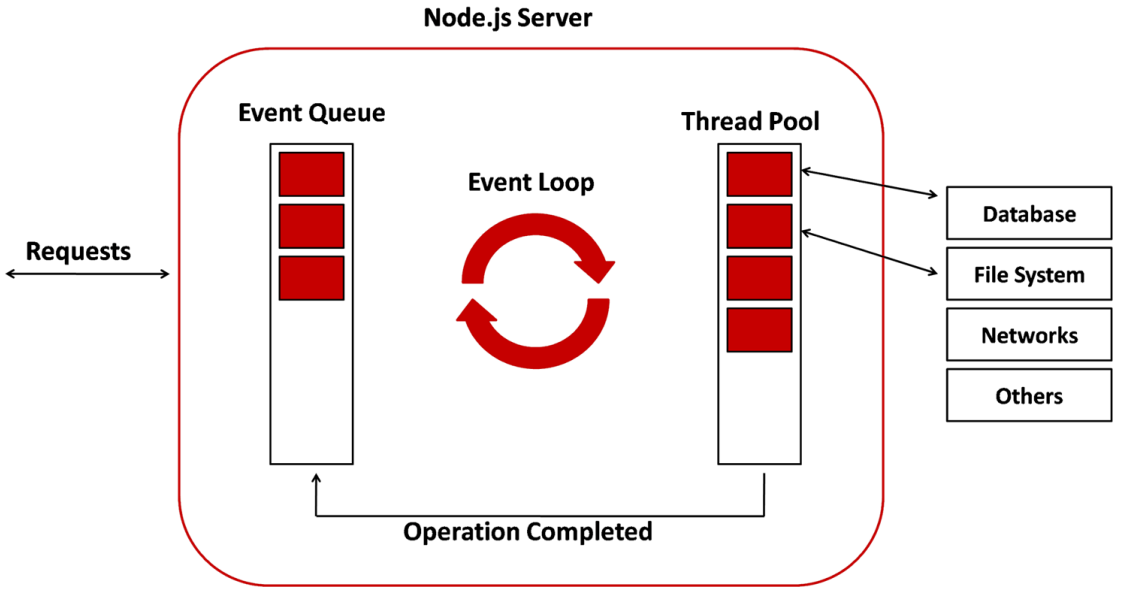


Comment fonctionne Node.js?

Event Queue

→ File d'attente d'évènements

→ Chaque requête adressée au serveur est stockée dans cette file d'attente afin d'être traitée par ordre d'arrivée

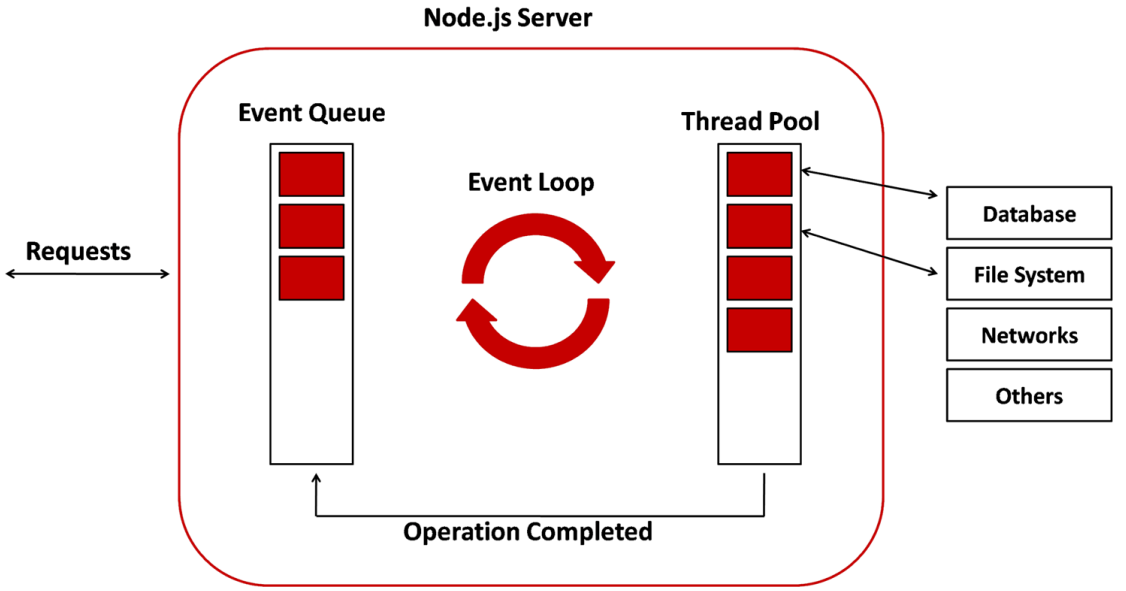


Comment fonctionne Node.js?

Event Loop

→ Peut être assimilée à un ordonnanceur

→ Communique avec l'OS afin de gérer les traitements à réaliser

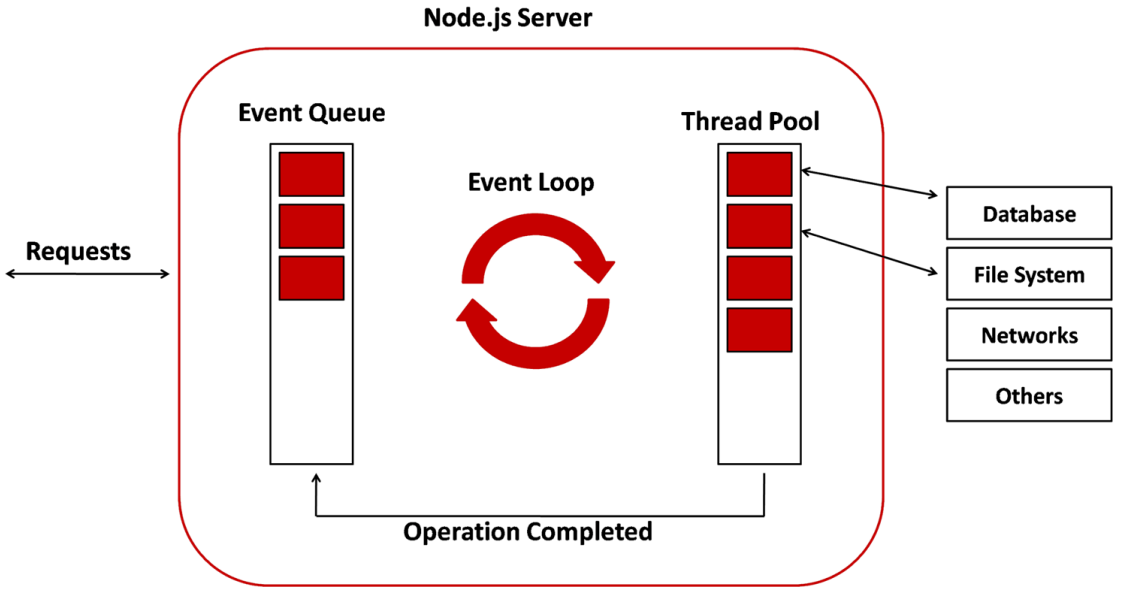


Comment fonctionne Node.js?

Thread Pool

→ Ensemble des traitements nécessitants des interactions avec une base de données, une API externe, un système de fichiers etc.

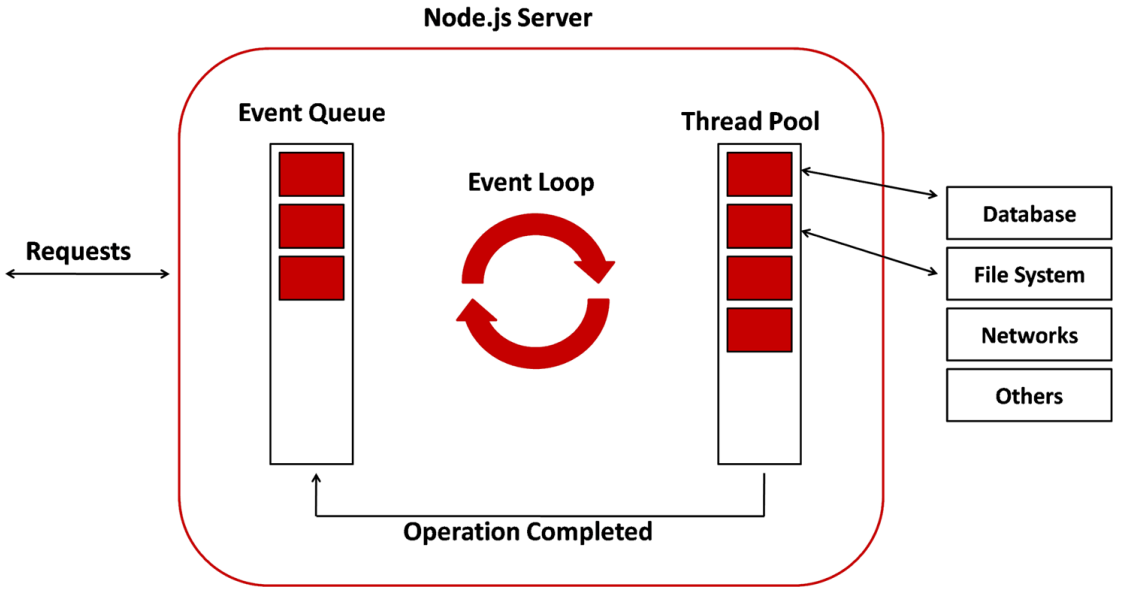
→ Les tâches sont mises en attente dans cette pile afin de libérer le thread Node.js en attendant leur exécution



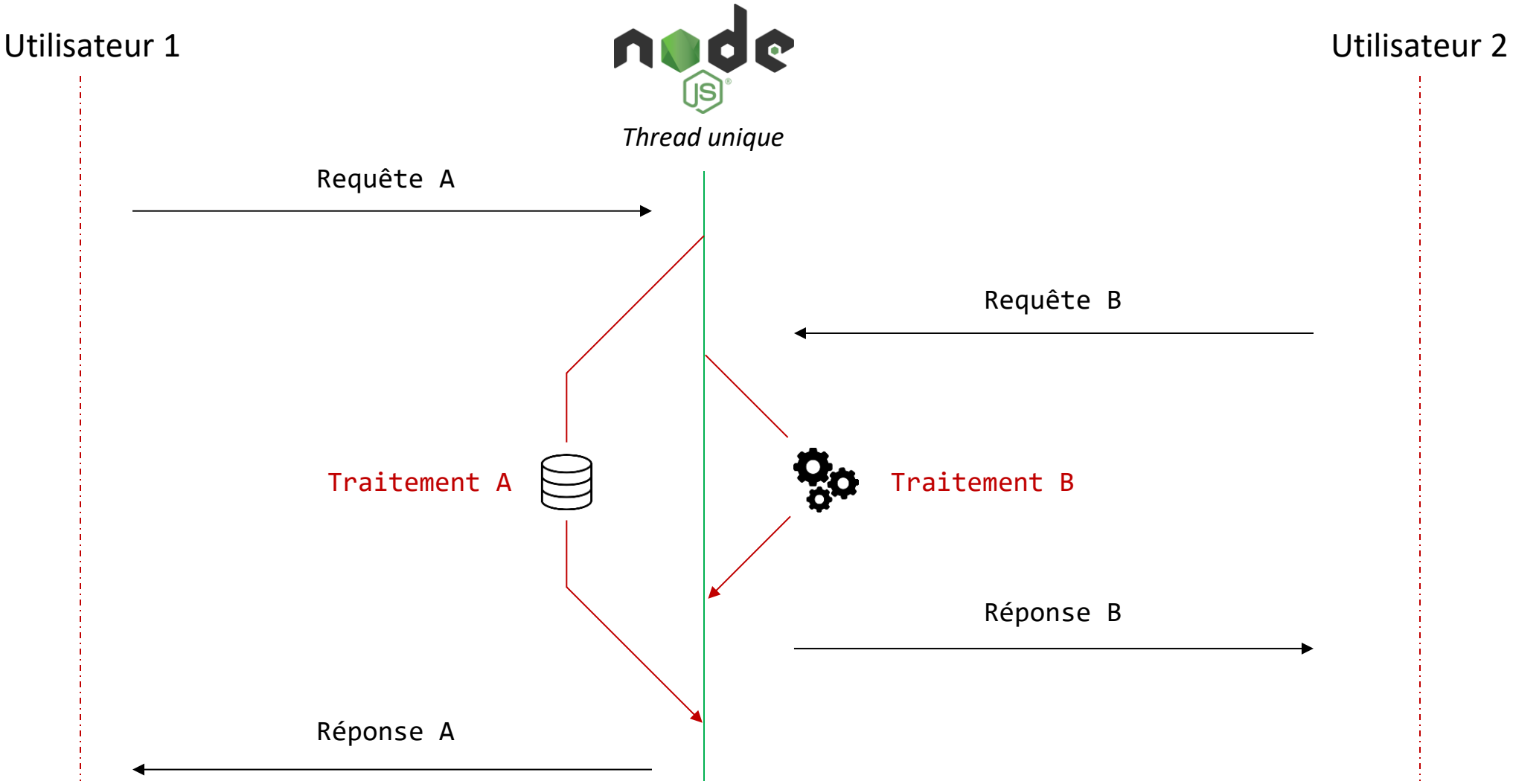
Comment fonctionne Node.js?

Operation Completed

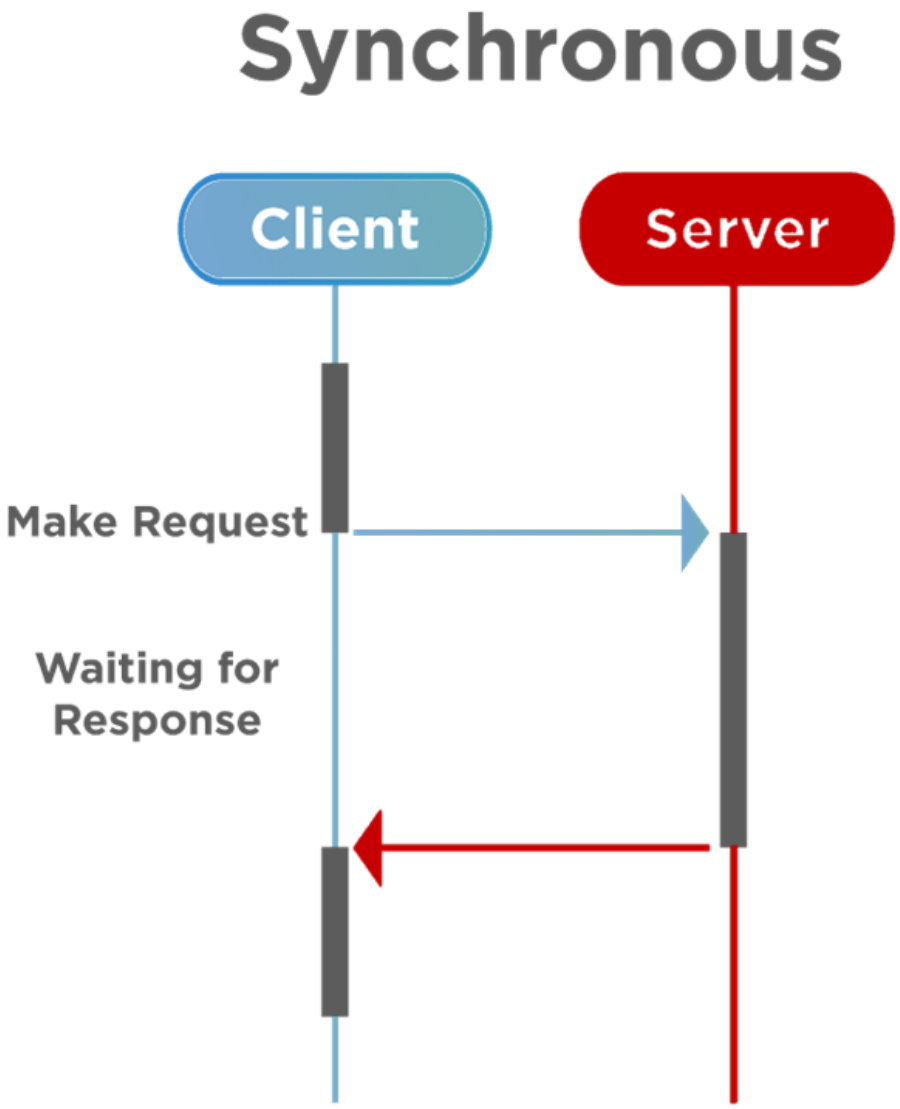
→ Lorsqu'une tâche a été traitée, la boucle d'évènement en est informée est retourne l'information

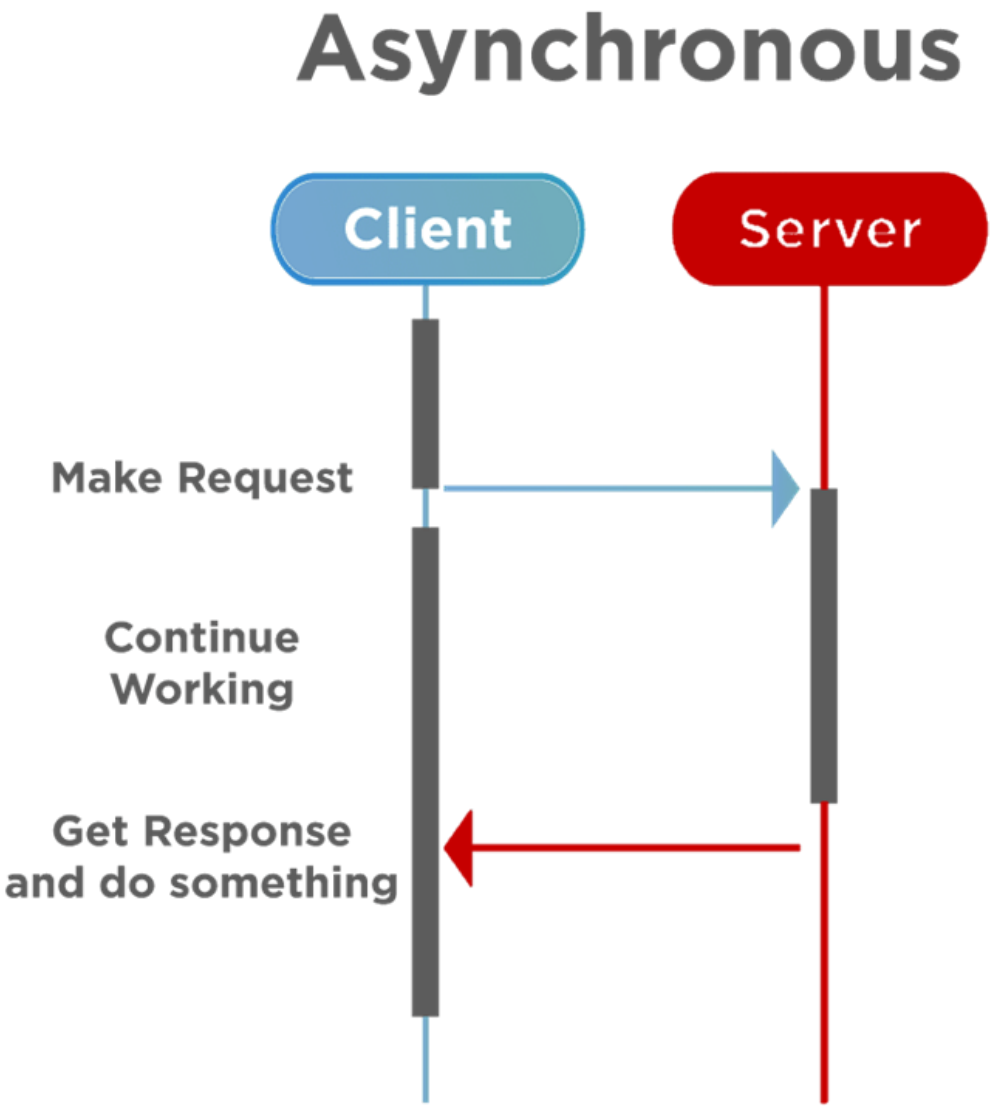


Comment fonctionne Node.js?



Synchrone *vs* Asynchrone





Qu’est-ce qu’une API ?

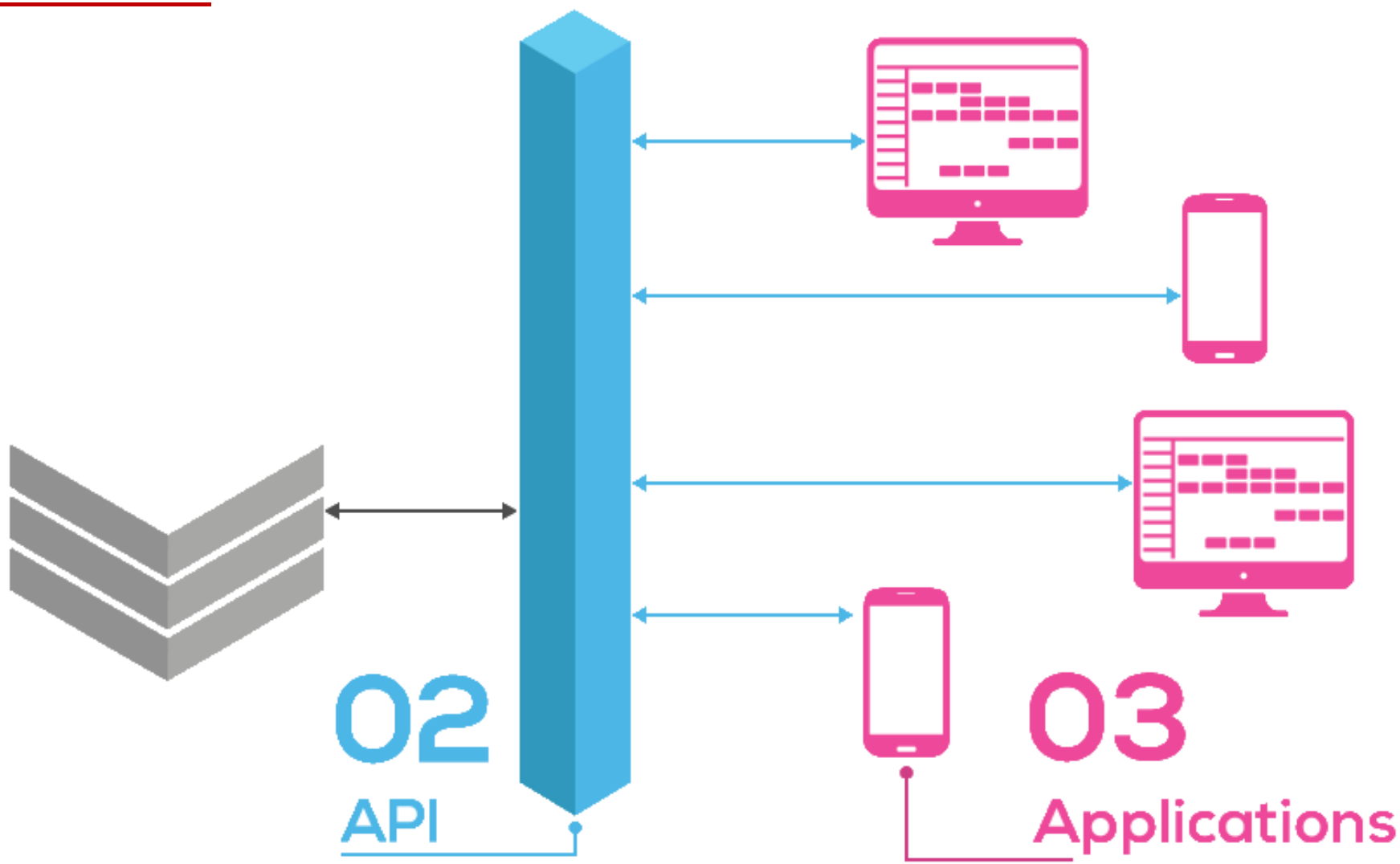
→ **A**pplication **P**rogramming **I**nterface (Interface de Programmation d’Application)

→ Interface logicielle qui permet de « connecter » un logiciel ou un service à un autre logiciel ou service afin d’échanger des données et des fonctionnalités

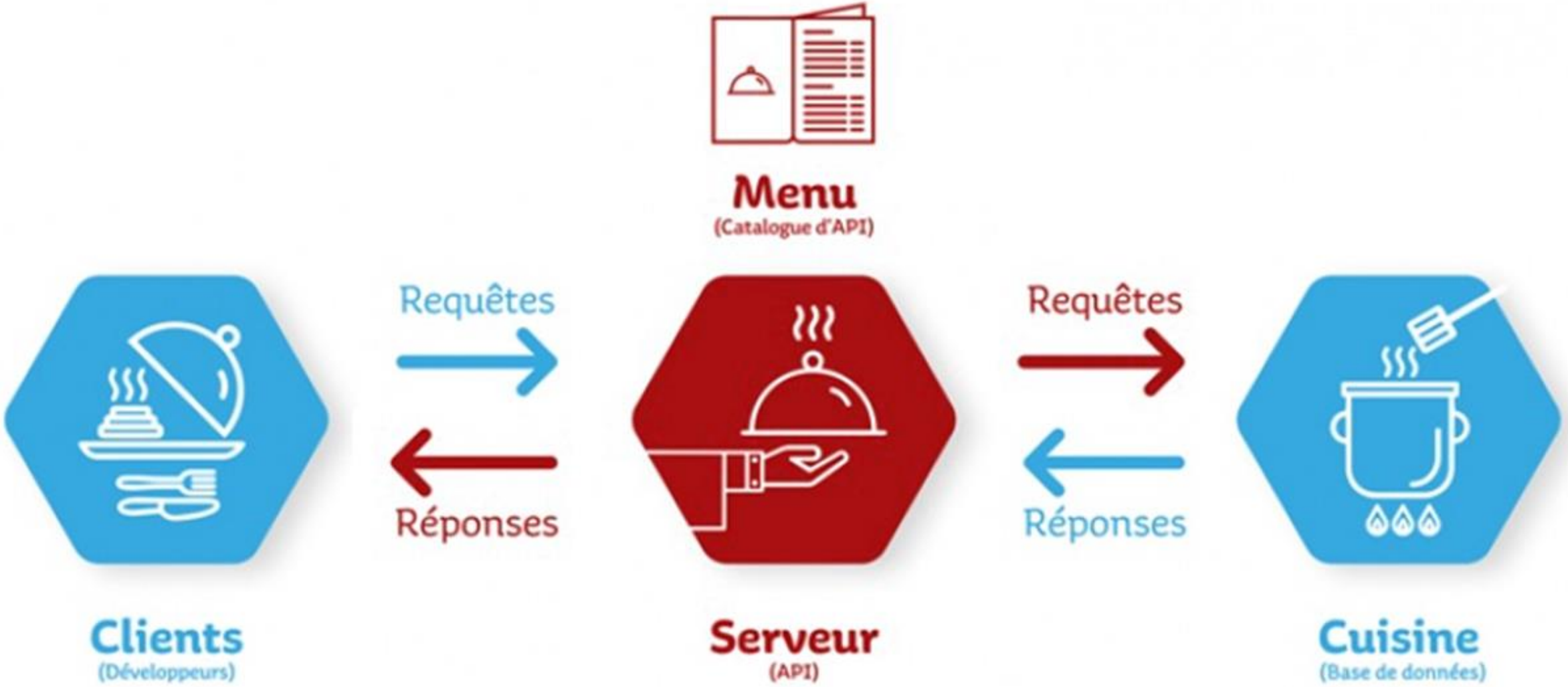
→ Contrat avec une documentation qui constitue un accord entre deux parties : si la partie 1 envoie une requête selon une structure particulière, le logiciel de la partie 2 devra répondre selon les conditions définies

→ Point d’entrée du backend

Qu’est-ce qu’une API ?



Qu’est-ce qu’une API ?



Qu’est-ce qu’une API REST?

→ Respecte les contraintes du style d'architecture REST

→ Permet d'interagir avec les services web RESTful

→ Utilise le protocole HTTP

→ Transmission de ressources

→ Format d’échange universel (JSON, XML, etc.)

Architecture REST

→ **Re**presentational **S**tate **T**ransfer (REST ou RESTful)

→ Style d'architecture permettant de construire des applications (Web, Intranet, Web Service)

→ Ensemble de conventions et de bonnes pratiques à respecter

→ Utilise les spécifications originelles du protocole HTTP

Conventions d'une API REST?

L'URI comme identifiant des ressources

→ REST se base sur les **URI** (**U**niform **R**esource **I**dentifier) afin d'identifier une ressource

→ Impératif de structurer la construction des URI (et donc URL)

→ Prendre en compte la hiérarchie des ressources et la sémantique des URL pour les éditer

Conventions d'une API REST?

L'URI comme identifiant des ressources

- Lister des livres

  NOK : http://mywebsite.com/book

  OK : http://mywebsite.com/books

Conventions d'une API REST?

L'URI comme identifiant des ressources

- Filtre et tri sur les livres

  NOK : http://mywebsite.com/books/filtre/policier/tri/asc

  OK : http://mywebsite.com/books?filtre=policier&tri=asc

Conventions d'une API REST?

L'URI comme identifiant des ressources

- Affichage d'un livre

  NOK : http://mywebsite.com/book/display/87

  OK : http://mywebsite.com/books/87

Conventions d'une API REST?

L'URI comme identifiant des ressources

- Tous les commentaires sur un livre

  NOK : http://mywebsite.com/books/comments/87

  OK : http://mywebsite.com/books/87/comments

Conventions d'une API REST?

L'URI comme identifiant des ressources

- Affichage d'un commentaire sur un livre

  NOK : http://mywebsite.com/books/comments/87/1568

  OK : http://mywebsite.com/books/87/comments/1568

Conventions d'une API REST?

L'URI comme identifiant des ressources

http://mywebsite.com/books/87/comments/1568 → un commentaire pour un livre

http://mywebsite.com/books/87/comments → tous les commentaires pour un livre

http://mywebsite.com/books/87 → un livre

http://mywebsite.com/books → tous les livres

Conventions d'une API REST?

Les réponses HTTP comme représentation des ressources

→ La réponse envoyée n'est pas une ressource, c'est la représentation d'une ressource

→ Une ressource peut avoir plusieurs représentations dans des formats divers : HTML, XML, CSV, JSON, etc.

→ C'est au client de définir quel format de réponse il souhaite recevoir via l'entête *Accept*

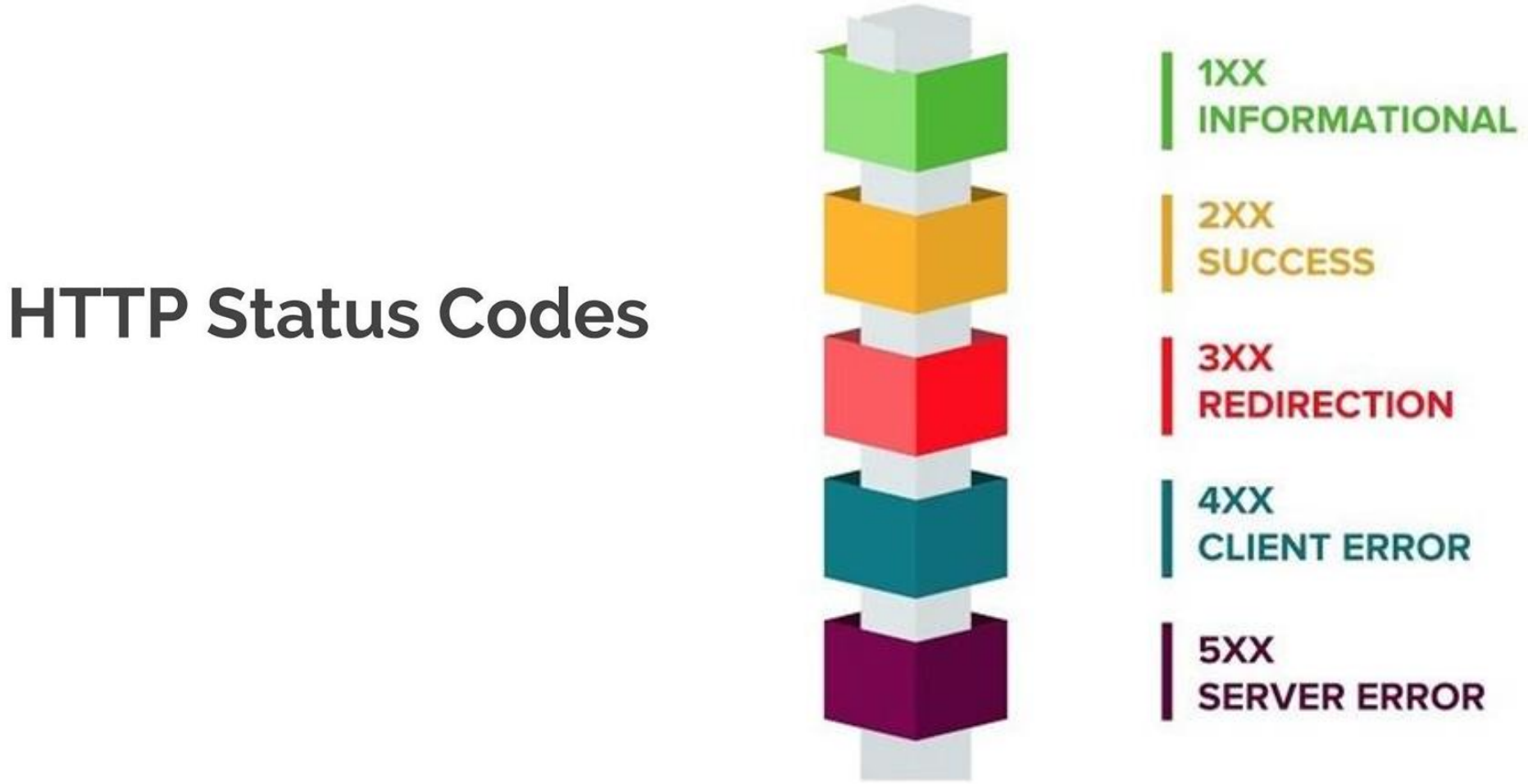
Conventions d'une API REST?

Les réponses HTTP comme représentation des ressources


```
{
    "data": [
        {
            "badges": [],
            "created_at": "2022-05-05T21:08:16.213000",
            "deleted": null,
            "id": "627420a012d18bb886e8638d",
            "last_modified": "2022-05-05T21:08:23.740000",
            "logo": "https://static.data.gouv.fr/avatars/2b/9a0e77ea294427a0e1ee1fdf7a9bb0-original.png",
            "logo_thumbnail": "https://static.data.gouv.fr/avatars/2b/9a0e77ea294427a0e1ee1fdf7a9bb0-100.png",
            "members": [
                {
                    "role": "admin",
                    "user": {
                        "avatar": null,
                        "avatar_thumbnail": null,
                        "class": "User",
                        "first_name": "Guillaume",
                        "id": "62741e21fc2803407d493126",
                        "last_name": "Desesquelles",
                        "page": "https://www.data.gouv.fr/fr/users/guillaume-desesquelles/",
                        "slug": "guillaume-desesquelles",
                        "uri": "https://www.data.gouv.fr/api/1/users/guillaume-desesquelles/"
                    }
                }
            ]
        }
    ]
}
```

Conventions d'une API REST?

Les réponses HTTP comme représentation des ressources



Conventions d'une API REST?

Les réponses HTTP comme représentation des ressources

2xx: Success

200 OK

It is the general status code. Most common code used to indicate success. The request has succeeded. The meaning of success varies depending on the HTTP method:
GET: The resource has been fetched and is transmitted in the message body.
HEAD: The entity headers are in the message body.
PUT or POST: The resource describing the result of the action is transmitted in the message body.
TRACE: The message body contains the request message as received by the server

201 Created

Successful creation occurred (via either POST or PUT). Set the Location header to contain a link to the newly-created resource (on POST). Response body content may or may not be present.

204 No Content

Status when wrapped responses are not used and nothing is in the body (e.g. DELETE).

3xx: Redirection

304 Not Modified

This is used for caching purposes. It tells the client that the response has not been modified, so the client can continue to use the same cached version of the response.

Conventions d'une API REST?

Les réponses HTTP comme représentation des ressources

4xx: Client Error

400 Bad Request

General error when fulfilling the request would cause an invalid state. Domain validation errors, missing data, etc. are some examples.

401 Unauthorized

The error code response for missing or invalid authentication token.

403 Forbidden

The error code for a user not authorized to perform the operation or the resource is unavailable for some reason (e.g. time constraints, etc.).

404 Not Found

Used when the requested resource is not found, whether it doesn't exist or if there was a 401 or 403 that, for security reasons, the service wants to mask.

5xx: Server Error

500 Internal Server Error

The general catch-all error when the server-side throws an exception.

Conventions d'une API REST?

Les liens comme relation entre ressources

→ Les liens indiquent la présence d'une relation entre plusieurs ressources

→ Peut être utilisé dans l'attribut *rel* sur les liens

- previous
- next
- last

Conventions d'une API REST?

Un paramètre comme jeton d'authentification

→ Authentifier une requête

→ Authentication Token

→ Chaque requête est envoyée avec un jeton passé en paramètre

→ Ce jeton temporaire est obtenu en envoyant une première requête d'authentification puis en le combinant avec nos requêtes.

Conventions d'une API REST?

Les verbes HTTP comme identifiant des opérations

→ Utiliser les verbes HTTP existants plutôt que d'inclure l'opération dans l'URI

→ Généralement pour une ressource, il y a 4 opérations possibles :

CREATE	→	Créer
READ	→	Afficher
UPDATE	→	Mettre à jour
DELETE	→	Supprimer

Qu’est-ce que le C.R.U.D. ?

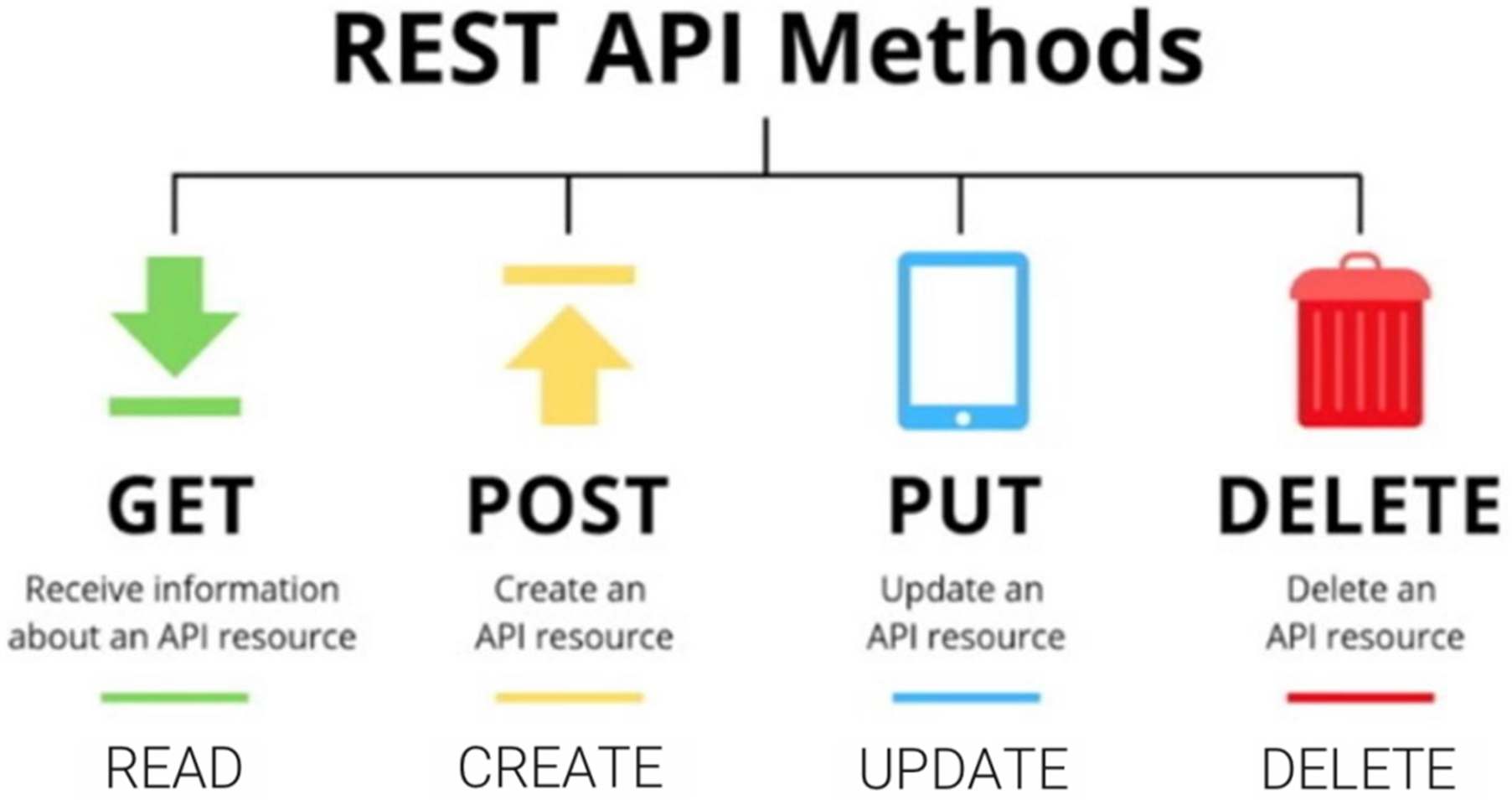
→ Acronyme des noms des quatre opérations de base de la gestion de la persistance des données et applications :

Qu’est-ce que le C.R.U.D. ?

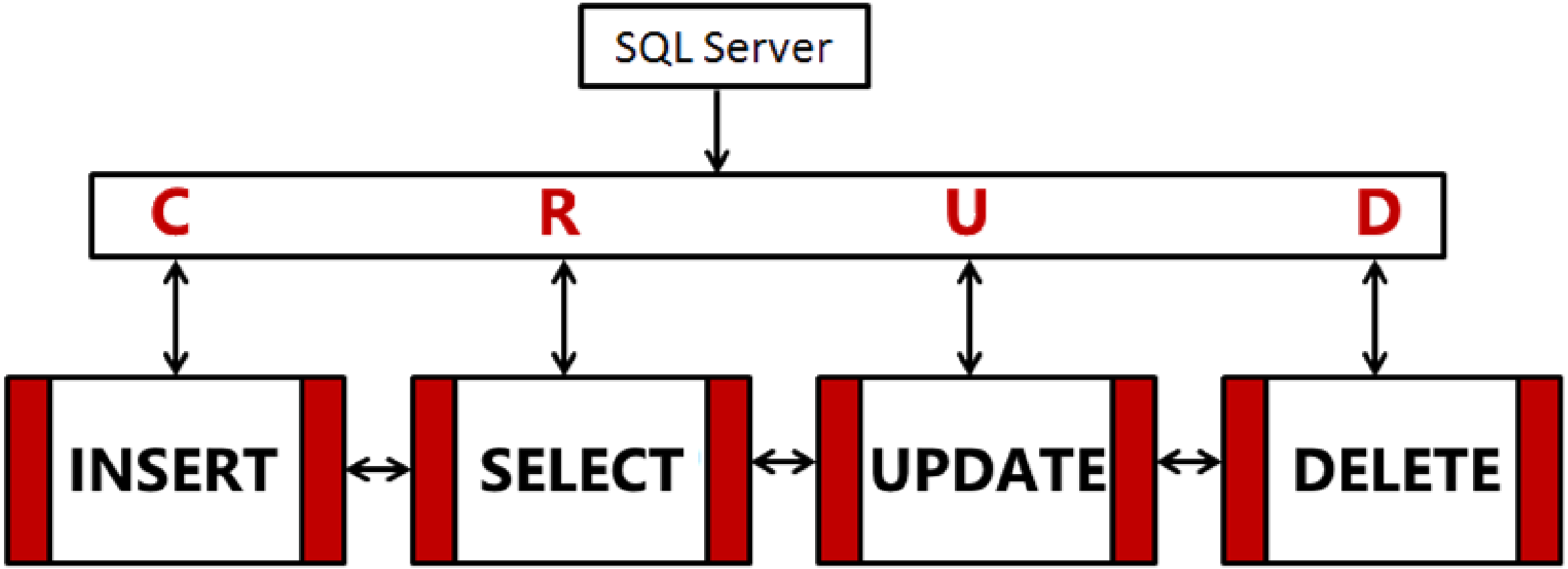
→ Résume les fonctions qu’un utilisateur a besoin d’utiliser pour créer et gérer des données

→ Facilitent le développement et l’utilisation des applications en optimisant l’accès au système de base de données exploité

Qu’est-ce que le C.R.U.D. ?



Qu’est-ce que le C.R.U.D. ?



PATCH

→ La méthode PATCH d'une requête HTTP applique des modifications partielles à une ressource.

```
{
        "id": 1,
        "first_name" : "Tom",
        "last_name": "Smith",
        "email": "tom.smith@example.com"
        "phone": {
                "home": "0123456789",
                "mobile": "9876543210"
        },
        "address": {
                "street": "34 avenue de l'opera",
                "zip_code": "75002",
                "city": "PARIS"
        }
}
```

```
PATCH /users/1
Content-Type: application/merge-patch+json

{
        "address": {
                "street": "50 avenue des Champs Elysées",
                "zip_code": "75008"
        }
}
```

Qu’est-ce que Express.js

→ **Express.js** est le framework le plus populaire pour Node.js

→ Permet d’écrire des fonctions de traitement pour différentes requêtes HTTP répondant à différentes URI (par le biais des routes).

→ Peut être vu comme un routeur : permet de déclarer les routes de l’application

→ Permet d’ajouter des requêtes de traitement « middleware »

Comment l'utiliser ?

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.send('Hello World!')
});

app.listen(port, () => {
  console.log(`Application exemple à l'écoute sur le port
${port}!`)
});
```

Import du module Express.js

Création de l'app Express.js

Définition d'une route **/**

Démarrage du serveur

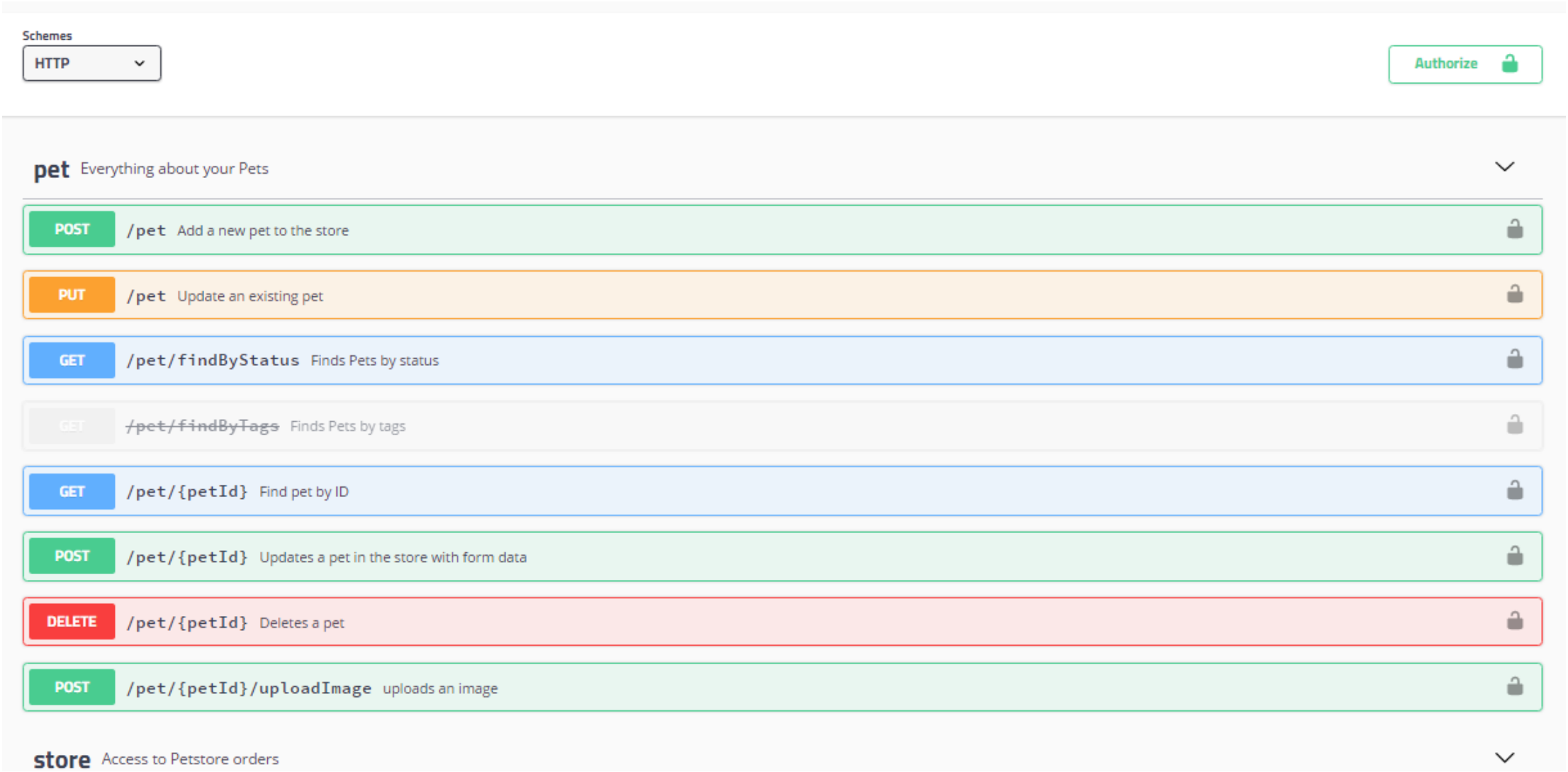
Qu’est-ce que OpenAPI V3

→ **OpenAPI** permet de décrire une API avec une méthode standardisée

→ Permet d’écrire des fonctions de traitement pour différentes requêtes HTTP répondant à différentes URI (par le biais des routes).

→ Permet de consulter la documentation et de tester les API via SwaggerUI

SwaggerUI


Schemes
HTTP
Authorize
pet Everything about your Pets
POST /pet Add a new pet to the store
PUT /pet Update an existing pet
GET /pet/findByStatus Finds Pets by status
GET /pet/findByTags Finds Pets by tags
GET /pet/{petId} Find pet by ID
POST /pet/{petId} Updates a pet in the store with form data
DELETE /pet/{petId} Deletes a pet
POST /pet/{petId}/uploadImage uploads an image
store Access to Petstore orders

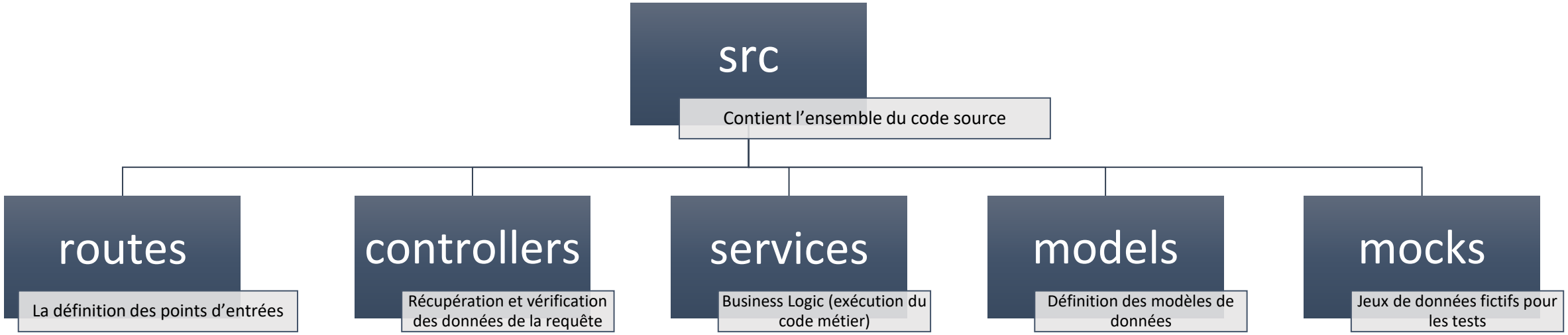
Comment l'utiliser ?

OpenAPI Specification

Documentation Swagger

OpenAPI 3.0: How to Design and Document APIs with the Latest OpenAPI Specification 3.0

Structure du backend



03

Base de données

mongoDB

mongoose

Sommaire

SQL

→ Relationnel

→ Requête pour analyser et récupérer données

→ Stockage sous forme de tableaux

→ Particulièrement adapté pour les requêtes complexes et intensives

→ Stockage optimisé et stabilité

→ Inconvénient : rigidité

NoSQL

→ Non relationnelle (ou distribué)

→ Adapté pour une variété de technologie d'application moderne (ex: WebApp)

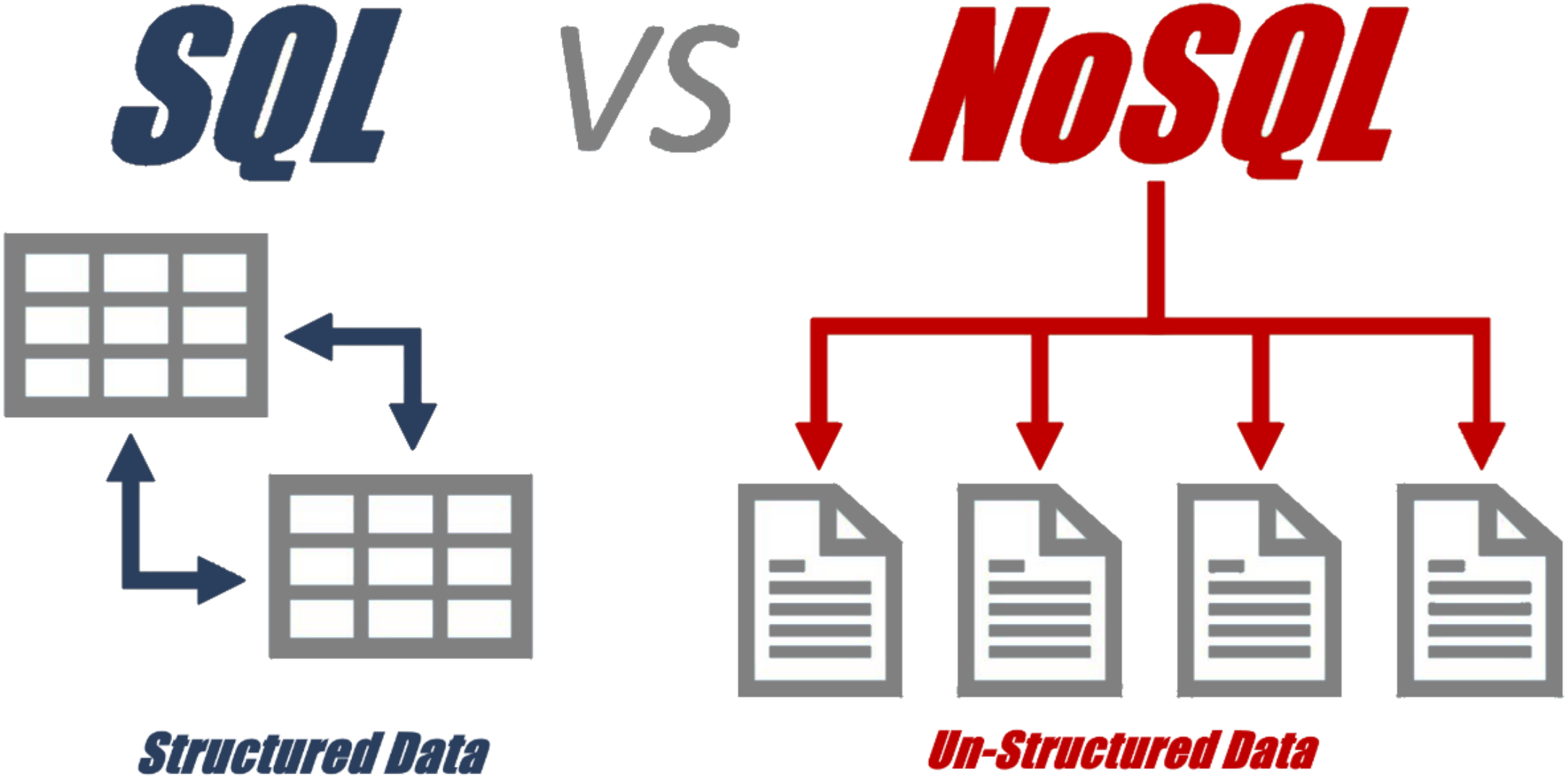
→ Stockage sous forme de documents / graphes / ensemble clés-valeurs

→ Particulièrement adapté pour les grandes bases de données et le Big Data

→ Facilité et flexibilité du stockage

→ Inconvénient : arfois trop permissif

Les principales différences



Structure du stockage

→ **SQL** organise le stockage de données sur le principe de tables reliées entre elles, avec une structure et des types de données qui sont rigides

→ **NoSQL** stocke et manipule des documents qui correspondent à des collections d'objets

→ Les tables SQL imposent un modèle de données strictes, donc il est difficile de faire des erreurs

→ NoSQL est plus flexible et pardonnable, mais la possibilité de stocker des données n'importe où peut entraîner des problèmes de cohérence

Organisation des données

→ En **SQL**, il est impossible d'ajouter des données sans avoir défini des tables et des types de champs dans ce que l'on appelle un schéma.

→ Ce schéma SQL contient aussi d'autres informations : clés primaires – index, contraintes, fonction, procédures stockées

→ Avec **NoSQL**, les données peuvent être ajoutées n'importe où, à tout moment, sans qu'il soit nécessaire de spécifier une conception de document ou même une collection à l'avance

→ La représentation des données en collection et leur résultat en flux JSON permet de consommer les données très rapidement et facilement

Normalisation ou non des données

→ En **SQL**, on parle de normalisation des données.

→ Chaque données est unique dans une base de données SQL

→ Avec **NoSQL**, on parle de dénormalisation des données

→ Les données ne sont pas reliées entres-elles par des clés primaires ou étrangère et peuvent donc être dupliquées dans différents documents

Les jointures

→ En **SQL**, la clause JOIN est utilisée pour mettre en relation des données de différentes tables

→ Avec **NoSQL**, il n'y a pas d'équivalent car les données ne sont pas sensées être stockées sous forme de relations

Intégrité des données

→ En **SQL**, il est possible d'appliquer des règles d'intégrité de données à l'aide de contraintes de clés étrangères

→ Le schéma SQL applique ces règles pour prévenir la création de données invalides ou d'enregistrements orphelins.

→ Par exemple, cela empêche de supprimer une donnée si un ou plusieurs enregistrements lui sont toujours attribués.

→ Avec **NoSQL**, ces mêmes options d'intégrité de données ne sont pas disponibles

Scalabilité

→ Le « load balancing » sur un serveur **SQL** est complexe car dû à la nature même dont les données sont stockées, organisées et reliées entre elles

→ Les modèles de données **NoSQL** peuvent rendre le processus plus facile

→ L'organisation des données en documents et la « denormalization » des collections permettent le partitionnement

→ Autorise une montée en charge de la base de données sur le matériel courant déployé sur site ou dans le Cloud

Qu’est-ce que MongoDB?

« MongoDB est un système de gestion de base de données orienté documents, répartissable sur un nombre quelconque d'ordinateurs et ne nécessitant pas de schéma prédéfini des données. »

✓ SGBD NoSQL

✓ Ne nécessitant pas de schéma prédéfini des données

✓ Un des SGBD les plus utilisés

Lexique

MySQL	MongoDB
Base de données	Base de données
Table	Collection
Enregistrement	Document
Colonne	Champs
Clé primaire	ObjectId
Clé étrangère	Référence
Jointure	Document embarqué
Index	Index
ORM (**O**bject **R**elationnal **M**apping)	ODM (**O**bject **D**ocument **M**apping)

Qu’est-ce que Mongoose

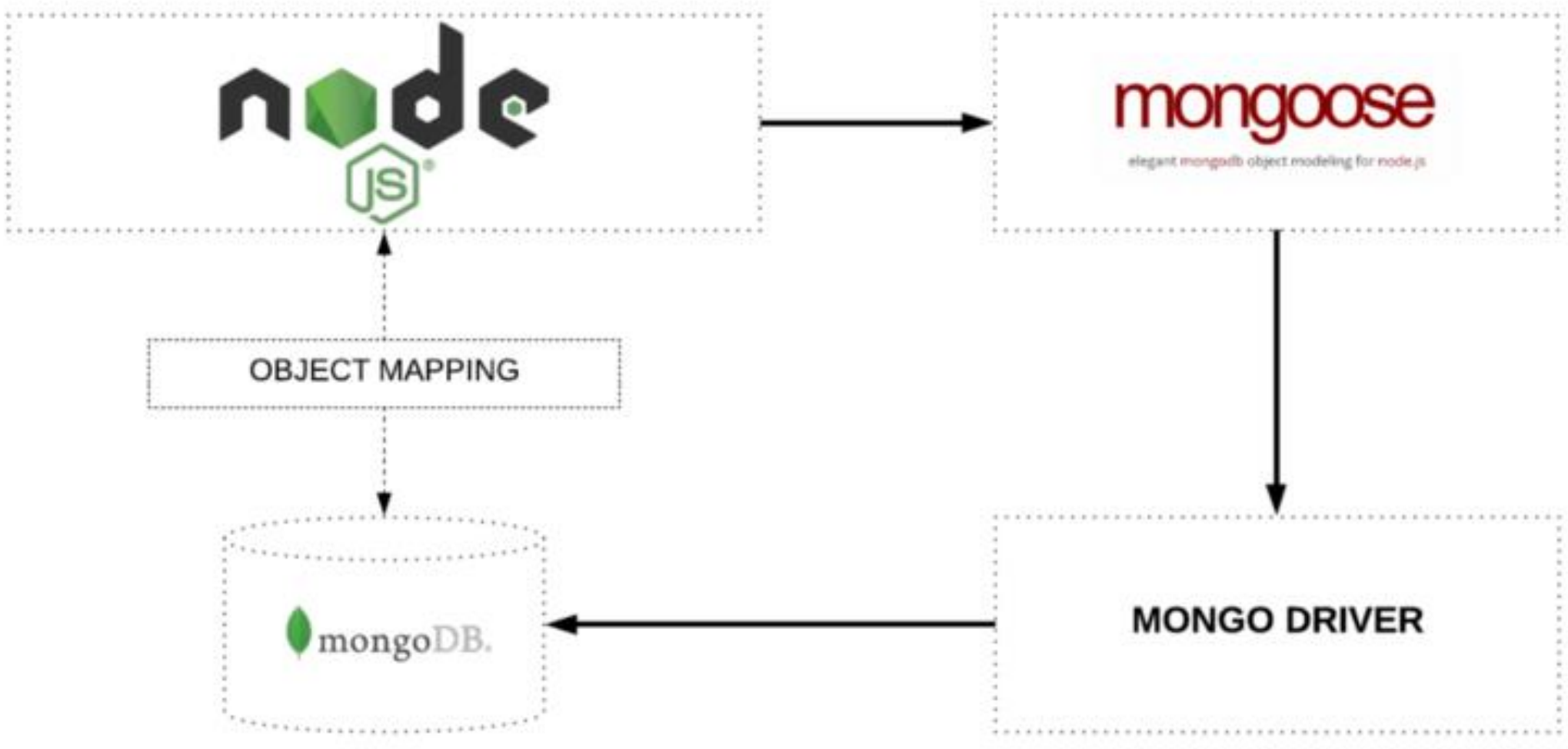
« Mongoose est une bibliothèque de programmation JavaScript orientée objet qui crée une connexion entre MongoDB et le framework d'application Web Express. »

✓ ODM (Object Document Mapper)

✓ Permet d’assurer le lien entre le backend et la base de données

✓ Embarque un driver MongoDB

Qu’est-ce que Mongoose



Utilisation de Atlas

Atlas

Créer une collection

db.createCollection(name, options)

Insérer des données

db.collection.insertOne()

db.collection.insertMany()

Lire des données

Sélectionner toutes les données d'une collection

```
db.collection.find( {} )
```

Sélectionner selon un critère de recherche

```
db.collection.find( { status: "D" } )
```

Documentation

Lire des données

Sélectionner selon plusieurs valeurs pour un critère

```
db.collection.find( { status: { $in: [ "A", "D" ] } } )
```

Sélectionner selon plusieurs critères cumulatifs

```
db.collection.find( { status: "A", qty: { $lt: 30 } } )
```

Documentation

Lire des données

Sélectionner selon plusieurs critères non cumulatifs

```
db.collection.find( { $or: [ { status: "A" }, { qty: { $lt: 30
} } ] } )
```

Sélectionner selon plusieurs critères cumulatifs et non cumulatifs

```
db.collection.find( {
     status : "A",
     $or : [ {qty : { $lt : 30 } }, { item : /^p/ } ]
} )
```

Documentation

Mettre à jour des données

db.collection.updateOne(filter, update, options)

db.collection.updateMany(filter, update, options)

Supprimer des données

db.collection.deleteOne()

db.collection.deleteMany()

Qu’est-ce qu’une agrégation ?

→ Les opérations d'agrégation traitent plusieurs documents et renvoient des résultats calculés

→ Utilisées par exemple pour :

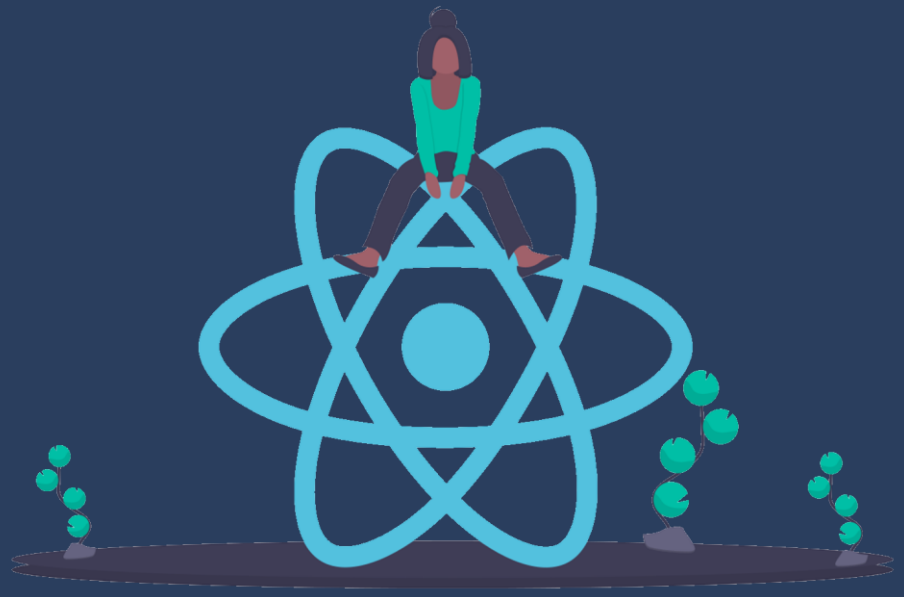
- Regrouper les valeurs de plusieurs documents ensemble
- Effectuer des opérations sur les données groupées pour renvoyer un seul résultat
- Analyser les changements de données au fil du temps

Exemple

```
db.orders.aggregate( [
   // Stage 1: Filter pizza order documents by pizza size
   {
      $match: { size: "medium" }
   },
   // Stage 2: Group remaining documents by pizza name and calculate total
quantity
   {
      $group: { _id: "$name", totalQuantity: { $sum: "$quantity" } }
   }
] )
```

04

Frontend with React



Sommaire

Qu’est-ce que React ?

« Une bibliothèque JavaScript pour créer des interfaces utilisateurs. »

✓ Déclaratif

✓ À base de composants

✓ Utilisable partout

Qu’est-ce que React ?

Déclaratif

→ Permet de créer des interfaces utilisateur de façon déclarative.

→ Plus besoin de toucher au DOM pour créer des interfaces WEB ou pour la gestion d’évènements.

→ Permet d’indiquer au navigateur ce dont on a besoin sans avoir à lui dire ce qu’il doit faire (ex: JQuery avec la recherche d’éléments du DOM et leur modification à la volée).

Qu’est-ce que React ?

À base de composants

→ Permet de créer des composants autonomes qui maintiennent leur propre état

→ Possibilité d’assembler les composants pour créer une interface graphique

→ Données complexes facilement utilisables dans les composants

→ Syntaxe à mi-chemin entre XML et HTML : JSX (JavaScript Syntax Extension)

Qu’est-ce que React ?

Utilisable partout

→ Indépendant de toute autre technologie déjà en place

→ Pas de nécessité de réécrire le code existant

Comment fonctionne React ?

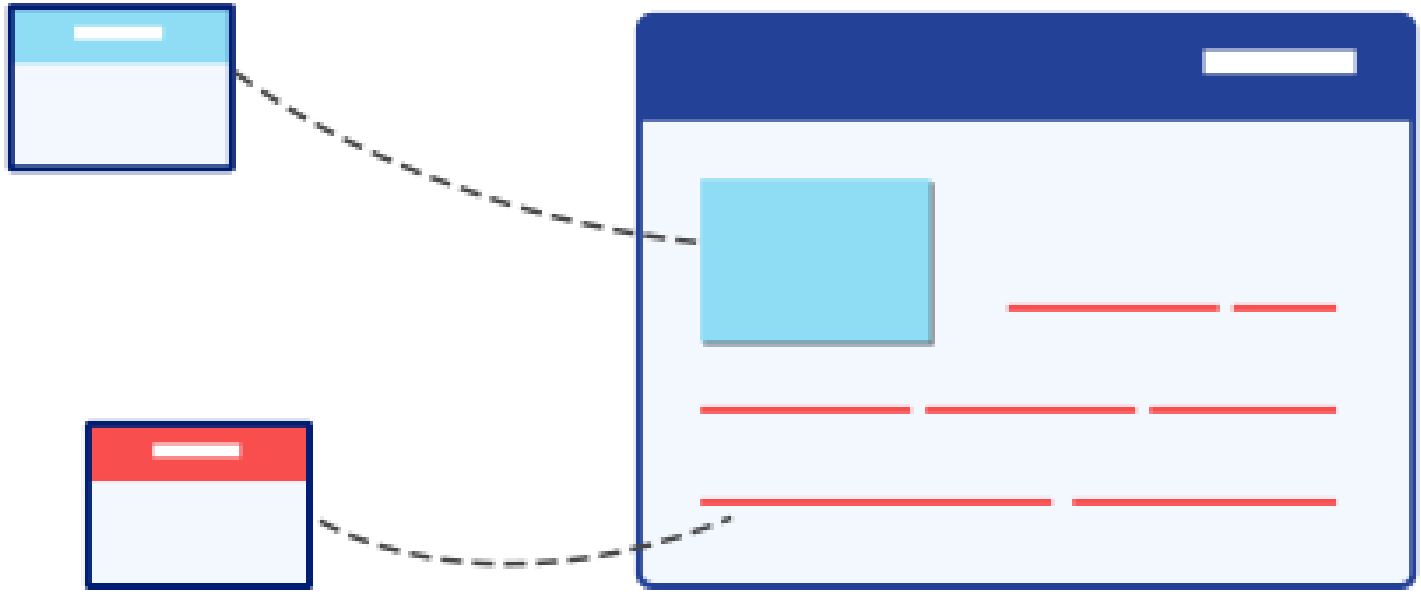
Décomposer son interface utilisateur

→ Simplicité

→ Imbrication

→ Réutilisabilité

→ Évolutivité



Comment fonctionne React ?

Utilisation d'un Virtual-DOM

→ DOM = **D**ocument **O**bject **M**odel

Interface de programmation qui permet une représentation structurée du document sous forme d'un arbre.

Une modification d'un élément implique la mise à jour complète du DOM.

Comment fonctionne React ?

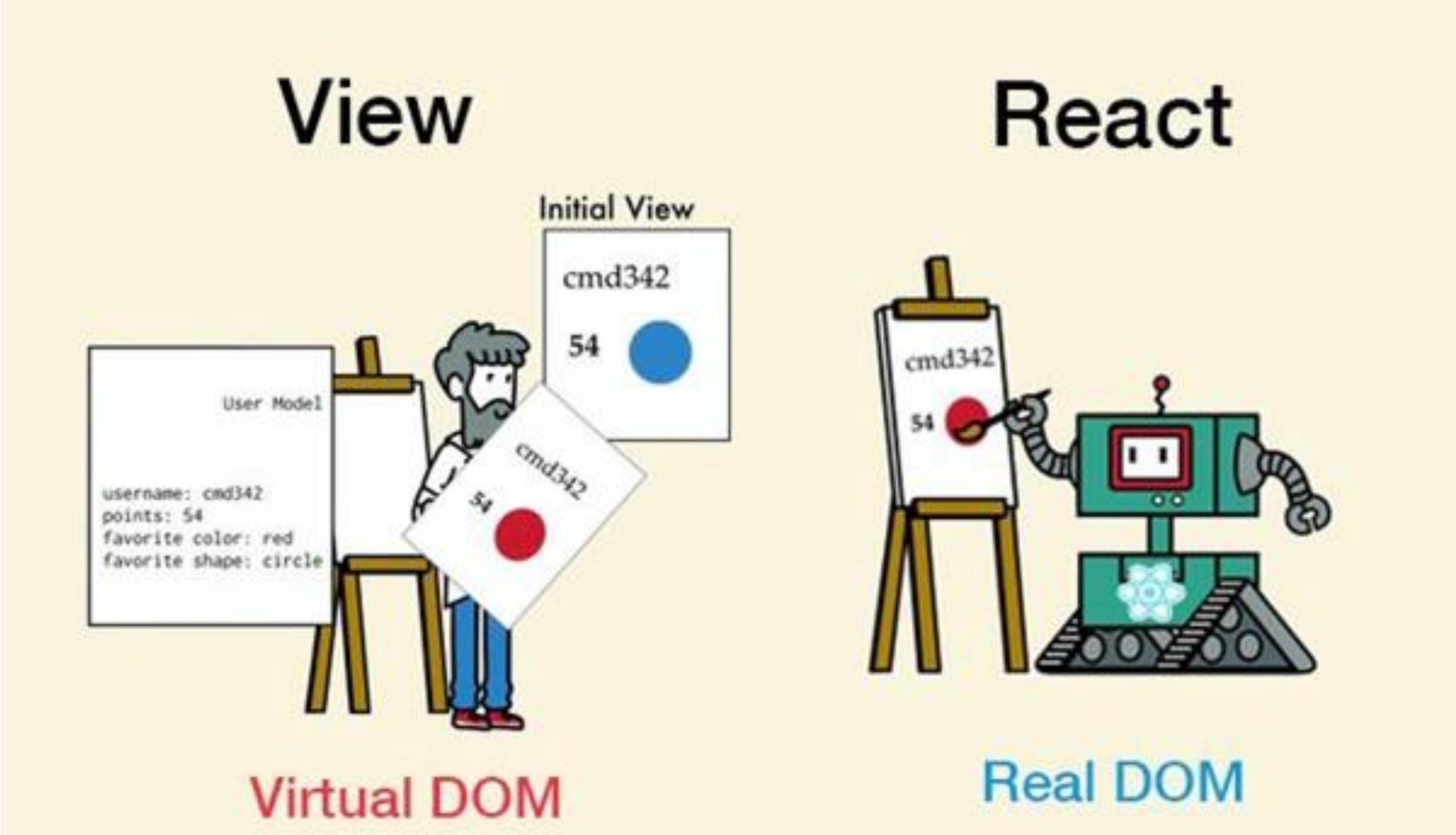
Utilisation d'un Virtual-DOM

Dès qu'un composant React évolue (données qui changent, style modifié…), l'interface doit être redessinée.

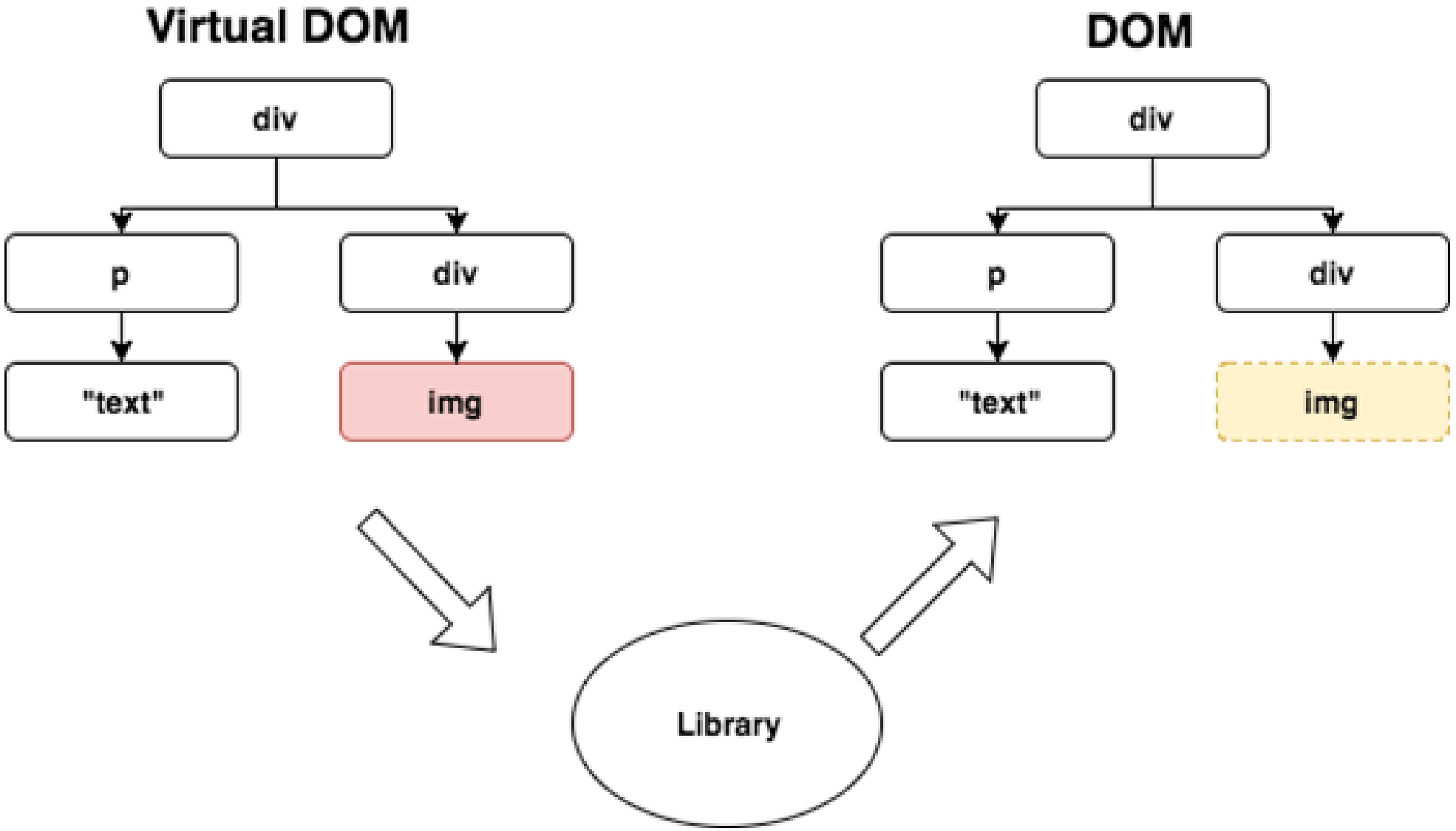
React utilise un Virtual DOM afin d'appliquer les modifications puis le compare avec le DOM du navigateur.

Seules les différences sont mises à jour dans le DOM du navigateur ce qui apporte davantage de performance.

Comment fonctionne React ?



Comment fonctionne React ?

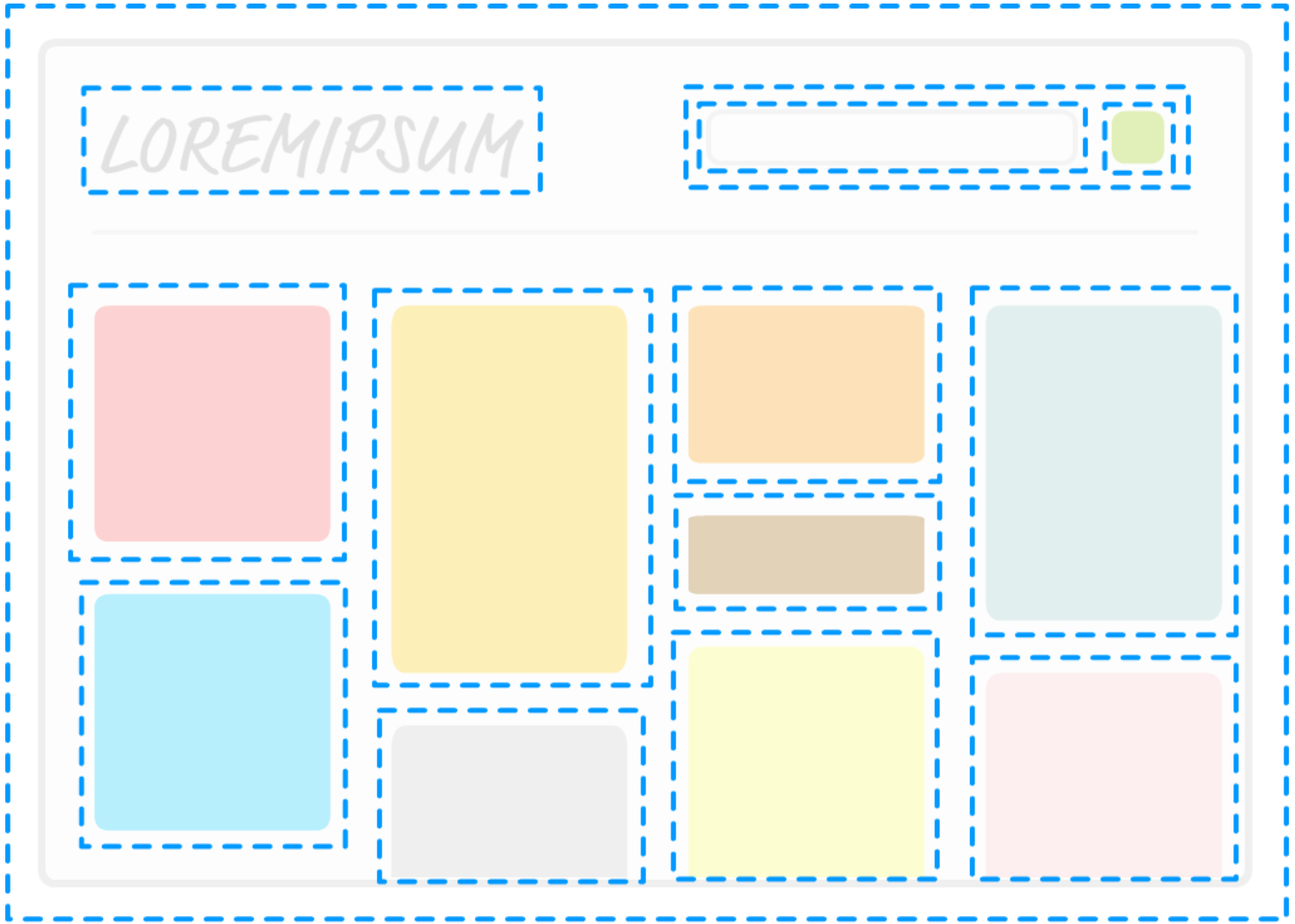


Utilisation des composants

L'interface utilisateur segmentée en plusieurs composants

1 composant est généralement composé :

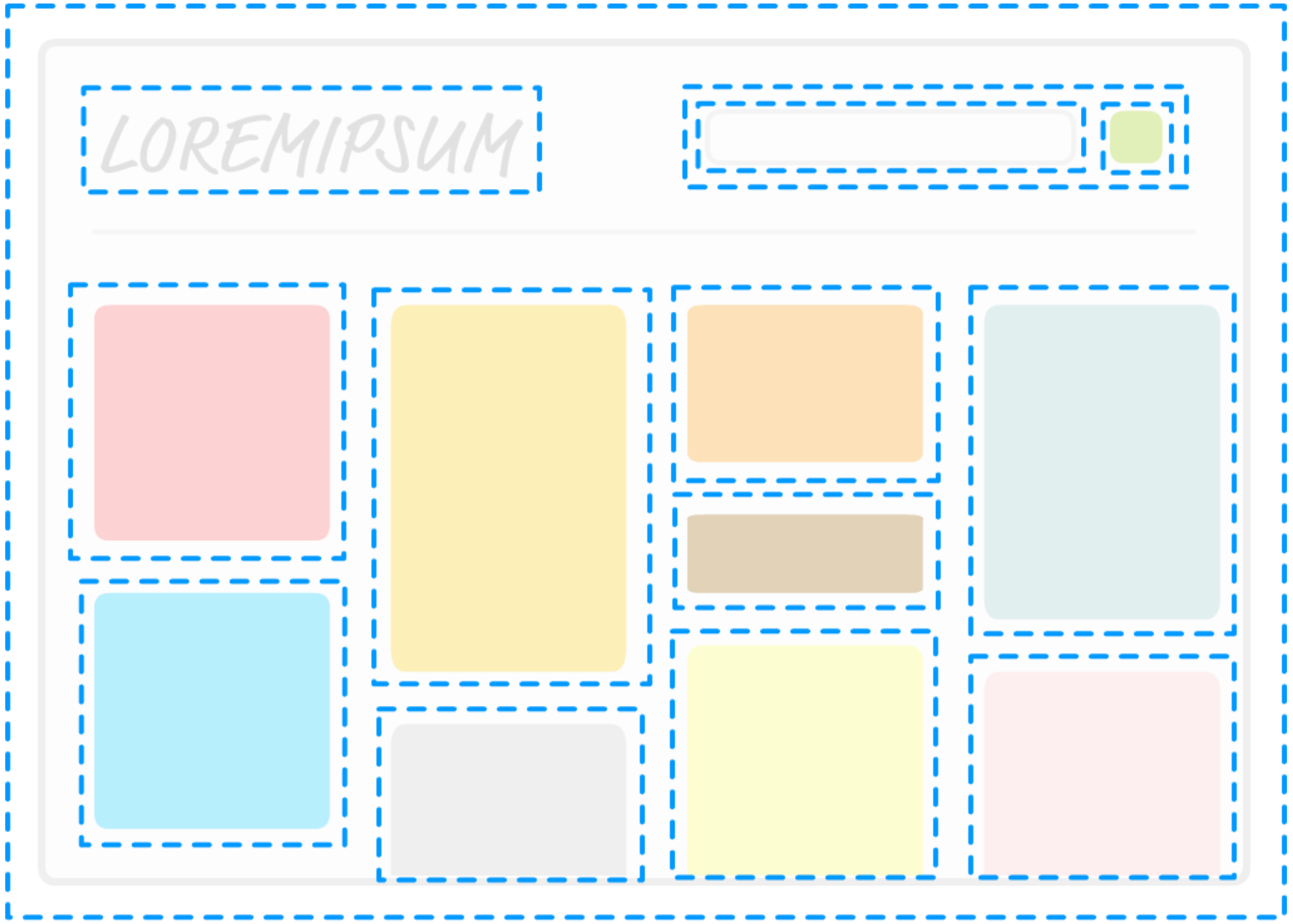
- d'un fichier de structure
- d'un fichier de style
- d'un fichier de tests



Utilisation des composants

Les composants peuvent ensuite être imbriqués entre eux.

Ils sont donc réutilisables sans avoir à réécrire le code.



Utilisation des composants

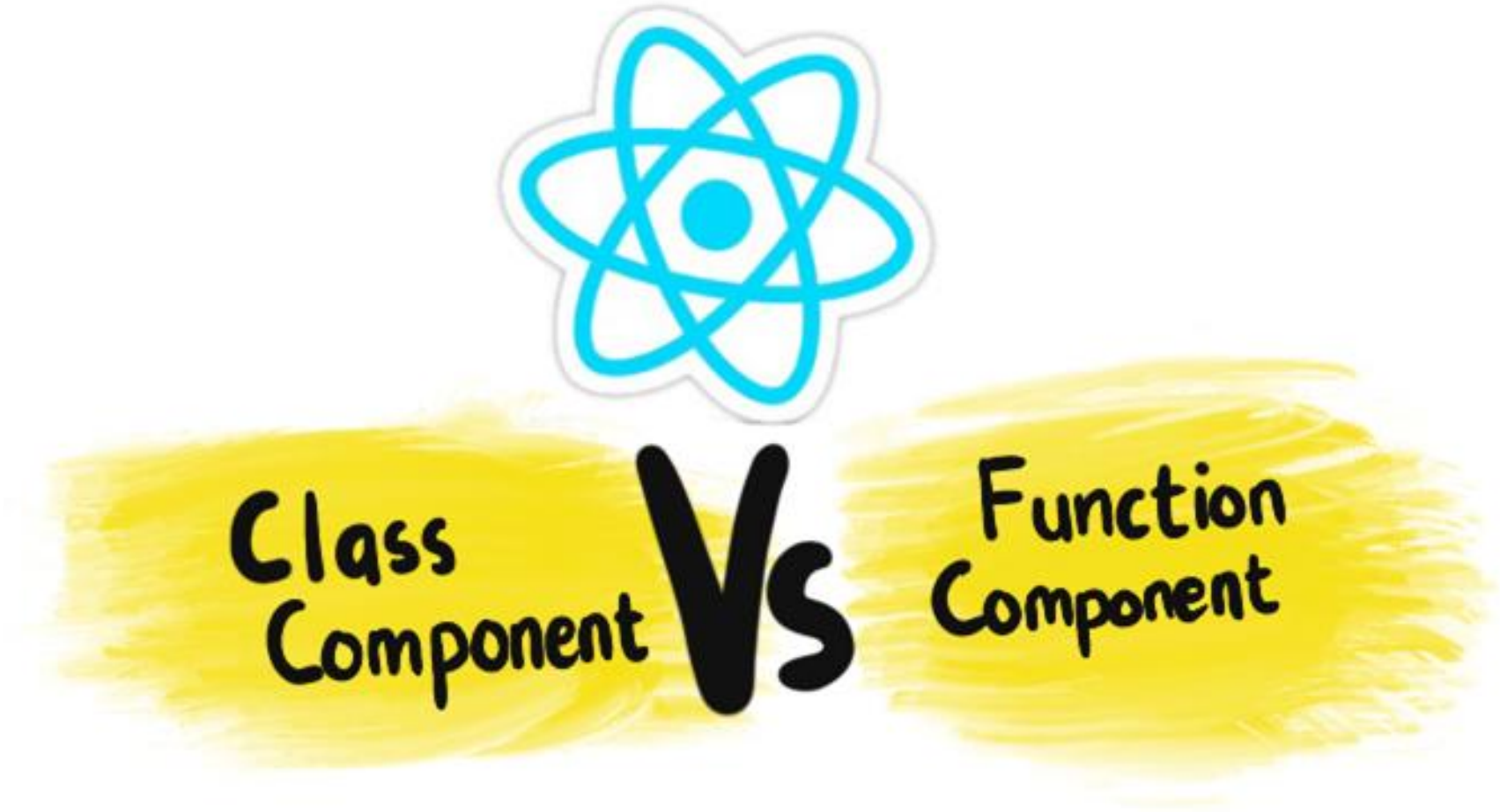
Création d'un composant

```
// On crée le composant
function MonComposant() {
  return (
    <div>Ceci est mon composant!</div>
  )
}
```

Utilisation du composant

```
// On peut désormais utiliser notre composant dans un autre composant
const Container = () => {
  return (
    <div className="container">
      <MonComposant />
    </div>
  )
}
```

Utilisation des composants



Une syntaxe différente

```jsx
import React, { useState } from "react";

const FunctionalComponent=()=>{
    const [count, setCount] = useState(0);

    const increase = () => {
        setCount(count+1);
    }

    return (
        <div style={{margin:'50px'}}>
            <h1>Welcome to Geeks for Geeks </h1>
            <h3>Counter App using Functional Component : </h3>
          <h2>{count}</h2>
            <button onClick={increase}>Add</button>
        </div>
    )
}

export default FunctionalComponent;
```

```jsx
import React, { Component } from "react";

class ClassComponent extends React.Component{
    constructor(){
        super();
        this.state={
            count :0
        };
        this.increase=this.increase.bind(this);
    }

    increase(){
        this.setState({count : this.state.count +1});
    }

    render(){
        return (
            <div style={{margin:'50px'}}>
                <h1>Welcome to Geeks for Geeks </h1>
                <h3>Counter App using Class Component : </h3>
                <h2> {this.state.count}</h2>
                <button onClick={this.increase}> Add</button>

            </div>
        )
    }
}

export default ClassComponent;
```

Class Component

→ Repose sur l'instanciation d'une classe JavaScript qui étend **`React.Component`**

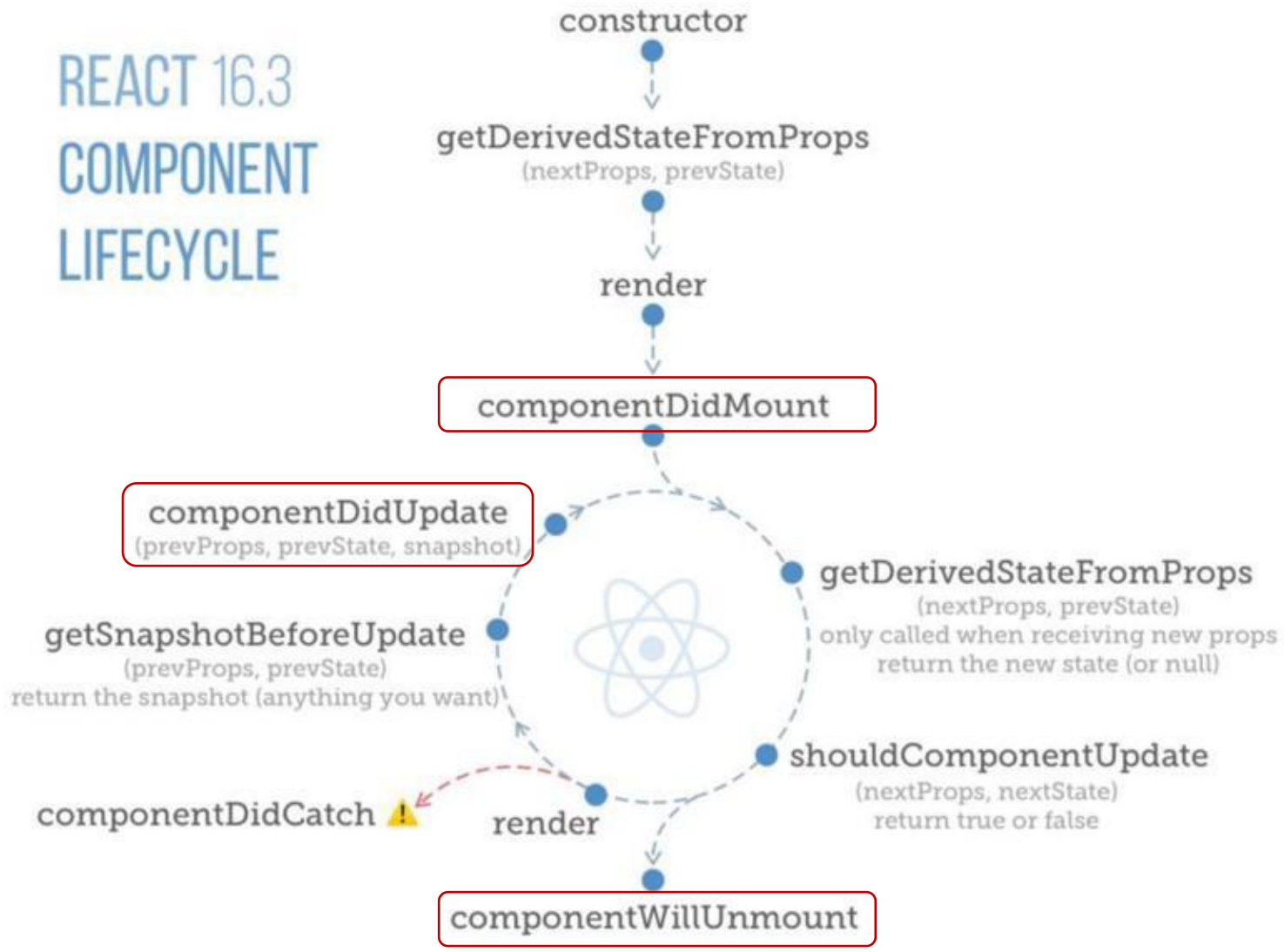
→ Implique des notions de POO (**P**rogrammation **O**rientée **O**bjets)

→ Dispose d'un cycle de vie

→ Plus verbeux à rédiger

→ Un peu moins performant sur de grosses applications

Class Component



Function component

→ Le type de composant le plus couramment utilisés avec les conventions actuelles

→ Repose sur des fonctions JavaScript classiques

→ Utilise les hooks React

→ Moins verbeux à rédiger

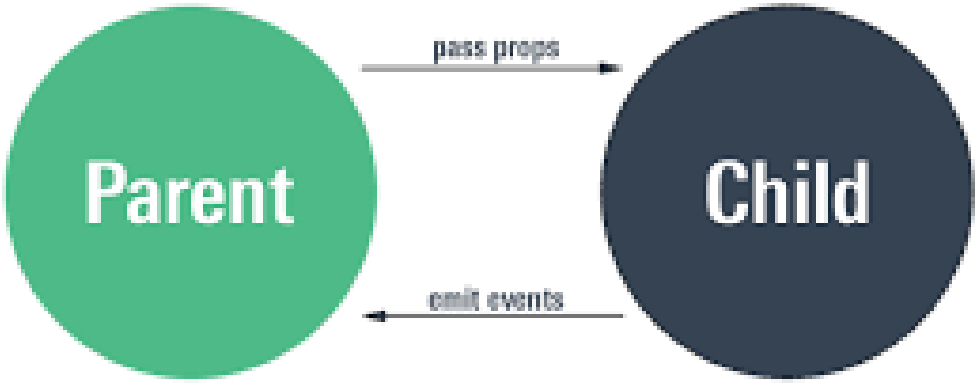
→ Un peu plus performant sur de grosses applications

Quel type de composant choisir ?

Class Component		Function Component (Using Hooks)
componentDidMount componentDidUpdate componentWillUnmount	→	useEffect
state	→	useState
ref={contentRef => ref}	→	useRef

Que sont les « *props* » ?

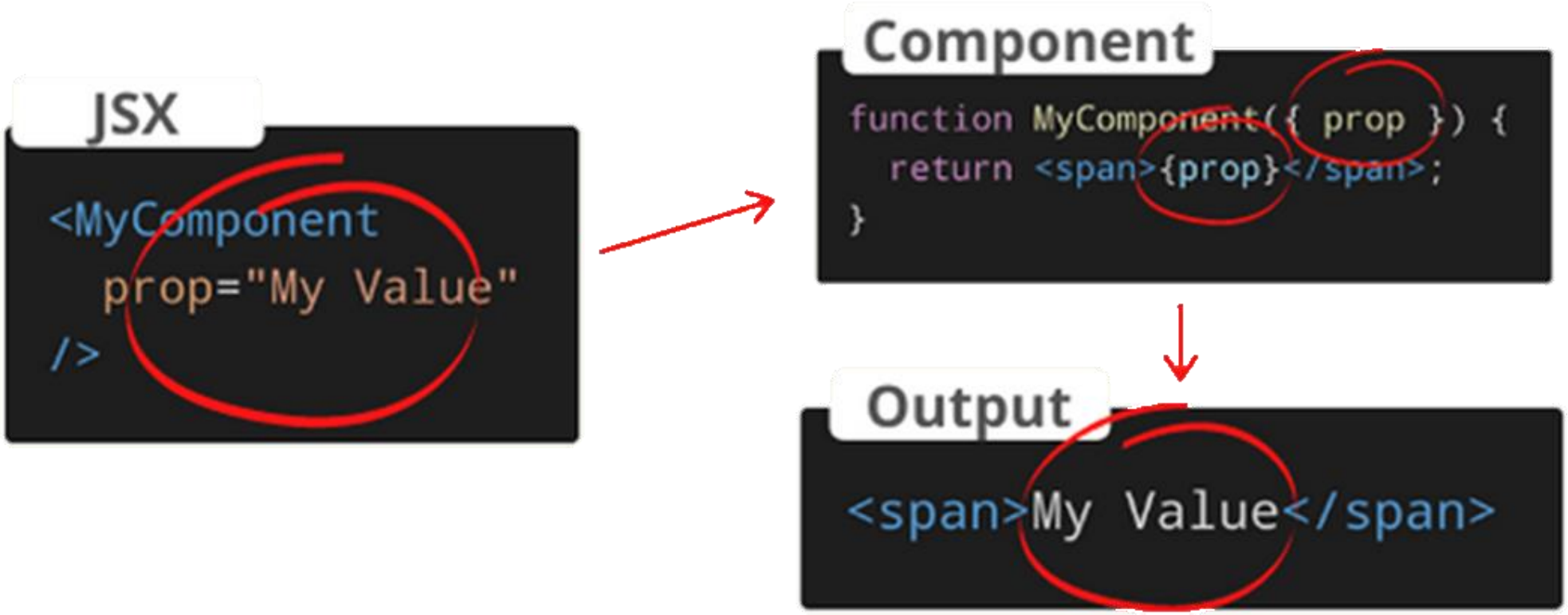
→ Des variables qui permettent de faire transiter de la donnée entre un composant parent et un composant enfant



Comment les utiliser ?

→ Les déclarer comme paramètres du composant fonctionnel

→ Préciser la valeur du paramètre là où l'on fait appel au composant



Comment les utiliser ?

1. String literals:

```
<MyComponent prop="My String Value">
```

2. Template literals with variables:

```
<MyComponent prop={`My String Value ${myVariable}`}>
```

3. Number literals:

```
<MyComponent prop={42} />
```

4. Boolean literals:

```
<MyComponent prop={false} />
```

Comment les utiliser ?

5. Plain object literals:

```
<MyComponent prop={{ property: 'Value' }} />
```

6. Array literals:

```
<MyComponent prop={['Item 1', 'Item 2']} />
```

7. JSX:

```
<MyComponent prop={<Message who="Joker" />} />
```

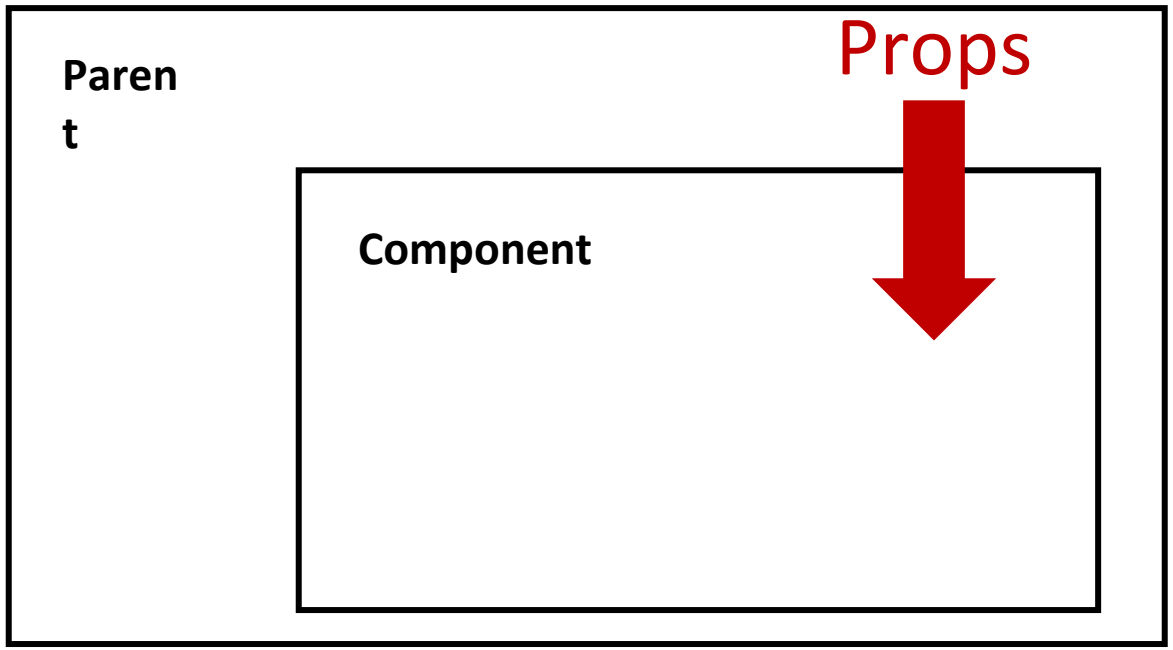
8. Variables having any kind of value:

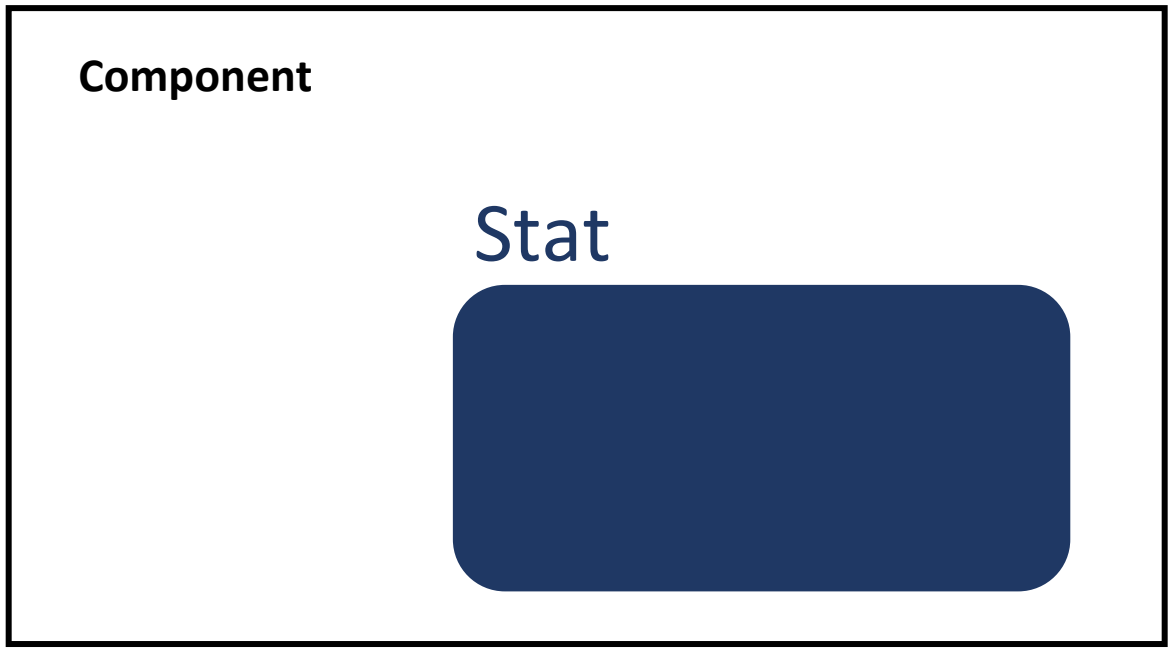
```
<MyComponent prop={myVariable} />
```

Qu'est-ce qu'un « *state* » ?

→ C'est un état interne à un composant qui est utilisé pour garder une trace des informations entre les différents rendus

→ Un state peut être assimilé à une variable locale propre au composant





Comment l'utiliser ?

```
import React, { useState } from 'react';

function Example() {
  // Déclare une nouvelle variable d'état, que l'on va appeler « count »
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Vous avez cliqué {count} fois</p>
      <button onClick={() => setCount(count + 1)}>
        Cliquez ici
      </button>
    </div>
  );
}
```

Qu'est-ce qu'un « *React hook* » ?

«Les Hooks sont des fonctions qui permettent de se brancher sur la gestion d'état local et de cycle de vie de React depuis des composants fonctionnels. Les Hooks ne fonctionnent pas dans des classes : ils vous permettent d'utiliser React sans classes. »

→ Utilisable uniquement au plus haut niveau de la fonction composant

→ S'utilise uniquement depuis un composant fonctionnel

Les différents types de hooks

useState

```
const [state, setState] = useState(initialState);
```

Renvoie une valeur d'état local et une fonction pour la mettre à jour.

Pendant le rendu initial, l'état local (state) a la même valeur que celle passée en premier argument (*initialState*).

La fonction **setState** permet de mettre à jour l'état local. Elle accepte une nouvelle valeur d'état local et planifie un nouveau rendu du composant.

```
setState(newState);
```

Les différents types de hooks

useEffect

```
useEffect(didUpdate);
```

Accepte une fonction qui contient du code impératif, pouvant éventuellement produire des effets.

La fonction fournie à **useEffect** sera exécutée après que le rendu est apparu sur l'écran. Vous pouvez considérer les effets comme des échappatoires pour passer du monde purement fonctionnel de React au monde impératif.

Par défaut, les effets de bord s'exécutent après chaque rendu, mais vous pouvez choisir d'en exécuter certains uniquement quand certaines valeurs ont changé.

Les différents types de hooks

useContext

```
const value = useContext(MyContext);
```

Accepte un objet contexte (la valeur renvoyée par *React.createContext*), et renvoie la valeur actuelle du contexte.

Un composant qui appelle **useContext** se rafraîchira toujours quand la valeur du contexte change.

Utile pour partager des props entre composants.

Les différents types de hooks

useReducer

```
const [state, dispatch] = useReducer(reducer, initialArg, init);
```

Alternative à **useState**.

useReducer est souvent préférable à **useState** quand vous avez une logique d'état local complexe qui comprend plusieurs sous-valeurs, ou quand l'état suivant dépend de l'état précédent.

Les différents types de hooks

useCallback

```
const memoizedCallback = useCallback(
  () => {
    doSomething(a, b);
  },
  [a, b],
);
```

Renvoie une fonction de rappel mémoïsée.

useCallback renverra une version mémoïsée (mise en cache) de la fonction de rappel qui changera uniquement si une des entrées a changé.

Les différents types de hooks

useMemo

```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

Renvoie une valeur mémoïsée.

useMemo recalculera la valeur mémoïsée seulement si une des entrées a changé. Cette optimisation permet d'éviter des calculs coûteux à chaque rendu.

Les différents types de hooks

useRef

```
const refContainer = useRef(initialValue);
```

useRef renvoie un objet *ref* modifiable dont la propriété *current* est initialisée avec l'argument fourni (*initialValue*).

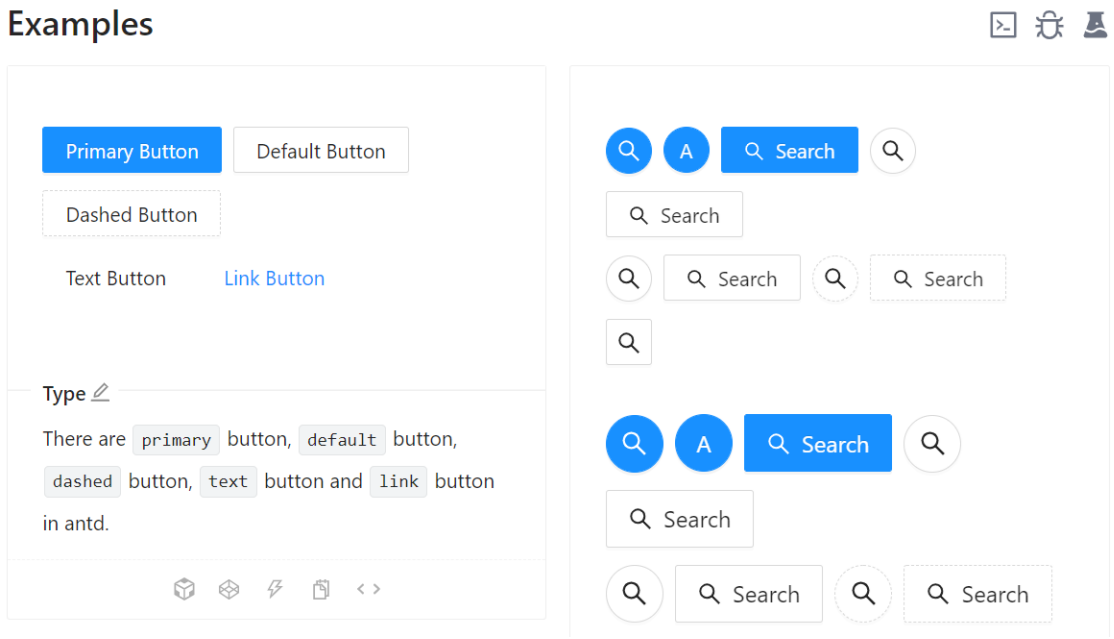
L'objet renvoyé persistera pendant toute la durée de vie du composant.

Pourquoi utiliser une bibliothèque de composants ?

→ Fournit de nombreux composants prêts à l'emploi

→ Évite de recréer des composants déjà existants

→ Gain de temps et d'effort



Quelques exemples

Ant Design

Blueprint

Material UI

PrimeReact

React Toolbox

React Bootstrap

Quelles précautions ?

→ S’assurer que la bibliothèque répond aux attentes

→ Vérifier la compatibilité avec les versions de Node et React utilisées

→ Dispose d’une communauté active (évolution, correctifs, support)

→ Être conscient de la dépendance à la bibliothèque utilisée (abstraction)

JavaScript

« JavaScript est un langage de programmation qui permet d'implémenter des mécanismes complexes sur une page web. »

→ 3ème couche des standards du WEB (avec HTML et CSS)

→ Exécuté par le moteur JavaScript du navigateur

→ Langage procédural interprété

→ Création, récupération et modification de contenu dynamiquement

→ Système d'évènements

ECMAScript

→ Ensemble de normes concernant les langages de programmation de type script

→ Standardisé par la société ECMA International

→ Version de référence : ES6 (ECMAScript 6 de 2015) = transformation majeure

→ Chaque version apporte son lot de nouvelles fonctionnalités

TypeScript

→ Langage open source, développé comme un sur-ensemble de Javascript

→ Tout code valide en Javascript l'est également en TypeScript

→ TypeScript introduit la notion d'un typage un peu plus fort que pour le langage Javascript