Frontend

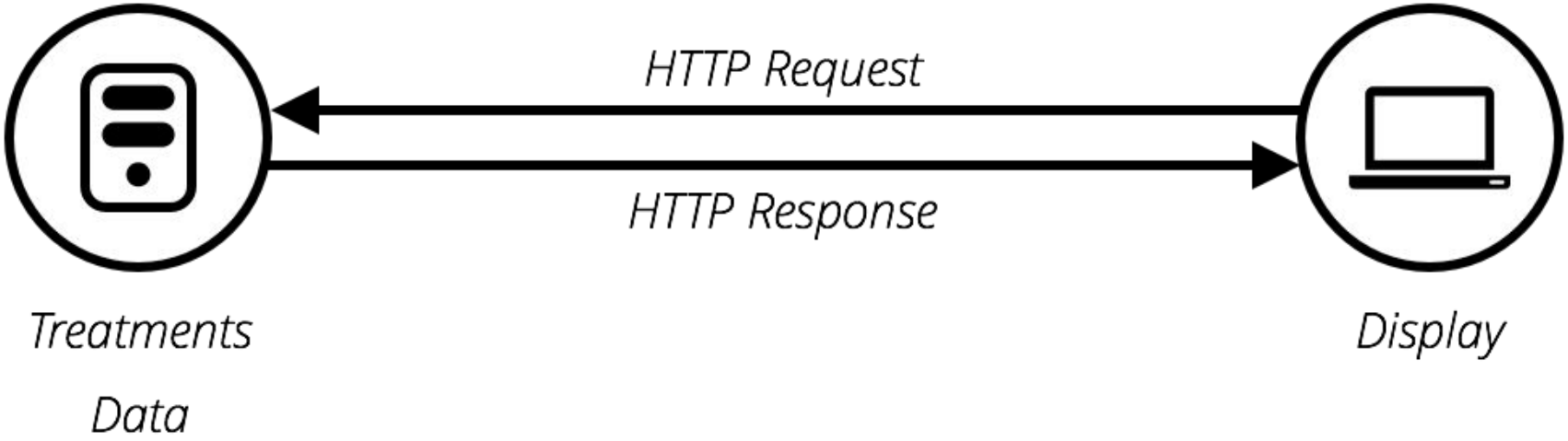
DSIA-5102A

État actuel du frontend web & consommation des APIs

Client-side & server-side rendering, JavaScript, frameworks, UX & DX, reactivity, bandwidth consumption, APIs, asynchronous requests

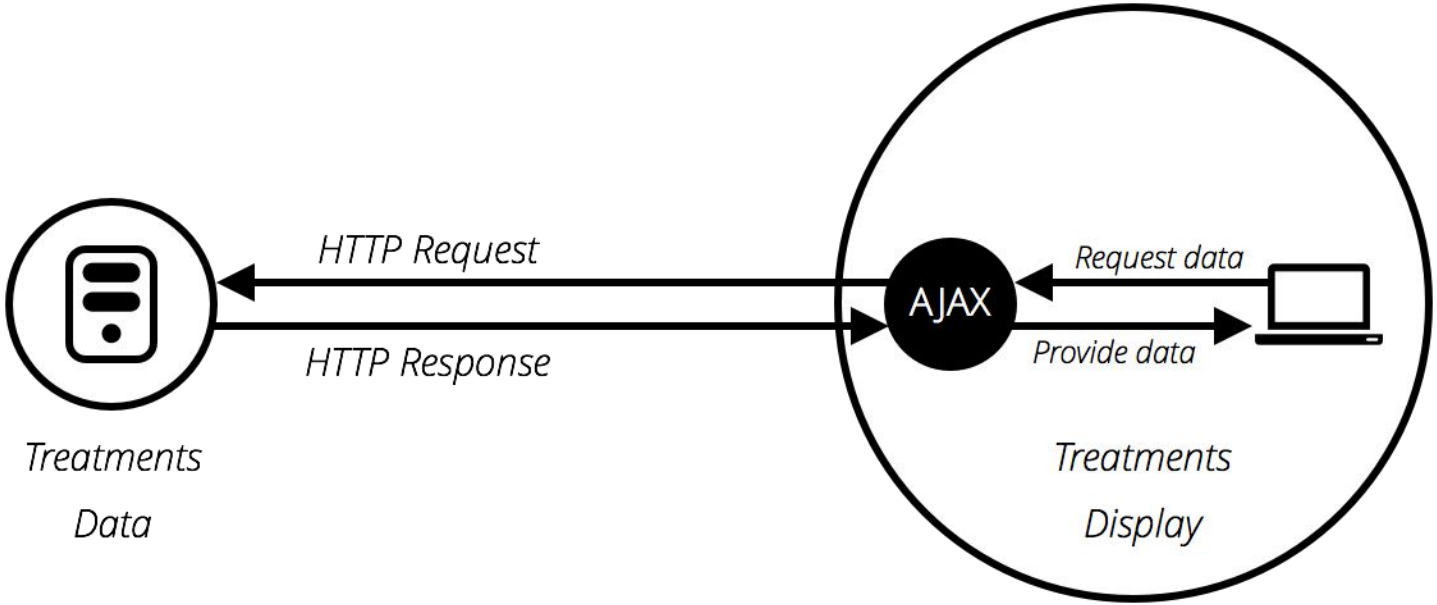
Website rendering

Au début



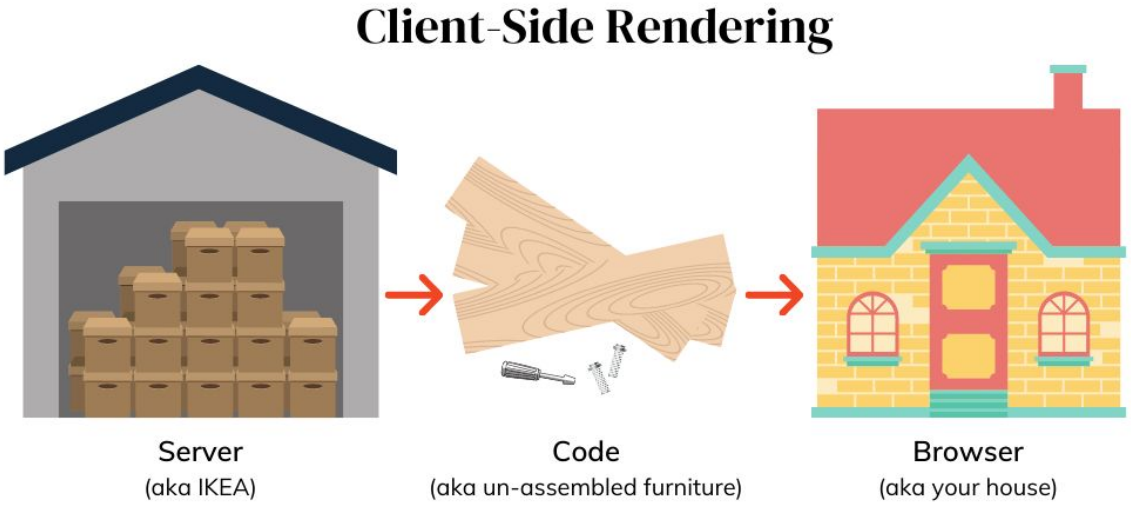
Server-side rendering

Entre temps



Ajax - Asynchronous JavaScript and XML

Et enfin



Client-side rendering

Fonctionnement

Server-side rendering

- Page générée entièrement: HTML, données, logique
- Rafraîchissement de la page entière à chaque requête

➢ Contenu, validation, données

Client-side rendering

- Page HTML sans contenu d'abord
- Puis JavaScript génère le contenu
- Rafraîchissement de quelques bouts à la fois

➢ Apparence et interactivité

Trade-offs

Server-side rendering

- Requêtes aux différents services dans le LAN du web server
- Poids total des assets élevé
- Web server à faire tourner
- SEO-friendly
- Ressources serveur ajustables
- Langages différents: PHP/Ruby/Python + JavaScript

Client-side rendering

- Requêtes aux différents services avec des allers-retours
- Poids léger des assets
- Contenu statique simple à héberger
- Chargement initial
- Pas SEO-friendly
- Ressources client
- Langage unique: JavaScript

CSR x SSR = ❤️

- Première phase: SSR render la page entière
- Phases suivantes d'hydratation: téléchargement du code client et son exécution
- Applications dites universelles ou isomorphiques
- Complexité de la stack hybride

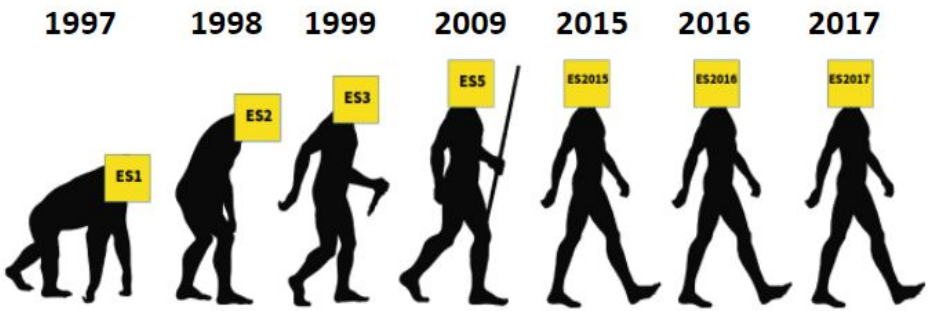
Pre-rendering

CSR et des librairies dédiées

1. Écran de chargement
2. Squelette de la page
3. Page entièrement rendered

JavaScript

Évolution de JS



- Un peu d’interactivité dans les pages de papi et mamie
- Standardisation par ECMA
- NodeJS, ou JS sans browser
- Frameworks CSR pour le web
- Framework Electron pour les apps desktop

Chromium

2008

- Open-source JavaScript engine
- Chrome, Opera, Edge, Brave

Frameworks

Frameworks

Why?

- Amélioration de l’expérience utilisateur et développeur; UX & DX
- Ecosystèmes, tooling
- Communautés grandes et actives

Frameworks

Client-side

- Trend des SPA - Single Page Applications
- Expérience ‘native’
- Comportement ‘réactif’
- Component-driven, data-driven
- Consommation d’APIs
- Exemple: Cycle de vie de Vue

Frameworks

Client-side

Livraison typique d’un site:

1. Ecriture du code
2. Build du code
    a. Transpilation
    b. Minification
3. Distribution du code

Frameworks

Client-side

- AngularJS: DOM templating
- ReactJS: JSX
- VueJS: DOM templating
- Et beaucoup (beaucoup) d’autres frameworks

Frameworks

Server-side

- Langages: PHP, Ruby, Python, JavaScript
- Interaction directe avec les composants piliers: backend, APIs, database, etc.
- Pages générées avec du templating dans le langage du framework
- Consommation d’APIs ou communication directe avec le backend

Frameworks

Server-side

- PHP: Laravel, Symfony
- Ruby: Ruby on Rails
- Python: Django, Flask
- WordPress

Consommer des APIs

Fonctionnement

Synchrone

- Code exécuté jusqu’à sa fin avant que la suite soit exécutée

Asynchrone

- Code suivant exécuté avant que le code asynchrone soit terminé

Fonctionnement

Synchrone

- Durée d’exécution courte
- Résultat ou la complétion sans surprise
- Nécessite une exécution séquentielle

Asynchrone

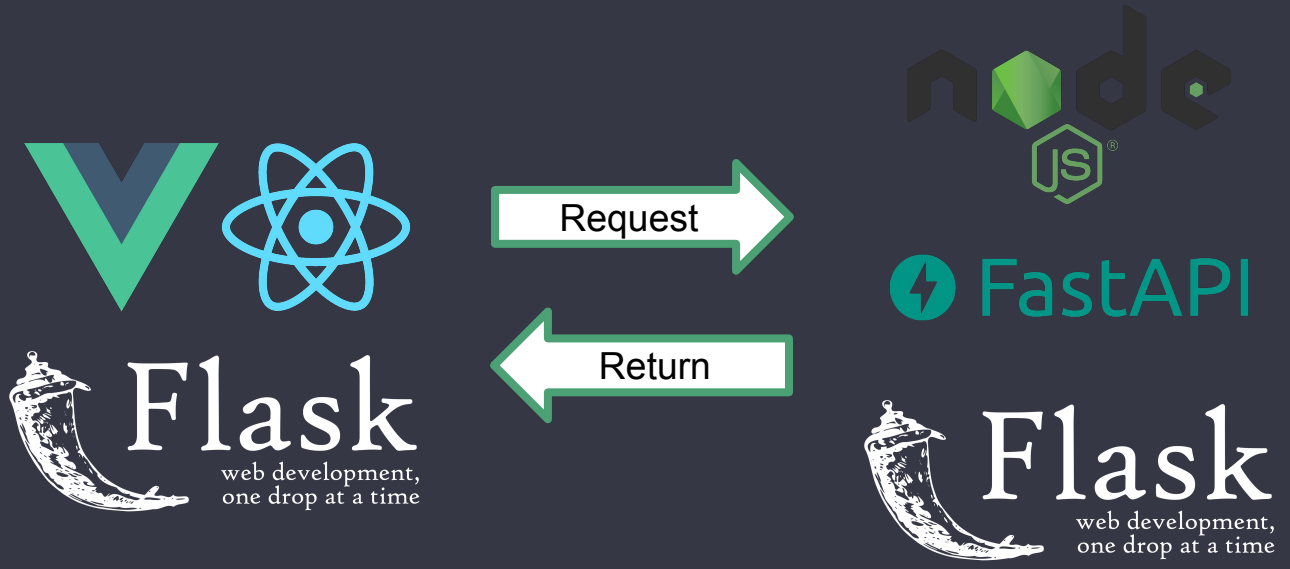
- Durée d’exécution longue
- Plusieurs bouts de code indépendants

Fonctionnement asynchrone

- Promesse: the Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.
- 3 status: pending, fulfilled, rejected
- 2 actions résultantes: then ou catch

Frontend moderne

- Client-side rendering force à utiliser des requêtes APIs
- Lourd coût des requêtes internet en temps
- Utilisation si possible de requêtes asynchrones


Request
Return
Flask
web development,
one drop at a time
FastAPI
Flask
web development,
one drop at a time