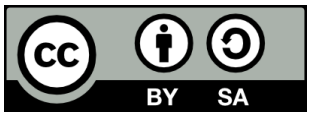
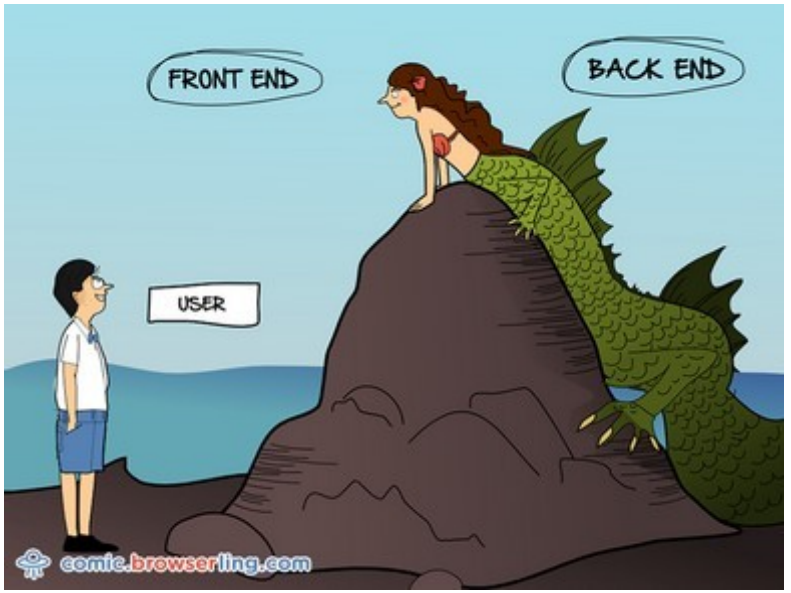
Approches de conception d'une application Web



Contenu

- **Rappel : Frontend vs Backend**
- **Rappel : API Web**
- **Approches de conception d’une application Web**
  - Server-Side Rendering (SSR) « traditionnel »
  - SSR traditionnel + AJAX
  - Client-Side Rendering (CSR)
  - Server-Side Rendering (SSR) moderne

Rappel : Frontend vs Backend



Frontend

- **Le frontend d'une application Web désigne tout ce qui est manipulé directement par le navigateur**
  - HTML
  - CSS
  - JavaScript
  - Ressources (*assets*, ex : images)
- **Frontend = Côté client**

Backend

- **Le backend désigne le code exécuté côté serveur**
- **Interagit généralement avec une base de données**
- **Peut être développé dans une grande variété de langages, ex :**
  - PHP
  - JavaScript (Node.js)
  - Java
  - Python
  - Ruby
  - C#

Rappel : API Web

API Web

- **Une API Web permet d’échanger des données avec un backend à travers un protocole Web**

API Web

- **Avec une API de type REST, on utilise les méthodes HTTP pour envoyer ou recevoir des données, ex :**
  - « GET users/Spongebob » pour récupérer les informations sur l'utilisateur « Spongebob »
  - « POST users/Spongebob » pour mettre à jour les informations sur l'utilisateur « Spongebob »
- **Les données échangées ne sont pas en HTML**
  - Typiquement JSON ou XML

Approches > Server-Side Rendering (SSR) « traditionnel »

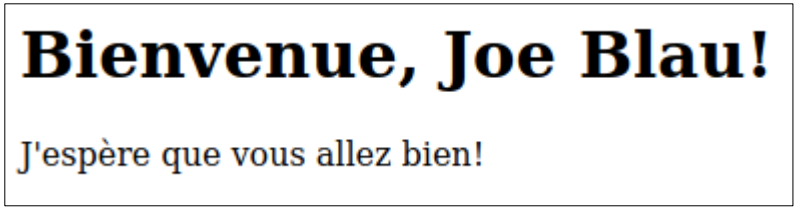
Server-Side Rendering (SSR) « traditionnel »

- **Le Server-Side Rendering (SSR) « traditionnel » correspond à la façon dont vous avez utilisé PHP jusqu’à maintenant**
  - PHP génère du HTML
  - Les données sont récupérées depuis la base de données et insérées directement dans la page avant de la retourner au client
  - Se prête bien à l'architecture MVC

Exemple de site Web dynamique

Page « login »

Page « accueil »



Le nom affiché (Joe Blau) dépend de l’utilisateur qui s’est connecté.

Chargement de la page « login.php »

Protocole HTTP

Navigateur Web

Hé mec, pourrais-tu me transmettre le contenu de la page **login.php** s'il-te-plaît?

Serveur Web

Bien sûr, le voici:

```
<html>
 <head>
  <title>Connexion à Mon super site Web</title>
 </head>
 <body>
  <h1>Connexion</h1>
  <form action="accueil.php">
   <label for="login-input">Nom d'utilisateur:</label>
            <input type="text" id="login-input" name="login" />
```

...

Actions de l’utilisateur

- **Joe Blau entre son nom d’utilisateur et son mot de passe sur la page « login.php »**
- **Il clique sur le bouton « Valider »**

Requête du navigateur pour la page « accueil.php »

Protocole HTTP

Navigateur Web

Hé mec, pourrais-tu me transmettre le contenu de la page **accueil.php** s'il-te-plaît?

Ah oui, j'oubliais. L'utilisateur m'a aussi fourni les données suivantes:

login: joeblau
password: superMotDePasse

Serveur Web

(sur un ton irrité)

Un chausson avec ça?

Transmission de la demande au serveur applicatif

Serveur Web

Mon bon ami, pourrais-tu s'il-te-plaît me fournir le contenu de la page **accueil.php**?
Le navigateur m'a transmis les données suivantes:

login: joeblau
password: superMotDePasse

Algorithme de génération du contenu de la page « accueil.php »

1. Récupérer les informations sur l'utilisateur depuis la base de données

2. Générer le HTML suivant en y insérant le nom de l'utilisateur:

```
<html>
  <head>
    <title>Mon super site Web</title>
  </head>
  <body>
    <h1>Bienvenue, <strong>prénom nom</strong>!</h1>
    <p>J'espère que vous allez bien!</p>
  </body>
</html>
```

Récupération des informations depuis la base de données

Mon bon monsieur, auriez-vous l'amabilité de me fournir le prénom et le nom de l'utilisateur qui a pour identifiant **joeblau**, et comme mot de passe **superMotDePasse**?

Sans problème, cher ami! Le prénom de l'utilisateur est **Joe**, et son nom de famille est **Blau**.

Serveur de bases de données

HTML généré

Le script PHP génère donc le HTML suivant:

```
<html>
  <head>
    <title>Mon super site Web</title>
  </head>
  <body>
    <h1>Bienvenue, <strong>Joe Blau</strong>!</h1>
    <p>J'espère que vous allez bien!</p>
  </body>
</html>
```

Transmission du HTML généré au serveur Web

Serveur Web

Voici le contenu de la page **accueil.php**:

```
<html>
  <head>
    <title>Mon super site Web</title>
  </head>
  <body>
    <h1>Bienvenue, <strong>Joe Blau</strong></h1>
    <p>J'espère que vous allez bien!</p>
  </body>
</html>
```

Transmission du HTML généré au navigateur

Protocole HTTP

Navigateur Web

Serveur Web

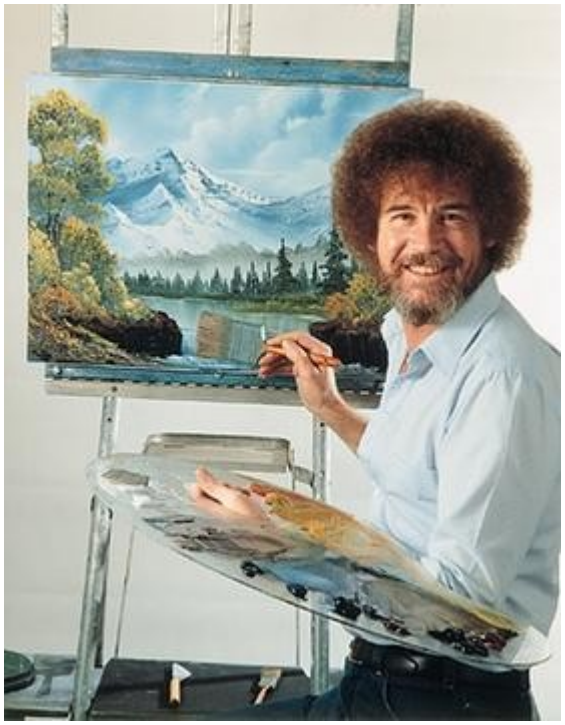
Voici le contenu de la page **accueil.php**:

```
<html>
  <head>
    <title>Mon super site Web</title>
  </head>
  <body>
    <h1>Bienvenue, <strong>Joe Blau</strong></h1>
    <p>J'espère que vous allez bien!</p>
  </body>
</html>
```

Rendu de la page Web par le navigateur

```
<html>
  <head>
    <title>Mon super site
Web</title>
  </head>
  <body>
    <h1>Bienvenue, <strong>Joe
Blau</strong></h1>
    <p>J'espère que vous allez
bien!</p>
  </body>
</html>
```

Interprétation

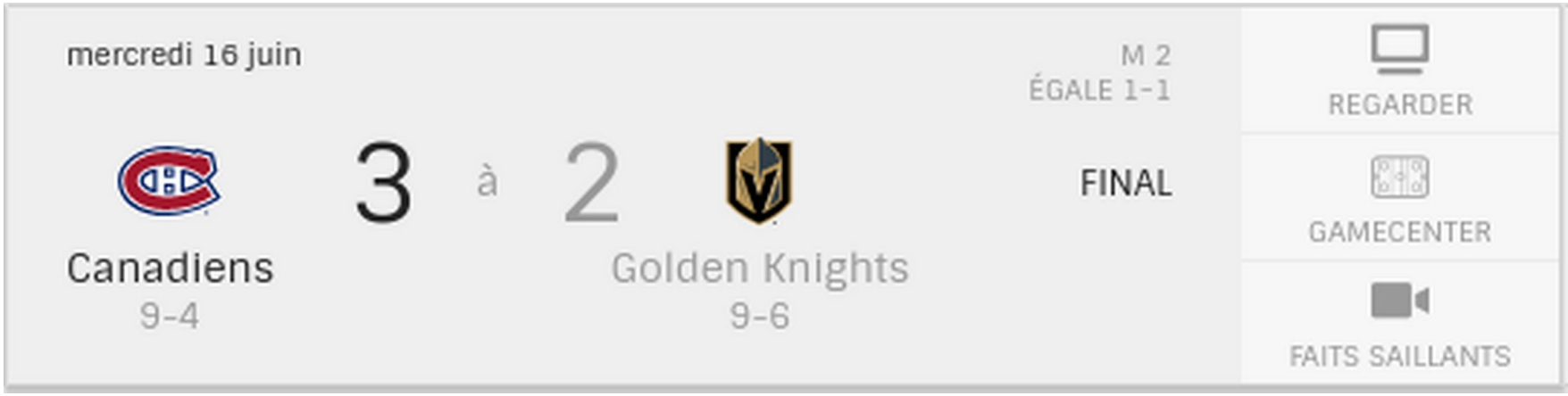


Le navigateur « dessine »
le contenu de la page
selon le HTML reçu

Approches > SSR traditionnel + AJAX

Exemple

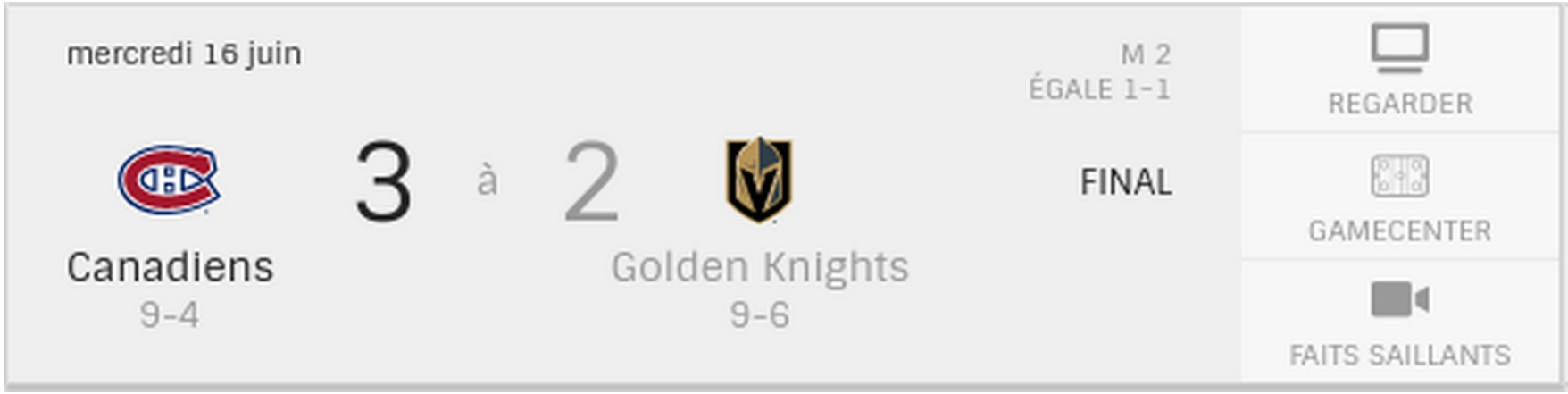
- **Imaginez que vous voulez suivre les scores lors d'un match des Canadiens :**



https://www.nhl.com/fr/canadiens/

Exemple

- **En SSR, les données sont injectées dans la page lors de sa génération par le serveur**
- **Si cette page était conçue uniquement en SSR, il faudrait la rafraîchir manuellement pour mettre les scores à jour**



https://www.nhl.com/fr/canadiens/

AJAX

- **Puisque le backend possède une API, on peut ajouter du JavaScript permettant de :**
  - Récupérer les scores à jour à une intervalle régulière (ex : toutes les 10 secondes)
  - Modifier le DOM pour mettre à jour les scores affichés au besoin
- **La page n'est pas rafraîchie au complet, seules les parties nécessaires sont modifiées**
  - Pas d'action visible de rafraîchissement dans le navigateur
  - C'est ce qu'on appelle AJAX (*Asynchronous JavaScript and XML*)

SSR + AJAX

- **On peut facilement ajouter des fonctionnalités AJAX à des applications conçues en SSR traditionnel**
- **On conserve donc la même architecture côté serveur (ex : MVC)**
  - On ajoute une API
  - Le code JavaScript côté client peut communiquer avec l'API suite au rendu initial de la page

Approches > Client-Side Rendering (CSR)

Client-Side Rendering (CSR)

- **Le *Client-Side Rendering* (CSR) désigne une approche où tout le rendu (génération du HTML) est effectué par le code JavaScript du frontend**

Client-Side Rendering

- **Le navigateur télécharge…**
  - Une page HTML presque vide
  - BEAUCOUP de JavaScript (lié à la page HTML)
- **Le JavaScript…**
  - Effectue des requêtes à l'API du backend pour échanger des données avec lui
  - « Redessine » le contenu de la page en fonction des actions de l'utilisateur et des données reçues du backend
- **Généralement développé à l'aide d'un framework ou librairie frontend**
  - Ex : React, Vue.js, Angular

Récupération du HTML

Protocole HTTP

Navigateur Web

Hé mec, pourrais-tu me transmettre le contenu de la page **index.html** s'il-te-plaît?

Serveur Web

Bien sûr, le voici:

```
<html>
  <head>
    <title>Mon super site Web</title>
    <script src="monSuperJavaScript.js" />
  </head>
  <body>
    ...
```

Récupération du JavaScript

Protocole HTTP

Navigateur Web

J'aurais aussi besoin du contenu de monSuperJavascript.js s'il-te-plaît.

Serveur Web

Tu en demandes beaucoup, comme toujours! Je te transmets le contenu de monSuperJavaScript.js à l'instant.

Algorithme du code JavaScript

- **Si l'utilisateur n'a pas encore fourni son nom d'utilisateur et son mot de passe:**
  - Insérer le HTML du formulaire de login dans le **body** du HTML affiché à l'écran
  - Lorsque l'utilisateur clique sur « Valider », passer à la prochaine étape
- **Une fois que l'utilisateur a fourni son nom d'utilisateur et son mot de passe:**
  - Récupérer son prénom et son nom depuis le backend
  - Générer le HTML suivant en y insérant le prénom et le nom de l'utilisateur:

    <h1>Bienvenue, {prénom et nom}!</h1>

    J'espère que vous allez bien!
  - Insérer le HTML généré dans le **body** du HTML affiché à l'écran

Appel de l'API par le code JavaScript du frontend

Protocole HTTP

Navigateur Web

J'aimerais appeler l'API **GET UserInfo** avec les informations suivantes:

login: joeblau
password: superMotDePasse

Serveur Web

Transmission de la demande au serveur applicatif

Serveur Web

J'aimerais appeler l'API **GET UserInfo**
avec les informations suivantes:

login: joeblau
password: superMotDePasse

Algorithme de GET UserInfo

1. Récupérer les informations sur l’utilisateur depuis la base de données

2. Retourner le résultat suivant:

{ firstName: "prénom de l’utilisateur", lastName: "nom de l’utilisateur" }

Récupération des informations depuis la base de données

Mon bon monsieur, auriez-vous l'amabilité de me fournir le prénom et le nom de l'utilisateur qui a pour identifiant **joeblau**, et comme mot de passe **superMotDePasse**?

Sans problème, cher ami! Le prénom de l'utilisateur est **Joe**, et son nom de famille est **Blau**.

Serveur de bases de données

Résultat généré

Le script PHP génère donc le résultat suivant:

```
{ firstName: "Joe", lastName: "Blau" }
```

C’est beaucoup plus petit que le HTML généré en SSR!

Transmission du résultat au serveur Web

Serveur Web

Voici le résultat de l'appel d'API:

{firstName: "Joe", lastName: "Blau" }

Transmission du résultat au navigateur

Protocole HTTP

Navigateur Web

Voici le résultat de l'appel d'API:

{ firstName: "Joe", lastName: "Blau" }

Serveur Web

Génération du HTML par le frontend

Le frontend sait donc qu'il doit générer le HTML suivant:

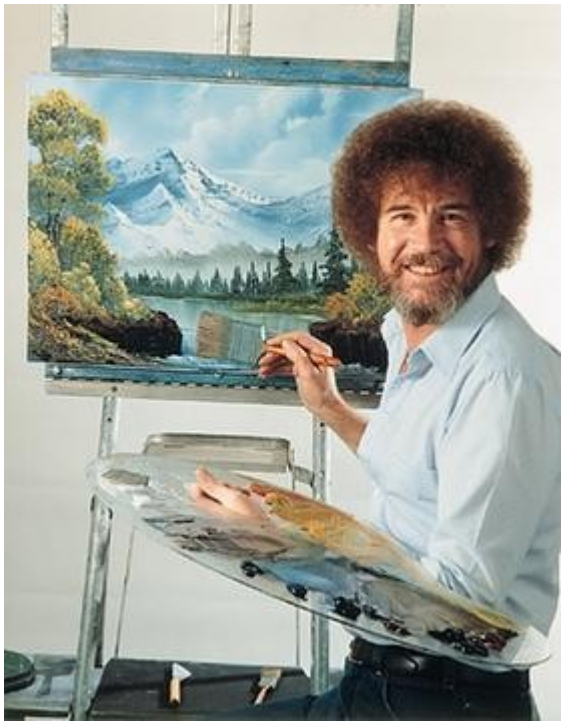
```
<h1>Bienvenue, Joe Blau!</h1>
<p>J'espère que vous allez bien!</p>
```

Il l'insère ensuite entre les balises <body> et </body> du HTML affiché à l'écran.

Rendu de la page Web par le navigateur

```
<html>
  <head>
    <title>Mon super site
Web</title>
  </head>
  <body>
    <h1>Bienvenue, <strong>Joe
Blau</strong></h1>
    <p>J'espère que vous allez
bien!</p>
  </body>
</html>
```

Interprétation



Le navigateur « dessine »
le contenu de la page
selon le HTML généré

Avantages du Client-Side Rendering

- **Séparation claire du frontend et du backend**
  - Ce sont deux applications distinctes!
  - Spécialisation possible des développeurs
  - Le même backend peut être réutilisé pour d'autres applications (ex : application mobile, objets connectés, etc)
- **Coûts de serveurs moins élevés**
  - Le serveur fait moins de travail
  - Mise à l'échelle (*scaling*) plus facile

Inconvénients du Client-Side Rendering

- **Plus de travail pour le client = moins performant sur les appareils peu puissants**
- **Moins bon pour le référencement**
  - Les moteurs de recherche ne laissent pas toujours le rendu se faire au complet au moment d'indexer une page
    - Anciennement, ils n'exécutaient pas le JavaScript du tout
  - Pas nécessairement grave pour une application transactionnelle, on devrait avoir une « *landing page* » statique

Single Page Application (SPA)

- **On voit souvent le terme « *Single Page Application* » ou SPA sur le Web**
- **Il s'agit simplement d'une application en CSR qui ne comporte qu'une seule page HTML**
  - Le JavaScript peut modifier le DOM au complet pour changer la « page » affichée, mais on ne voit jamais le navigateur charger une nouvelle page Web
  - Plus fluide pour l'utilisateur
  - Reproduit le « feeling » d'une application native
- **Exemples de SPA :**
  - Gmail
  - Version Web de Microsoft Teams
  - Netflix

Approches > Server-Side Rendering moderne

Approche moderne du SSR

- **Des inconvénients du CSR, est né le SSR moderne!**

Approche moderne du SSR

- **On développe une application en CSR « presque » normalement**
  - Il peut y avoir quelques différences, selon la technologie utilisée
- **Lors du premier accès à la page, le JavaScript (appels d'API, manipulations du DOM, etc) est exécuté côté serveur**
- **Le HTML généré est retourné au client plutôt qu'une page HTML vide**
- **Par la suite, la page se comporte comme en CSR**

Approche moderne du SSR

- **On ne fait pas du SSR « manuellement »**
- **On utilise une librairie, un framework, etc qui le permet**
  - Ex : Next.js

Avantages et inconvénients

- **Avantages**
  - *Peut* être meilleur pour le référencement
  - Moins de travail à faire côté client, donc plus performant sur les appareils peu puissants
  - Moins de requêtes HTTP (les données initiales sont déjà sur la page)
- **Inconvénients**
  - Coûts de serveur plus élevés
  - Déploiement plus complexe

Fin de la présentation

Des questions?

Photo par Jules Bss sur Unsplash