h e g

Haute école de gestion
Genève

Analyse des différentes méthodes de mise en cache Web

Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES

par :

Pamela VERDON

Conseiller au travail de Bachelor :

Sonia PERROTTE, chargée de cours

Genève, le 23 septembre 2024

Haute École de Gestion de Genève (HEG-GE)

Filière Informatique de gestion

Hes·so GENÈVE
Haute Ecole Spécialisée de Suisse occidentale

Déclaration

Ce travail de Bachelor est réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre Bachelor of Science HES-SO en Informatique de gestion.

L'étudiant a envoyé ce document par email à l'adresse remise par son directeur de mémoire afin qu'il l'analyse à l'aide du logiciel de détection de plagiat COMPILATIO.

L'étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de Bachelor, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de Bachelor, du juré et de la HEG.

« J'atteste avoir réalisé seule le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie. »

Fait à Genève, le 20 septembre 2024

Verdon Pamela

.

Remerciements

Je tiens à remercier tout particulièrement la directrice de mon travail de Bachelor, Madame Sonia Perrotte, pour son accompagnement, ses conseils et ses corrections qui ont été essentiels à la réalisation de mon travail.

Je remercie également ma sœur et mon frère pour leur relecture attentive et leurs précieuses remarques qui m'ont aidée à peaufiner ce document.

Résumé

Ce travail explore les méthodes de mise en cache sur le Web, avec pour objectif principal d'évaluer leur efficacité et d'identifier celles qui conviennent le mieux à divers scénarios. La mise en cache est un élément clé pour améliorer les performances des sites et applications Web, mais il est essentiel de comprendre quelles approches sont les plus adaptées à chaque situation.

Dans un premier temps, une étude théorique des mécanismes de stockage sur le Web est menée. Elle sert à clarifier la différence entre le stockage et la mise en cache. Cette distinction est importante car, même si le stockage et la mise en cache sont tous deux des formes de gestion des données, leurs objectifs et leur utilisation diffèrent. Puis, le travail se concentre sur les différentes méthodes de mise en cache, en décrivant théoriquement leurs aspects, en fonction de leur implémentation côté client, côté serveur ou dans des environnements intermédiaires, comme les caches proxy ou les CDN.

Dans un deuxième temps, plusieurs technologies de mise en cache sont comparées et notées selon deux critères : la performance et la sécurité. Cette analyse comparative est principalement un indicateur qui sert à comprendre les avantages et inconvénients de chacune des technologies.

Enfin, ce travail se concrétise par des tests de performance menés dans un environnement local et contrôlé, où les concepts discutés sont mis en pratique. Une application de test simulant un site de vente en ligne de vêtements a été spécialement créée. Elle a été mise en place avec Express, MySQL et Vue.js et, pour le cache, Redis et un service worker ont été utilisés. Les résultats démontrent que la mise en cache a un réel impact positif sur les performances, particulièrement dans le cas de la mise en cache côté serveur avec Redis, où la performance peut être améliorée de 90%.

Table des matières

Liste des tableaux

Liste des figures

Table des acronymes

ACL	Access Control List
API	Application Programming Interface
AWS	Amazon Web Services
CDN	Content Delivery Network
CSS	Cascading Style Sheets
DDoS	Distributed Denial of Service
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LAN	Local Area Network
PDF	Portable Document Format
RAM	Random Access Memory
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTL	Time To Live
URL	Uniform Resource Locator
VU	Virtual User

1. Introduction

Dans un monde de plus en plus connecté, les performances des sites Web et des applications jouent un rôle central dans la satisfaction des utilisateurs et le succès des entreprises. La mise en cache, une technique de stockage temporaire de données, se présente comme un mécanisme indispensable pour améliorer la rapidité et l'efficacité des interactions sur le Web.

Historiquement, le Web a évolué depuis de simples pages statiques vers des applications dynamiques et interactives (Mendez 2021). Cette évolution a rendu nécessaire des solutions plus sophistiquées pour gérer efficacement les ressources et améliorer les performances. C'est le cas de la mise en cache, qui permet d'accéder rapidement aux informations et qui réduit la charge sur les serveurs.

Des études montrent que les temps de chargement des pages ont un impact direct sur la satisfaction des utilisateurs et leur propension à effectuer des achats sur un site d'e-commerce (Deloitte Ireland 2020). Une gestion adéquate du cache peut donc se traduire par des gains économiques et une fidélisation des utilisateurs.

Malgré son importance, la mise en cache est loin d'être une solution simple à implémenter. Les principaux défis se trouvent dans la gestion efficace des caches côté client et serveur, la synchronisation des données mises en cache, et les implications de sécurité. Par exemple, une mise en cache incorrecte peut entraîner des problèmes de cohérence des données, où les utilisateurs voient des informations périmées ou erronées. De plus, certaines stratégies de cache peuvent introduire des vulnérabilités de sécurité, exposant les systèmes à des cyberattaques potentielles.

Ce travail de Bachelor a pour but d'analyser les différentes méthodes actuelles de mise en cache, d'évaluer leur efficacité et de proposer les meilleures pratiques adaptées à des situations spécifiques. L'objectif est de fournir un cadre analytique qui permettra de comprendre les avantages et les inconvénients des diverses approches de mise en cache, tout en offrant des recommandations pour leur mise en œuvre.

Pour atteindre cet objectif, je commencerai par étudier les origines et la structure du Web, pour fournir une base nécessaire à la compréhension des concepts de stockage et de mise en cache. Ensuite, je me pencherai sur les différentes méthodes de stockage sur le Web, en distinguant le stockage côté client et le stockage côté serveur.

J’approfondirai ensuite les techniques de mise en cache Web, en les catégorisant selon leur implémentation côté client, côté serveur ou de manière intermédiaire. Cette exploration servira à identifier les spécificités et les avantages de chaque méthode de mise en cache.

Par la suite, je proposerai une analyse et une évaluation des différentes méthodes de mise en cache, basées sur des critères préalablement définis. Cette analyse comparative servira à déterminer quelles méthodes sont les plus adaptées à des contextes particuliers.

Enfin, je passerai à la mise en pratique des concepts discutés en mettant en place des tests et des expérimentations dans un environnement contrôlé. Les résultats de ces tests seront présentés et discutés, afin d'évaluer l'efficacité des différentes méthodes de mise en cache dans un scénario réel.

2. Stockage sur le Web

2.1 Architecture du Web

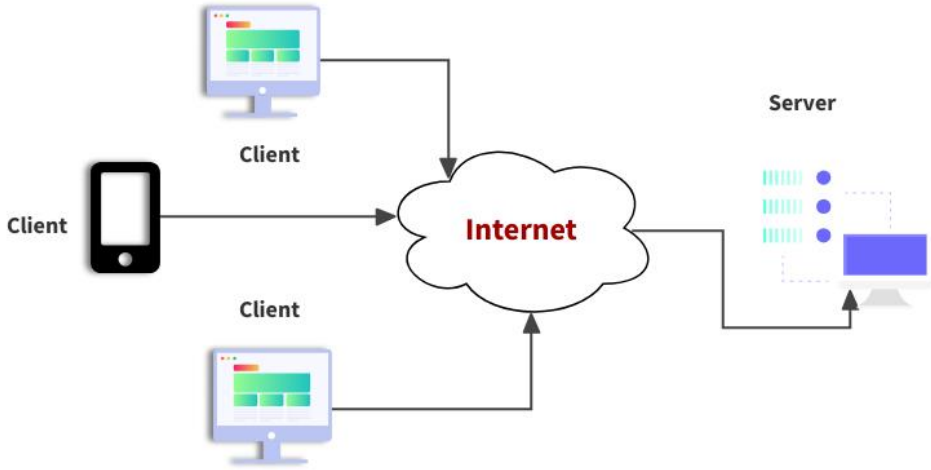
2.1.1 Explication de l'architecture client-serveur

L'architecture client-serveur est un modèle fondamental qui sous-tend la majorité des interactions sur le Web. Ce modèle repose sur la séparation des rôles entre deux principaux acteurs : le client et le serveur. Ils ont chacun un rôle distinct mais complémentaire, ce qui laisse place à une communication efficace et à une distribution des tâches. (Agarwal 2024)

Le client est généralement un utilisateur final qui envoie des requêtes pour accéder à des ressources ou des services. Il existe principalement trois types de client. Le client lourd exécute la majorité des traitements de données localement, en se reposant peu sur le serveur. Le client léger, quant à lui, utilise principalement les ressources du serveur pour le traitement des données, alors que le client hybride combine des caractéristiques des deux types précédents. Ce dernier effectue certains traitements localement tout en se fiant au serveur pour le stockage des données persistantes. (HEAVY.AI 2022)

Le serveur est un système qui héberge des ressources, des services ou des applications et répond aux requêtes envoyées par les clients. Les serveurs peuvent stocker des fichiers, des bases de données, des scripts et des applications Web. Ils traitent les requêtes des clients et renvoient les réponses appropriées. (HEAVY.AI 2022)

Figure 1 : Architecture client-serveur



(Agarwal 2024)

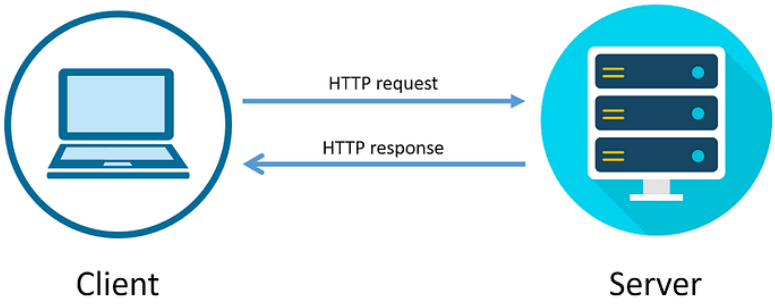
Cette figure présente l'architecture client-serveur mentionnée précédemment et montre la répartition des rôles entre clients et serveur. Il y a plusieurs clients, qu'il s'agisse d'ordinateurs ou de dispositifs mobiles, qui interagissent avec un serveur central via Internet. Les clients envoient des requêtes pour accéder aux ressources ou services, tandis que le serveur traite ces requêtes et renvoie les réponses nécessaires. Le sens des flèches du schéma indiquent le trajet d'une requête.

L'interaction entre le client et le serveur suit un cycle de requêtes-réponses. Voici comment ce cycle fonctionne généralement :

1. Initiation de la requête : Le client initie une requête en se connectant à un serveur via une URL (Uniform Resource Locator). Cette URL spécifie l'adresse du serveur et la ressource demandée.
2. Envoi de la requête : Le navigateur du client envoie une requête HTTP (HyperText Transfer Protocol) au serveur. La requête peut demander une page Web, une image, un fichier ou une action spécifique.
3. Traitement de la requête : Le serveur reçoit la requête et la traite.
4. Réponse du serveur : Le serveur envoie une réponse HTTP au client. La réponse contient le statut de la requête (succès, erreur, redirection, etc.) et les données demandées, si la requête a été réussie (comme une page HTML, une image, des données JSON, etc.).
5. Affichage des données : Le client reçoit la réponse et affiche les données ou les traite selon les instructions reçues. Par exemple, un navigateur affichera une page Web, tandis qu'une application mobile pourrait mettre à jour son interface utilisateur avec les nouvelles données.

(MDN 2024a)

Figure 2 : Cycle de requêtes-réponses HTTP



(Patil 2020)

La figure présentée image le cycle de requêtes-réponses HTTP. Un client envoie une requête HTTP au serveur, qui répond par une réponse HTTP contenant la ressource demandée ou un statut d'erreur.

L'architecture client-serveur présente plusieurs avantages. En centralisant les données sur un seul serveur, elle simplifie la gestion de la sécurité, des autorisations et de l'authentification des utilisateurs. Cette structure laisse également la possibilité d'ajouter des ressources telles que des serveurs, des segments de réseau ou des ordinateurs sans perturber le fonctionnement global du système. Les clients peuvent également partager des ressources comme des imprimantes, des fichiers et des bases de données hébergées sur le serveur, ce qui optimise l'utilisation des ressources et réduit les coûts. Enfin, cette architecture rend les clients légers indépendants, car, en s'appuyant sur le serveur pour le traitement et l'accès aux données, ils n'ont pas besoin de systèmes d'exploitation spécifiques. (Agarwal 2024)

Cependant, l'architecture client-serveur présente aussi des inconvénients. Tout d'abord, elle présente un point de défaillance unique. En cas de panne du serveur, l'ensemble du réseau devient indisponible, interrompant toutes les opérations des clients. De plus, les clients dépendent fortement d'une connexion réseau stable pour accéder au serveur, ce qui peut entraîner des problèmes de performance en cas de congestion ou de connexions lentes. Les coûts de mise en œuvre et de maintenance de cette architecture peuvent également être élevés, à cause des besoins en matériel de serveur, en licences logicielles et en expertise informatique. (Agarwal 2024)

2.1.2 Navigateurs Web

Les navigateurs Web sont des applications installées sur le dispositif du client. Ils autorisent les utilisateurs à accéder aux ressources et aux services disponibles sur le Web. Parmi les navigateurs les plus populaires, il y a Google Chrome, Mozilla Firefox, Safari, Microsoft Edge et Opera. Ces navigateurs interprètent du code HTML, CSS et JavaScript pour afficher les pages Web et assurer une interface utilisateur interactive. (Wikipedia 2024a)

2.1.3 Réseaux

Le réseau constitue l'infrastructure qui permet la communication entre les clients et les serveurs. Il comprend divers éléments comme les routeurs, les commutateurs, les câbles, et les protocoles de communication. Le réseau Internet est un réseau mondial de réseaux, laissant les dispositifs connectés échanger des informations.

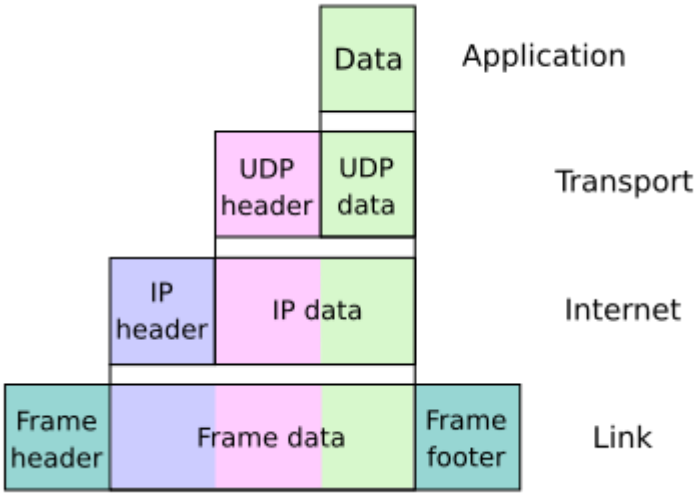
Les protocoles de communication, tels que TCP/IP (Transmission Control Protocol/Internet Protocol) et HTTP (HyperText Transfer Protocol), servent au transport et à la gestion des données sur les réseaux. HTTP fonctionne comme le langage commun permettant aux navigateurs et aux serveurs de communiquer. Il spécifie

comment les informations doivent être envoyées et reçues entre ces entités. HTTP est utilisé pour demander et transférer des ressources Web comme des pages HTML, des images, des fichiers PDF, etc. (Affonso 2024)

HTTPS (HTTP Secure) permet d'ajouter une couche de sécurité via SSL/TLS (Secure Sockets Layer/Transport Layer Security), assurant que les données échangées sont cryptées. (Affonso 2024)

TCP gère le transport des données. Il assure que les données envoyées par HTTP arrivent à destination, sans perte ni erreur, en divisant les messages en segments et en veillant à ce qu'ils soient correctement assemblés à l'arrivée. IP, de son côté, est responsable de l'acheminement de ces segments à travers le réseau, en utilisant des adresses pour diriger les données vers le bon endroit. Cette combinaison garantit que les requêtes et les réponses sont transmises correctement entre le client et le serveur. (Affonso 2024)

Figure 3 : Encapsulation des données



(Wikipedia 2024b)

Cette image met en lumière l'architecture réseau, divisée en quatre couches. Elle montre comment les protocoles de communication, tels que HTTP, TCP et IP, encapsulent les données pour leur transmission sur le réseau. Autrement dit, elle illustre comment un message est structuré pour le transport. Au sommet, les données représentent le contenu du message envoyé par le client ou le serveur. Ce contenu est ensuite encapsulé avec des en-têtes de différents protocoles à chaque niveau. HTTP, au niveau « Application », spécifie comment les informations doivent être envoyées et reçues entre le navigateur et le serveur. Ensuite, au niveau « Transport », TCP s'occupe de diviser le message en segments pour assurer une transmission fiable et sans erreur.

Au niveau « Internet », IP gère l'acheminement de ces segments à travers le réseau en utilisant des adresses IP pour diriger chaque segment vers la bonne destination. Enfin, la couche « Link » s'assure que les données sont envoyées physiquement sur le réseau. Cette figure montre donc comment chaque protocole contribue à l'encapsulation et au transport efficace des messages entre les clients et les serveurs.

2.1.4 Bases de données

Les bases de données sont des systèmes de stockage de données qui assurent aux applications Web la possibilité de stocker, récupérer et manipuler des données de manière efficace. Malgré le fait que les structures puissent varier, toutes les bases de données offrent une méthode organisée pour le traitement des informations.

2.2 Rôle et importance du stockage

Le stockage est un élément clé de l'architecture des applications et des sites Web modernes. Il s’occupe de conserver et de gérer les données nécessaires pour assurer le bon fonctionnement des services en ligne.

Une fonction importante du stockage est la conservation des données utilisateur. Les applications Web doivent souvent accéder à diverses informations, telles que les préférences des utilisateurs, le contenu des sites et les configurations spécifiques. Par exemple, les informations de connexion, les articles dans un panier d'achat et les paramètres de personnalisation doivent être sauvegardés pour que les utilisateurs puissent poursuivre leurs activités sans interruption, même après avoir fermé leur navigateur ou rechargé la page. (Ramotion 2023)

Le stockage joue également un rôle dans la gestion des sessions utilisateur. Les sessions maintiennent l'état de la connexion de l'utilisateur et conservent les informations nécessaires pendant la navigation. Les données, comme les identifiants de session, les préférences et les paramètres utilisateur, sont stockées temporairement, ce qui assure une continuité dans l'expérience utilisateur. (Microsoft 2016)

De plus, il existe des mécanismes de stockage côté client qui possèdent une capacité de stockage pouvant gérer des volumes de données conséquents. Cela facilite le développement d'applications Web, qui ont la possibilité de stocker des informations localement sans surcharger les serveurs. En autorisant les développeurs à définir la durée de conservation des données, il est possible d’améliorer la performance des applications en réduisant la fréquence des interactions avec les serveurs. (Ramotion 2023)

Le stockage du côté client contribue aussi à réduire la latence. En conservant les données localement, proches de l'utilisateur, il est facile de diminuer le temps d'accès aux informations. Pour les applications qui ont besoin de réponses rapides, cela peut s’avérer particulièrement utile. (Microsoft 2016)

Au-delà du stockage côté client, le stockage côté serveur joue un rôle complémentaire dans la gestion centralisée des données utilisateur, la persistance des informations et le traitement de grandes quantités de données. Les sessions côté serveur, par exemple, permettent de conserver des informations spécifiques à l'utilisateur de manière sécurisée et centralisée, réduisant ainsi le risque de surcharge du client avec des données volumineuses (Peña 2019). Les bases de données gèrent également les informations efficacement à grande échelle, conférant un accès rapide et structuré aux données nécessaires au fonctionnement des applications Web (Oracle 2020).

2.3 Stockage côté client

Le stockage côté client est une composante permettant aux navigateurs de conserver et de gérer les données localement. Ces mécanismes de stockage sont essentiels pour la persistance des informations à travers les sessions de navigation et pour la personnalisation des interactions.

2.3.1 Cookies

Les cookies sont de petits morceaux de données enregistrés sur le navigateur de l'utilisateur. Ces données sont renvoyées au serveur à chaque requête HTTP, pour maintenir la continuité de la session. (Ramotion 2023)

Cependant, comparés aux autres mécanismes de stockage Web comme le local storage et le session storage, les cookies ont des limites. Leur capacité de stockage est relativement petite, environ 4 Ko. En outre, la manipulation des cookies via JavaScript est moins pratique qu’avec d’autres méthodes de stockage local. (Ramotion 2023)

Malgré cela, les cookies restent largement utilisés pour des fonctions spécifiques. Ils sont spécialement efficaces pour la gestion des sessions et l'authentification des utilisateurs. Par exemple, ils peuvent conserver des identifiants de session pour maintenir la connexion entre les pages d'un site. De plus, les cookies sont souvent utilisés pour enregistrer les préférences des utilisateurs, telles que les choix de thème ou de langue, ce qui laisse la possibilité d'ajuster l'expérience en fonction des préférences personnelles. En autorisant la collecte de données pour améliorer les services proposés par les sites, les utilisateurs permettent aux cookies de suivre et d’analyser leur comportement. (Ramotion 2023)

2.3.2 Local storage

Le local storage est un mécanisme de stockage côté client qui conserve des données sans expiration automatique. Contrairement aux cookies, les données stockées dans le local storage ne sont pas envoyées au serveur avec chaque requête HTTP. (Ramotion 2023)

Utilisé pour des cas comme la conservation de jetons d'authentification, la sauvegarde des préférences utilisateur ou l'enregistrement de données de formulaires, le local storage facilite une expérience utilisateur continue et sans interruption. Les informations restent disponibles même après une fermeture imprévue du navigateur, améliorant ainsi la fiabilité des services Web. (Ramotion 2023)

Il possède une capacité de stockage significative, généralement autour de 10 Mo par domaine, ce qui permet de conserver une grande quantité d'informations, telles que des documents ou des paramètres utilisateur. Il réduit la nécessité de récupérer ces données à chaque visite et, donc, minimise les requêtes au serveur. (Microsoft 2016)

Le local storage est structuré par domaine et sous-domaine. Dès lors, chaque domaine possède son propre espace de stockage distinct. Les données stockées par un domaine sont accessibles à ses sous-domaines, mais pas à d'autres domaines ou sous-domaines qui ne seraient pas liés. La confidentialité des données est respectée en limitant leur accès aux seuls domaines et sous-domaines autorisés. (Microsoft 2016)

2.3.3 Session storage

Le session storage est conçu pour stocker des données temporaires liées à une seule session de navigation. À l'inverse du local storage qui conserve les informations même après la fermeture du navigateur, le session storage efface automatiquement les données lorsque l'onglet ou la fenêtre du navigateur est fermé. Cela en fait une option pratique pour conserver des informations pendant une session de navigation unique, comme les données d'un formulaire en cours de remplissage. (Ramotion 2023)

Le session storage a une capacité de stockage plus grande que les cookies, atteignant environ 5 Mo, ce qui est suffisant pour la plupart des besoins de stockage temporaire. (Ramotion 2023)

Chaque session de navigation est isolée, c'est-à-dire que les données stockées dans un onglet ne sont pas accessibles depuis un autre onglet ou une autre fenêtre, même si elles proviennent du même domaine. Les informations sensibles ou temporaires restent privées et ne sont pas partagées entre différentes sessions de navigation. (Ramotion 2023)

2.3.4 IndexedDB

IndexedDB est une API (Application Programming Interface) destinée au stockage côté client de grandes quantités de données structurées. Grâce à une structure basée sur des index, IndexedDB peut gérer des volumes de données très importants, comparés aux mécanismes de stockage plus simples. Cette API offre la possibilité d'effectuer des recherches rapides et efficaces à travers les données stockées, même lorsque ces dernières sont volumineuses ou complexes. (Ramotion 2023)

2.4 Stockage côté serveur

Le stockage côté serveur est essentiel pour la gestion des données des utilisateurs, pour la persistance des informations et pour la manipulation de grandes quantités de données. Contrairement au stockage côté client qui se concentre sur la gestion des données au niveau du navigateur, le stockage côté serveur implique des systèmes de gestion de données qui fonctionnent sur des serveurs distants.

2.4.1 Sessions serveur

Les sessions côté serveur conservent des informations spécifiques à l'utilisateur sur le serveur pendant qu'une session est active. Lorsqu'un utilisateur se connecte à une application Web, une session est générée et un identifiant unique est envoyé au client sous forme de cookie. Ce cookie est utilisé pour associer les requêtes ultérieures de l'utilisateur avec les données de session stockées sur le serveur. Les sessions côté serveur offrent une gestion centralisée des données utilisateur et réduisent les risques liés à la sécurité des informations sensibles. À l'opposé des cookies qui stockent directement des informations sur le client, les sessions côté serveur évitent de surcharger le client avec beaucoup de données. (Peña 2019)

2.4.2 Bases de données

Les bases de données sont conçues pour assurer un accès rapide et structuré aux données. Elles sont principalement classées en deux grandes catégories : les bases de données relationnelles et non relationnelles.

Les bases de données relationnelles, comme celles qui utilisent SQL (Structured Query Language), organisent les données en tables structurées et permettent des relations complexes entre les données. (Oracle 2020)

Les bases de données non relationnelles (NoSQL) ont émergé pour répondre à des besoins différents, notamment le stockage de données non structurées ou semi-structurées. Ces bases de données offrent une plus grande flexibilité et peuvent gérer des volumes de données massifs avec des structures variées. (Oracle 2020)

À côté des bases de données relationnelles et non relationnelles, il existe d'autres types de bases de données adaptées à des besoins spécifiques et aux évolutions technologiques récentes. Par exemple, les bases de données orientées graphe ont été élaborées pour stocker et interroger des données représentées sous forme de graphes, avec des entités et des relations entre elles. Elles sont utiles pour des applications nécessitant des analyses de relations complexes, comme les réseaux sociaux. Les bases de données à pilotage automatique, quant à elles, utilisent le Cloud et l’apprentissage automatique pour automatiser la gestion des bases de données. Ces innovations répondent aux nouveaux défis en matière de gestion des données et permettent aux entreprises de traiter des volumes de données toujours croissants, tout en optimisant les coûts et les ressources. (Oracle 2020)

3. Mise en cache Web

3.1 Introduction

La mise en cache est un mécanisme fondamental dans l'optimisation des performances des sites Web et des applications en ligne. Elle vise à stocker temporairement des copies de données fréquemment utilisées afin de les rendre accessibles plus rapidement lors des futures demandes. Ce processus réduit les temps de chargement en évitant de télécharger à nouveau des ressources depuis le serveur à chaque requête. Par exemple, dans le cas du cache du navigateur, ce dernier peut accéder aux ressources mises en cache localement, comme les images, les fichiers CSS et JavaScript, plutôt que de les récupérer sur le serveur à chaque fois que l'utilisateur visite une page ou interagit avec le site. (Miryala 2023)

3.1.1 Différences entre la mise en cache et le stockage

La mise en cache se distingue principalement du stockage par son objectif et par sa durée de conservation des données. Tandis que le stockage est prévu pour la persistance des données à long terme, la mise en cache est éphémère et est destinée à accélérer l'accès aux données sur le court terme. Les ressources en cache sont soumises à une expiration définie par des en-têtes HTTP, garantissant que le navigateur récupère un contenu actualisé lorsque cela est nécessaire. En termes de capacité, la mise en cache peut gérer de grandes quantités de données, souvent plusieurs Go, en fonction des paramètres du navigateur et des paramètres du cache. Les méthodes de stockage, quant à elles, ont des capacités de stockage bien plus limitées. Alors que la mise en cache peut partager des données entre utilisateurs accédant aux mêmes ressources, les méthodes de stockage sont plus isolées et spécifiques au domaine ou à la session. En matière de sécurité, la mise en cache et les méthodes de stockage ont leurs propres préoccupations. La mise en cache peut poser des problèmes de confidentialité concernant les cookies tiers, car elle peut stocker des données de manière accessible entre différentes sessions et utilisateurs. En revanche, les méthodes de stockage offrent une meilleure sécurité, exceptés les cookies qui, eux, peuvent poser des problèmes de confidentialité plus importants. Ces derniers sont envoyés avec chaque requête HTTP et peuvent être utilisés pour suivre les utilisateurs à travers différentes sessions et sites Web.(Miryala 2023)

Le tableau ci-après résume ces différences en termes d'objectif, de durée de vie, de capacité de stockage, de portée, de partage de données et de sécurité. Il compare la mise en cache avec trois méthodes de stockage : le local storage, le session storage et les cookies.

Tableau 1 : Mise en cache VS méthodes de stockage

	Cache	**Local storage**	**Session storage**	**Cookies**
Objectif	Amélioration des performances en stockant les ressources fréquemment consultées.	Stockage persistant de petites données à travers les visites.	Stockage temporaire spécifique à la session, effacé à la fermeture du navigateur ou de l'onglet.	Stockage de petites quantités de données envoyées par les sites Web.
Durée de vie	De courte durée, dépend des délais d'expiration du cache.	Jusqu'à suppression manuelle.	Jusqu'à la fermeture de l'onglet du navigateur.	Selon l'expiration (TTL) définie.
Capacité de stockage	Généralement plusieurs Go, dépend du navigateur et des paramètres de cache.	10 Mo	5 Mo	4 Ko
Portée	Non limité à un domaine ; peut inclure diverses origines.	Domaine spécifique.	Domaine spécifique ; isolé à un seul onglet ou fenêtre.	Domaine spécifique.
Partage des données	Partagé entre les utilisateurs accédant aux mêmes ressources.	Partagé entre différents onglets du même domaine.	Non partagé entre onglets ou fenêtres.	Limité au domaine ; peut être de première ou tierce partie.
Sécurité	Problèmes possibles avec les cookies tiers.	Plus sécurisé car non envoyé avec chaque requête.	Plus sécurisé car non envoyé avec chaque requête.	Problèmes de confidentialité, nécessite l'accord de l'utilisateur.

(Miryala 2023)

3.1.2 Avantages et inconvénients

La mise en cache offre de nombreux avantages qui peuvent considérablement améliorer les performances des applications. Tout d'abord, elle permet de récupérer les données plus rapidement. En stockant en mémoire les données fréquemment consultées, le cache évite de devoir les récupérer systématiquement depuis la base de données. Cela réduit la latence, c'est-à-dire le délai entre la requête du client et la réponse du système, et améliore la navigation. De plus, en diminuant le nombre de demandes adressées à la

base de données, la mise en cache contribue à réduire les coûts d'exploitation et à maintenir une performance stable, même en cas de pic de trafic. L'utilisation du cache aide aussi à réduire la consommation d'énergie d'un appareil client qui n'a donc plus besoin de télécharger à plusieurs reprises les mêmes données. (ByteSizedPieces 2023)

Néanmoins, la mise en cache n'a pas que des avantages. Une des préoccupations principales est le risque de présenter des données obsolètes aux utilisateurs. En effet, comme les données sont stockées temporairement, il peut arriver qu'elles soient mises à jour dans la base de données, mais que la version périmée mise en cache soit encore affichée pendant un certain temps. Cela peut poser des problèmes dans des contextes où les informations doivent être constamment à jour, comme les sites de commerce en ligne lors des périodes de soldes où les prix des produits peuvent changer fréquemment. Un autre inconvénient est la gestion du cache. Par exemple, déterminer quand invalider une entrée dans le cache pour s'assurer que les données ne sont pas obsolètes est une tâche complexe. Si cette invalidation n'est pas gérée correctement, elle peut entraîner des incohérences dans les données affichées ou même des failles de sécurité. De plus, comme les caches utilisent de la mémoire pour stocker des données, ils peuvent rapidement consommer beaucoup d'espace, notamment si une grande quantité de données doit être mise en cache. Ainsi, les performances de l'appareil hôte peuvent être ralenties. Enfin, la mise en place et la maintenance d'un système de cache ajoutent une couche supplémentaire de complexité au développement d'une application, rendant le code plus difficile à gérer et à comprendre, surtout si des méthodes sophistiquées sont utilisées pour gérer le cache. (ByteSizedPieces 2023)

3.1.3 Classification

Il n'existe pas de méthode universelle pour classer les différents types de mise en cache. Les approches varient, allant de la distinction basée sur l'infrastructure à des catégorisations basées sur la stratégie de mise en cache. Pour ce travail, il a été choisi d'évoquer trois critères de classification des méthodes de mise en cache : le type de cache (privé ou partagé), la stratégie de mise en cache, et l'architecture sous-jacente. Bien que plusieurs méthodes seront abordées, il n'est pas possible de couvrir exhaustivement toutes les techniques existantes en raison de leur grande diversité.

Premièrement, une distinction est faisable entre le cache privé et le cache partagé :

- Cache privé : C'est un système de stockage temporaire utilisé par les navigateurs Web pour conserver localement les ressources téléchargées via http. Il est spécifique à un utilisateur unique. Les données stockées ne sont pas partagées entre différents utilisateurs ou dispositifs. Ce cache permet de charger plus rapidement les pages Web que l'utilisateur a déjà visitées, car le navigateur n'a pas besoin de redemander ces ressources au serveur, sauf si elles sont périmées ou modifiées. La navigation hors-ligne est possible pour certains contenus qui ont été préalablement mis en cache.
- Cache partagé : Il stocke des réponses pour qu'elles puissent être réutilisées par plusieurs utilisateurs. Ce type de cache est souvent mis en place par des fournisseurs d'accès à Internet ou des entreprises au sein de leur infrastructure, avec des proxies par exemple, pour optimiser les performances du réseau local. Les ressources populaires sont réutilisées plusieurs fois, allégeant la charge des serveurs d'origine. Le cache partagé trouve son utilité dans des environnements où plusieurs utilisateurs accèdent aux mêmes ressources.

(MDN 2023)

Deuxièmement, il existe différentes stratégies de mise en cache, qui dictent la gestion des données entre le cache et la source de données. Les principales stratégies de mise en cache sont les suivantes :

- Cache-aside : Cette stratégie implique que l'application est en charge du cache. Lorsqu'une demande de données est effectuée, elle commence par vérifier si les informations sont déjà présentes dans le cache. Si ce n'est pas le cas, l'application les récupère depuis la base de données puis les enregistre dans le cache pour les futures requêtes. Cette approche est simple mais elle exige qu'une attention soit portée à la « fraîcheur » des données.
- Write-Through : La stratégie write-through assure que chaque mise à jour de données est enregistrée à la fois dans le cache et à la fois dans la base de données. Cette double écriture garantit que le cache contient toujours des données actuelles, car les modifications sont répercutées immédiatement dans les deux systèmes. Toutefois, cette méthode peut ralentir les opérations d'écriture en raison du besoin de mettre à jour deux emplacements de stockage en parallèle.
- Write-Behind : Avec cette méthode, les données sont d'abord écrites dans le cache et ne sont transférées vers la base de données qu'ultérieurement. Ce procédé accélère les opérations d'écriture en permettant un enregistrement immédiat dans le cache, mais il peut entraîner des divergences de données en raison du décalage entre la mise à jour du cache et celle de la base de données.
- Read-Through : Le cache est utilisé comme la principale source d'informations. Lorsque des données sont demandées, le cache est consulté en premier lieu. Si les données ne sont pas disponibles dans le cache, elles sont alors récupérées par le cache depuis la base de données et mises en cache pour les accès futurs. Cette méthode est particulièrement bénéfique lorsque les lectures sont fréquentes et les écritures rares ou lorsque l'accès à la base de données est lent, car elle améliore la réactivité des lectures.

(Binieli 2023)

Finalement, il est également possible de classer les méthodes de mise en cache selon l'architecture sous-jacente. Voici un aperçu des catégories :

- Cache côté client : Implémenté au niveau des dispositifs des utilisateurs finaux, ce cache inclut des mécanismes tels que le cache du navigateur ou les service workers. Il est principalement utilisé pour améliorer les temps de réponse et réduire les charges réseau en stockant localement les ressources.
- Cache côté serveur : Situé sur les serveurs d'application ou de base de données, ce type de cache comprend le cache de page, le cache d'objets ou encore le cache de base de données. Il optimise les performances des serveurs en réduisant la fréquence des accès directs aux données.
- Cache distribué : Réparti sur plusieurs serveurs ou nœuds, ce cache sert aux systèmes nécessitant une haute disponibilité et une grande capacité d'extension. Il partage la charge de stockage et de traitement des données entre plusieurs instances.
- Cache proxy : Utilisé pour optimiser les performances des réseaux locaux, ce cache se trouve au niveau des proxies Web ou des reverse proxies. Il stocke et réutilise des ressources pour plusieurs utilisateurs.
- Cache CDN (Content Delivery Network) : Réparti sur un réseau de serveurs géographiquement distribués, ce cache est utilisé pour accélérer la distribution de contenu à travers le monde. Il rapproche les données des utilisateurs finaux.

(Dang 2023; Binieli 2023; Apurva 2023)

Il est important de noter que cette dernière classification par l'architecture est un parti pris pour structurer la suite de ce travail et que d'autres approches sont tout aussi valides. Chacune de ces catégories sera explorée en détail dans les sections suivantes pour mieux comprendre leur fonctionnement et leur utilité.

3.2 Cache côté client

3.2.1 Cache du navigateur

Le cache du navigateur stocke des copies locales des ressources Web, telles que les fichiers HTML, CSS et JavaScript, ainsi que les images. En utilisant les données stockées en cache, le navigateur affiche les pages plus rapidement et évite de solliciter le serveur à chaque demande. Pour les utilisateurs qui visitent fréquemment les mêmes sites Web, le cache fluidifie la navigation. (MDN 2023)

Le fonctionnement du cache du navigateur repose sur la gestion des en-têtes HTTP qui dictent comment et combien de temps les ressources doivent être conservées. Par exemple, les en-têtes comme *Cache-Control*, *Expires*, et *ETag* sont utilisés pour contrôler la durée pendant laquelle une ressource peut rester dans le cache avant qu'une nouvelle demande soit nécessaire. (MDN 2023)

L'en-tête *Cache-Control* définit des directives telles que *max-age*, qui indique le temps maximal pendant lequel une ressource est considérée comme « fraîche ». Cette directive spécifie combien de temps les données sont conservées avant de nécessiter une nouvelle validation. Le cache du navigateur peut également être configuré pour éviter le stockage de certaines informations. Par exemple, les directives comme *no-store* ou *no-cache* sont utilisées pour indiquer que les données ne doivent pas être stockées ou qu'elles doivent être validées avant d'être utilisées. Cela permet de gérer les informations sensibles ou les contenus qui changent fréquemment et qui ne doivent pas être conservés en cache. (MDN 2023)

Lorsque le cache reçoit une demande pour une ressource périmée, il peut utiliser des mécanismes de validation pour vérifier si la ressource a été modifiée sur le serveur. Les en-têtes comme *ETag* et *Last-Modified* permettent de valider les ressources en cache sans avoir à les télécharger à nouveau. L'en-tête *ETag* est un identifiant unique généré pour chaque version d'une ressource. Il agit comme un validateur fort, ce qui signifie qu'il vérifie avec précision si la version stockée en cache est toujours valide par rapport à celle présente sur le serveur. L'en-tête *Last-Modified* indique la dernière date à laquelle la ressource a été modifiée. C'est un validateur faible, car il a une précision à la seconde près, ce qui ne reflète pas forcément les modifications mineures. Si le serveur confirme que la ressource n'a pas changé, il répond avec un statut *304 Not Modified*, laissant le navigateur utiliser la version en cache et économiser de la bande passante. (MDN 2023)

3.2.2 Service workers

Le besoin en service workers est né du constat que les applications Web, même les plus performantes, peuvent être inopérantes lorsqu'il n'y a pas de connexion réseau. Les tentatives pour résoudre ce problème ont souvent échoué à offrir un contrôle suffisant sur le cache et les requêtes. Les service workers sont une solution en agissant comme des intermédiaires qui interceptent les requêtes réseau. Ils répondent avec des ressources mises en cache, autorisant ainsi les utilisateurs à accéder à certaines fonctionnalités même sans connexion Internet. (MDN 2024b)

Pour utiliser un service worker, il faut d'abord l'enregistrer auprès du navigateur via un fichier JavaScript, qui doit être servi sur HTTPS pour des raisons de sécurité. Lorsqu'un service worker est enregistré, il passe par un processus d'installation, où il remplit le cache avec des ressources nécessaires pour une utilisation hors ligne. Cette phase est suivie par l'activation, où le nouveau service worker remplace les éventuelles versions précédentes et prend le contrôle des pages ouvertes. (MDN 2024b)

Le cycle de vie du service worker est conçu pour que les mises à jour se fassent en douceur. Lors de l'activation, la nouvelle version entre en action pour les nouvelles pages et remplace l'ancienne version une fois que les pages déjà ouvertes sont rechargées ou fermées. Lorsqu'une nouvelle version est disponible, elle est installée en arrière-plan et ne devient active que lorsque l'ancienne version n'est plus utilisée. Pendant cette période, les nouvelles ressources peuvent être mises en cache, et l'ancienne version est nettoyée pour éviter les conflits et pour économiser de l'espace de stockage. (MDN 2024b)

Le service worker est essentiellement un proxy qui peut manipuler les requêtes réseau grâce à l'événement *fetch*. Lorsqu'une requête est effectuée, il l'intercepte, vérifie si la réponse est dans le cache, et, si ce n'est pas le cas, la récupère depuis le réseau. Il peut également gérer les erreurs en fournissant des réponses par défaut lorsque le réseau est inaccessible. Par conséquent, les réponses sont personnalisées en fonction de la disponibilité du réseau ou des ressources dans le cache. (MDN 2024b)

Une autre fonctionnalité est le préchargement des ressources lors de la navigation. Il autorise le téléchargement des ressources avant même qu'une requête spécifique ne soit émise, réduisant donc le temps d'attente pour les utilisateurs. Les ressources préchargées peuvent être utilisées si elles sont disponibles avant de faire appel au réseau. (MDN 2024b)

Les outils de développement modernes mettent à disposition des fonctionnalités pour tester et gérer les service workers. Par exemple, dans Chrome et Firefox, il est possible de réinitialiser les service workers et leurs caches via des options dans les paramètres ou les outils de développement. (MDN 2024b)

3.3 Cache côté serveur

3.3.1 Cache de page

Le cache de page consiste à stocker des versions complètes des pages Web générées sur le serveur. Lorsqu'une page est demandée, le serveur peut servir directement cette page depuis le cache. Ce type de cache est principalement utile pour les sites avec du contenu statique ou peu modifié et réduit considérablement la charge sur le serveur. Par exemple, un site Web très fréquenté peut utiliser le cache de page pour servir les pages HTML pré-construites à une large audience. (Apurva 2023)

3.3.2 Cache d’objets

La mise en cache d'objets se concentre sur le stockage d'objets individuels ou d'entités complexes souvent utilisés par les applications Web. Contrairement au cache de page, qui conserve des pages Web entières, le cache d'objets stocke des éléments tels que les résultats de requêtes de base de données ou les réponses d'API. (Apurva 2023)

3.3.3 Cache de base de données

Le cache de base de données est destiné à stocker les résultats des requêtes fréquemment exécutées pour diminuer les accès directs à la base de données. Il peut aussi contenir des tables entières ou simplement des parties de ces tables. Les résultats stockés dans le cache sont récupérés bien plus rapidement que si la base de données devait être interrogée de nouveau et la répétition des requêtes coûteuses est évitée. (Apurva 2023)

3.3.4 Cache en mémoire

Le cache en mémoire stocke les données directement dans la mémoire vive (RAM) du serveur, fournissant un accès extrêmement rapide par rapport aux solutions de stockage basées sur disque. Il est idéal pour les systèmes nécessitant un accès très rapide aux données, comme les serveurs Web et les bases de données haute performance. (Binieli 2023)

Cependant, la capacité de stockage est limitée par la quantité de RAM disponible sur le serveur, restreignant donc le volume de données pouvant être mis en cache. De plus, les données stockées dans la RAM ne sont pas persistantes, elles peuvent être perdues en cas de panne du système ou lors du redémarrage. Le coût de la RAM est aussi plus élevé que celui des solutions de stockage sur disque. Pour les ensembles de données particulièrement grands, le cache peut vite devenir cher. (Dang 2023)

3.4 Cache distribué

Le cache distribué est une technique de mise en cache qui stocke et gère des données de façon décentralisée. Les données ne sont pas stockées de manière uniforme sur tous les serveurs. Au lieu de cela, elles sont réparties entre plusieurs serveurs dans le réseau, leur permettant ainsi de se partager la charge de stockage et de traitement des données. Les volumes élevés de données et les demandes simultanées sont mieux traités. (Dang 2023)

Pour ce faire, des stratégies de partitionnement des données sont mises en œuvre. Le partitionnement favorise la distribution des données de manière équilibrée sur les différents serveurs, évitant la surcharge d'un serveur unique. Par exemple, des

algorithmes de hachage sont utilisés pour assurer cette distribution uniforme. Ils minimisent les mouvements de données lorsque de nouveaux serveurs sont ajoutés ou retirés du réseau. (Redis 2024a)

Dans un tel environnement, chaque serveur conserve une partie des données mises en cache, et le routage des requêtes vers le serveur approprié est géré pour garantir que les données sont récupérées efficacement sans nécessiter une consultation de chaque serveur à chaque demande. (Binieli 2023)

Les systèmes de cache distribué sont résilients et tolérants aux pannes. Si un serveur rencontre un problème ou devient indisponible pour une quelconque raison, les données sont encore accessibles à partir des autres serveurs où elles sont répliquées. La disponibilité augmente, le risque de perte de données est réduit et la continuité du service est assurée même en cas de défaillance d'un ou plusieurs serveurs. (Dang 2023)

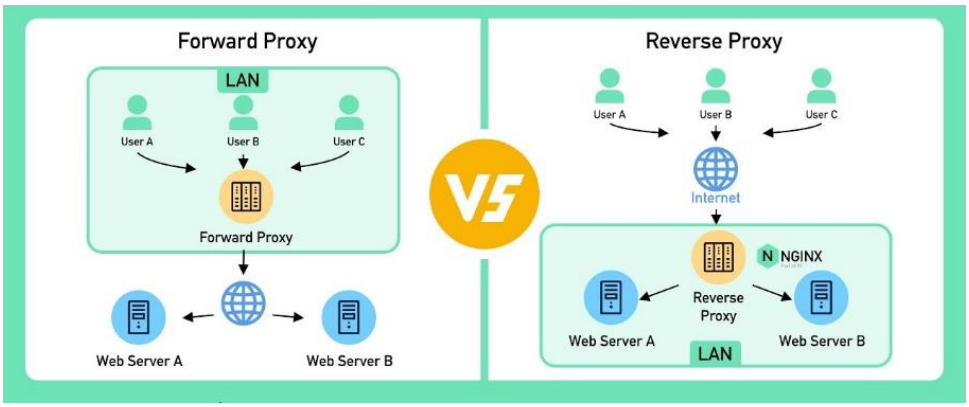
Pourtant, cette approche peut générer une surcharge réseau, car la transmission des données entre les différents nœuds du réseau augmente le trafic. Elle introduit également une gestion et une synchronisation complexes des données entre les serveurs. Gérer un cache distribué nécessite de résoudre des problèmes comme la cohérence des données, le partitionnement et la réplication. (Dang 2023)

Pour garantir que toutes les copies des données restent synchronisées et cohérentes, des mécanismes de réplication et de synchronisation sont mis en place. Par exemple, lorsqu'une donnée est mise à jour dans l'un des serveurs, ces modifications doivent être propagées aux autres serveurs pour éviter les incohérences. Certains systèmes utilisent des protocoles de cohérence et d'intégrité pour maintenir la synchronisation. (Dang 2023)

3.5 Cache proxy

Il existe deux principaux types de proxies qui jouent des rôles distincts : le proxy direct et le proxy inverse. Leur position dans l'infrastructure réseau diffère, tout comme leurs fonctions principales. Tandis que le proxy direct se situe entre les clients d'un réseau et Internet, le proxy inverse se place entre les serveurs d'un site Web et les utilisateurs finaux. (ByteByteGo 2022)

Figure 4 : Proxy direct VS Proxy inverse



(ByteByteGo 2022)

Le schéma met en évidence les deux types de proxies dans l'architecture réseau, bien que leurs différences puissent être subtiles. Les rectangles « LAN » délimitent les réseaux locaux. À gauche, le proxy direct est positionné entre les clients et les serveurs externes, interceptant les requêtes sortantes des utilisateurs. Il se trouve sur le même réseau que les clients. À droite, le proxy inverse se situe sur le même réseau que les serveurs, interceptant les requêtes entrantes pour les distribuer efficacement aux serveurs d'origine.

Le proxy direct est utilisé par des clients au sein d'un même réseau pour interagir avec des serveurs externes. Ce type de proxy intercepte les requêtes des clients, les relaie aux serveurs externes, puis renvoie les réponses des serveurs aux clients. Il permet de centraliser et de contrôler l'accès des utilisateurs à Internet. (ByteByteGo 2022)

L'un des avantages du proxy direct est sa capacité à masquer l'adresse IP des clients en utilisant celle du proxy, protégeant donc l'identité des utilisateurs sur Internet. De plus, il laisse la possibilité d'accéder à des sites normalement bloqués en contournant les restrictions d'accès imposées par certains réseaux ou gouvernements. De la même façon il peut être utilisé pour mettre en place des restrictions d'accès, filtrant par exemple l'accès à certains sites ou contenus en fonction des politiques de l'entreprise ou de l'organisation. (ByteByteGo 2022)

À l'opposé, le proxy inverse se situe devant les serveurs Web, interceptant les requêtes des clients avant qu'elles n'atteignent les serveurs d'origine. Ce type de proxy distribue le trafic entrant de manière efficace et protège les serveurs d'origine. (ByteByteGo 2022)

Grâce au proxy inverse, l'adresse IP des serveurs est masquée. Il devient plus difficile pour les attaquants de cibler directement ces serveurs avec des cyberattaques telles que les attaques par déni de service (DDoS), ayant pour but d'immobiliser un service.

En outre, il équilibre la charge de trafic entre plusieurs serveurs, ce qui est essentiel pour les sites à fort trafic. Enfin, il améliore la disponibilité et la rapidité des services en redirigeant les requêtes vers les serveurs les plus proches géographiquement des utilisateurs. (ByteByteGo 2022)

La mise en cache est une fonctionnalité généralement présente dans les proxies inverses. En stockant les réponses des serveurs d'origine dans un cache local, le proxy inverse peut répondre plus rapidement aux requêtes des utilisateurs en leur servant des versions mises en cache des ressources. La mise en cache est moins courante dans les proxies directs, car ces derniers sont principalement orientés vers la gestion des requêtes des clients plutôt que l'optimisation des performances des serveurs. (IBM 2024)

3.6 Cache CDN

Les CDN (Content Delivery Network) mettent en cache du contenu, comme des images, des vidéos ou des pages Web, dans plusieurs serveurs proxy situés plus près des utilisateurs finaux que les serveurs d'origine. Cette proximité géographique, couplée à la mise en cache, réduit la latence et accélère la durée de chargement des pages Web. (Cloudflare 2024)

Lorsqu'un utilisateur demande du contenu via un CDN, ce dernier récupère d'abord ce contenu à partir du serveur d'origine, puis, il enregistre une copie pour les requêtes ultérieures. Cette méthode assure que le contenu est disponible rapidement pour les utilisateurs suivants, sans accéder constamment au serveur d'origine. (Cloudflare 2024)

Les serveurs de cache du CDN sont répartis dans des datacenters situés à divers endroits à travers le monde. Cela signifie qu'un utilisateur à Paris et un autre à Sydney, par exemple, peuvent accéder au même contenu depuis des serveurs différents, situés à proximité de leur emplacement. De cette façon, la distance physique entre l'utilisateur et le serveur qui fournit les données est réduite. (Cloudflare 2024)

Les données mises en cache par un CDN restent dans le cache pendant une durée déterminée par un paramètre appelé *Time to live* (TTL), qui est inclus dans l'en-tête HTTP des réponses des serveurs d'origine. Il indique combien de temps le contenu doit être stocké avant d'être rafraîchi ou supprimé. En fixant un TTL approprié, le CDN évite de solliciter continuellement le serveur d'origine pour des données qui n'ont pas changé et qui sont encore valides. Lorsque le TTL expire ou si le contenu n'est pas demandé pendant un certain temps, le cache peut être purgé ou mis à jour. (Cloudflare 2024; 2023)

4. Analyse et évaluations des méthodes de mise en cache

4.1 Critères d'évaluation

Pour évaluer les différentes méthodes de mise en cache Web, il faut commencer par définir des critères pour comparer les solutions de manière objective et exhaustive. Dans le cadre de cette analyse, je me concentrerai sur deux critères principaux : la performance et la sécurité. Ces critères ont été choisis en raison de leur impact significatif sur l'expérience utilisateur et pour la fiabilité des systèmes Web. Bien que deux autres critères, la scalabilité et la facilité de mise en œuvre, aient été initialement considérés, j'ai décidé de me pencher uniquement sur deux critères pour garantir une analyse plus précise.

La performance est essentielle pour assurer une expérience utilisateur optimale. Un accès rapide aux données stockées en cache réduit significativement le temps de réponse des requêtes. Une bonne performance contribue à des interactions plus fluides et renforce la satisfaction et la fidélisation des utilisateurs.

Les méthodes de mise en cache doivent également être évaluées pour leur capacité à gérer les vulnérabilités connues, à intégrer des pratiques de sécurité recommandées, et à réagir efficacement en cas de faille de sécurité. La complexité de la configuration de la sécurité, ainsi que les exigences spécifiques de l'environnement, peuvent influencer la sécurité globale des systèmes de cache.

Ainsi, la performance et la sécurité ont été retenues comme critères principaux pour plusieurs raisons. Tout d'abord, l'impact direct sur l'utilisateur est évident. Des temps de réponse rapides et une latence minimale sont des facteurs déterminants pour offrir une bonne expérience utilisateur. Ensuite, la protection des données est une composante importante dans le contexte actuel, où les cyberattaques sont de plus en plus fréquentes et sophistiquées. Assurer la confidentialité et l'intégrité des données mises en cache est indispensable pour maintenir la confiance des utilisateurs et se conformer aux réglementations en vigueur.

4.2 Technologies à comparer

Je procéderai à une analyse en comparant d'une part les différentes catégories entre elles, et d'autre part, en examinant les solutions au sein de chaque catégorie. Cette démarche permettra d'évaluer à la fois les distinctions entre les types de cache et à la fois les différences au sein de chacun dans le but de fournir une vue complète sur leurs performances et leurs spécificités.

Concernant les comparaisons au sein de chaque catégorie de cache, je commencerai par comparer le cache du navigateur et les service workers, deux technologies du cache côté client. Ces solutions stockent les ressources localement, sur l'appareil de l'utilisateur. Le cache du navigateur gère automatiquement les éléments fréquents, tandis que les service workers permettent un contrôle plus fin ainsi que la possibilité de fonctionner hors ligne.

Pour les caches côté serveur, j'analyserai Redis, Memcached, et Apache avec *mod_cache*. Ces technologies de mise en cache stockent les données fréquemment consultées dans la mémoire du serveur. Redis et Memcached sont tous deux connus pour leur rapidité, mais Redis se distingue par des fonctionnalités supplémentaires, telles que la persistance des données et le support de structures de données avancées. Apache avec *mod_cache*, quant à lui, intègre directement la mise en cache dans le serveur HTTP.

Dans le cadre du cache distribué, je comparerai Memcached, Redis, Hazelcast, et Apache Ignite. Ces technologies sont conçues pour fonctionner dans des environnements où les données doivent être accessibles rapidement à travers plusieurs nœuds. Memcached et Redis se concentrent sur la mise en cache rapide en mémoire, alors que Hazelcast et Apache Ignite apportent des capacités avancées dans le traitement distribué et dans la gestion de données à grande échelle.

Dans la catégorie des caches proxy, Nginx et Varnish seront comparés. Ces serveurs proxy sont utilisés pour améliorer la vitesse de livraison des contenus Web en mettant en cache les réponses HTTP. Nginx est reconnu pour sa capacité à gérer un grand nombre de connexions simultanées. Quant à Varnish, il est spécifiquement optimisé pour la mise en cache de contenu Web et possède des options de configuration supplémentaires.

Pour finir, j'examinerai Cloudflare et Amazon CloudFront, deux plateformes bien connues et utilisées dans le domaine des caches CDN. Ces réseaux de distribution de contenu diffusent du contenu Web de manière efficace à travers le monde entier. Cloudflare se distingue par ses fonctionnalités de sécurité, comme la protection contre les attaques DDoS. De l'autre côté, CloudFront s'intègre étroitement avec les divers services AWS (Amazon Web Services) pour une gestion optimisée des contenus, tout en tirant parti de l'écosystème d'Amazon.

4.3 Méthode

J’évaluerai chaque technologie de mise en cache en attribuant une note sur 10 pour chaque critère. Cette analyse est purement théorique et peut contenir une certaine subjectivité, car elle reflète mon interprétation des caractéristiques techniques de chaque solution. Elle est basée sur de nombreuses recherches effectuées sur des sites comparatifs, des forums et des documents officiels. C’est pourquoi l’évaluation repose principalement sur l’impression que j’ai pu tirer au fil de mes lectures et non sur des données concrètes ou des mesures d’analyse.

La performance sera évaluée en fonction de la rapidité et de l'efficacité de la mise en cache, ainsi que de l'impact sur les temps de réponse des systèmes Web. La sécurité sera évaluée en tenant compte de la capacité de chaque solution à protéger les données mises en cache, notamment à travers le chiffrement, la gestion des accès, la résistance aux vulnérabilités et la difficulté de développement.

Après l'évaluation, j’expliquerai les raisons de chaque note attribuée. Puis, je formulerai des recommandations basées sur ces évaluations pour différents contextes d'utilisation, permettant ainsi de guider le choix des technologies de mise en cache en fonction des besoins spécifiques.

4.4 Analyse comparative globale

4.4.1 Évaluation

Tableau 2 : Évaluation globale des technologies de mise en cache

	Performance /10	**Sécurité /10**	**Total /20**
Cache côté client	7	6	13
Cache côté serveur	8	7	15
Cache distribué	8	8	16
Cache proxy	8	8	16
Cache CDN	9	9	18

4.4.2 Explications et recommandations

Le cache côté client est efficace pour stocker des ressources statiques comme les images et les fichiers CSS, HTML et JavaScript. La performance est assez bonne grâce à la proximité des données, directement sur l'appareil de l'utilisateur. Toutefois, elle est limitée par la capacité de stockage du navigateur et par la nature des contenus, qui sont généralement moins volumineux et moins fréquemment mis à jour. La sécurité des caches côté client est relativement faible. Comme les données sont stockées sur l'appareil de l'utilisateur, il n’est pas possible pour le serveur de contrôler le cache de

l’utilisateur et les données sont plus vulnérables aux attaques. De plus, le contrôle de la mise à jour et de l'invalidation des données est plus complexe, rendant l’intégrité et la sécurité des informations difficiles à gérer. (d’Arros 2024)

Le cache côté serveur possède une performance plus élevée en stockant les réponses HTTP et d'autres données sur le serveur. Ce type de cache est particulièrement efficace pour les contenus dynamiques et volumineux, qui nécessitent un accès rapide directement depuis le serveur. Il permet une gestion centralisée qui améliore l’intégrité des données et facilite les mises à jour. La performance peut varier en fonction de la proximité du serveur par rapport au client, mais en général, elle est supérieure à celle du cache côté client pour les grandes quantités de données. La sécurité du cache côté serveur est plus élevée que celle du cache côté client, car elle permet un contrôle centralisé des accès et des politiques de sécurité. Cette sécurité peut être compromise si le serveur lui-même est vulnérable. (d’Arros 2024)

Les caches distribués ont une très bonne performance en favorisant un accès rapide aux données à grande échelle, et en partageant la charge sur plusieurs serveurs (Dang 2023). Les caches distribués bénéficient d'une meilleure résilience en matière de sécurité par rapport à la mise en cache sur un unique serveur. La répartition des données et la redondance des serveurs limitent l'impact d'une faille de sécurité. Cependant, cette résilience ne compense pas entièrement les défis liés à la configuration complexe et à la gestion de la sécurité à travers plusieurs nœuds. Cela les rend légèrement plus vulnérables.

Le cache proxy est performant pour la mise en cache de contenus Web dans la mesure où il réduit la charge sur les serveurs d'origine. Les caches proxy sont un choix idéal pour les sites à fort trafic grâce à leur particularité à gérer de nombreuses connexions simultanées. Les proxies offrent une bonne sécurité, notamment en masquant les adresses IP des serveurs d'origine et complique donc les attaques directes comme les attaques par DDoS. (ByteByteGo 2022)

La performance des réseaux de distribution de contenu (CDN) est presque toujours optimale, même sous une forte charge, grâce à leur scalabilité intrinsèque. Les CDN proposent des fonctionnalités de sécurité avancées, incluant le contrôle des accès et la protection contre les attaques DDoS, entre autres. (AWS 2024)

Le cache côté client convient particulièrement aux sites Web qui nécessitent un accès rapide aux ressources statiques, en raison de la proximité entre les données et l’utilisateur. Toutefois, il présente des vulnérabilités en matière de sécurité et doit être

complété par d'autres solutions plus robustes pour des applications nécessitant une meilleure sécurité. En revanche, le cache côté serveur surpasse le cache côté client en termes de gestion de données dynamiques et volumineuses. Ce type de cache est recommandé pour les entreprises et applications nécessitant une gestion efficace de la sécurité et de la performance pour des données plus complexes. Les caches distribués s'adressent plutôt aux environnements ayant besoin de garantir que les données sont toujours accessibles, même en cas de forte demande ou de panne. Leur capacité à répartir la charge sur plusieurs serveurs permet d'obtenir une excellente performance, mais au prix d'une gestion et d'une sécurité complexes. Ces caches sont adaptés aux grandes entreprises souhaitant bénéficier d'une infrastructure solide mais qui, en contrepartie, doivent porter une attention particulière sur la sécurité des données. Le cache proxy, quant à lui, est idéal pour les sites Web à fort trafic où la gestion simultanée de nombreuses connexions est importante, avec des besoins modérés en sécurité. Enfin, les CDN sont les solutions les plus performantes et sécurisées, parfaites pour les entreprises qui doivent diffuser du contenu à une audience mondiale. Leur scalabilité et leur capacité à réduire la latence de manière drastique en font un choix idéal pour les sites de commerce électronique et les plateformes de médias, entre autres (AWS 2024).

4.5 Analyse comparative par catégorie

4.5.1 Cache côté client

4.5.1.1 Évaluation

Tableau 3 : Évaluation de technologies de cache côté client

	Performance /10	**Sécurité /10**	**Total /20**
Cache du navigateur	7	6	13
Service workers	8	8	16

4.5.1.2 Explications et recommandations

La note en performance du cache du navigateur est justifiée par le fait que ce cache réduit considérablement la latence. Néanmoins, il reste limité par la capacité de stockage du navigateur et par la gestion des données, qui peuvent parfois devenir obsolètes si elles ne sont pas correctement invalidées. En ce qui concerne la sécurité, la note de 6/10 reflète les vulnérabilités inhérentes au fait que les données sont stockées localement sur l'appareil de l'utilisateur. Même si les navigateurs modernes ont des mécanismes pour protéger ces données, elles restent accessibles et donc potentiellement exploitables par des attaques locales en cas de mauvaises configurations.

En fonctionnant comme des scripts qui s'exécutent en arrière-plan et qui interceptent les requêtes réseau, les service workers sont un peu plus performants que le simple cache du navigateur. Ils peuvent accéder aux ressources en mode hors ligne et peuvent gérer les mises à jour des données en arrière-plan sans perturber la navigation de l'utilisateur. Côté sécurité, leur enregistrement se fait uniquement via HTTPS pour assurer le chiffrement des communications entre le client et le serveur. De plus, il est possible de limiter l'accès uniquement aux ressources nécessaires, d'implémenter des scripts de validation qui garantissent l'authenticité des service workers, ou encore d'utiliser des mécanismes ayant pour rôle de récupérer les mises à jour de façon sécurisée. La note en sécurité reflète une bonne base qui, pour être améliorée, doit être configurée par un développeur. (Zee Palm 2024)

Le cache du navigateur est recommandé pour les sites qui nécessitent un chargement rapide de ressources statiques et qui visent à offrir une meilleure expérience utilisateur lors des visites récurrentes. Il convient aux sites avec un contenu relativement stable, où les risques de compromission de sécurité sont limités, ou pour des ressources qui ne contiennent pas de données sensibles. Les service workers sont particulièrement adaptés aux Progressive Web Apps (PWA) ou simplement à des sites fournissant des fonctionnalités hors ligne (Zee Palm 2024).

4.5.2 Cache côté serveur

4.5.2.1 Évaluation

Tableau 4 : Évaluation de technologies de cache côté serveur

	Performance /10	**Sécurité /10**	**Total /20**
Redis	9	8	17
Memcached	8	7	15
Apache avec *mod_cache*	7	6	13

4.5.2.2 Explications et recommandations

Redis obtient une note élevée en performance en raison de sa capacité à gérer des opérations en mémoire avec une latence extrêmement faible (en dessous de la milliseconde). Il est aussi polyvalent, car il prend en charge plusieurs structures de données, comme les listes, les ensembles ou les tableaux associatifs. Redis met également en place des moyens pour persister les données en dehors de la mémoire vive, dans le disque dur, contrairement à Memcached qui se limite à la mémoire vive. Concernant la sécurité, Redis possède un mode protégé qui réduit les risques liés à des configurations incorrectes en limitant l'accès aux seules interfaces de confiance. Il offre

également des mécanismes d'authentification via des ACL (liste de contrôle d'accès) et la possibilité de désactiver ou renommer certaines instructions critiques pour limiter les actions non autorisées. Malgré cela, Redis est plutôt conçu pour des environnements de confiance, et son exposition directe à des réseaux non sécurisés peut entraîner des vulnérabilités significatives. (ScaleGrid 2024; Redis 2024b)

Memcached obtient une note légèrement inférieure en raison de sa spécialisation dans les opérations de cache basiques, mais il se distingue par sa simplicité et par sa performance. Memcached est un système de cache en mémoire basé sur un modèle clé-valeur, où les valeurs sont des chaînes de caractères. Il est optimisé pour des opérations de lecture et d'écriture rapides, avec une faible latence, typiquement inférieure à celle de Redis lorsqu'il s'agit de gérer des ensembles de données simples. Par contre, il ne dispose pas des fonctionnalités de persistance présentes dans Redis. Il est possible de sécuriser l'utilisation de Memcached mais cela doit se faire manuellement car il n'y a pas de solutions directement intégrées. (ScaleGrid 2024)

Apache HTTP Server a des performances relativement bonnes, lorsqu'il est configuré avec le module *mod_cache*. Il s'agit du module d'Apache qui permet la mise en cache sur le serveur. Bien que *mod_cache* soit puissant, sa performance dépend fortement d'une configuration adéquate, du bon choix de cache (en mémoire ou sur disque), ainsi que de la gestion correcte des règles de cache via le fichier « .htaccess ». La note pour la sécurité d'Apache avec *mod_cache* reflète plusieurs vulnérabilités. En utilisant *mod_cache*, une fois que contenu est mis en cache, il est servi sans vérification des autorisations d'accès et peut exposer des informations sensibles à des utilisateurs non autorisés. De plus, si le serveur est compromis, il devient facile d'altérer ou de manipuler le contenu du cache et le seul moyen d'atténuer ce risque est de veiller à toujours effectuer les mises à jour d'Apache. (LoadForge 2023; Apache 2024)

Redis excelle dans les situations où des structures de données complexes sont requises, comme le stockage de sessions utilisateur, les analyses en temps réel et la gestion de systèmes de messagerie sophistiqués. Sa capacité à manipuler une variété de types de données et à persister les informations le rend particulièrement adapté aux applications mobiles qui exploitent des index géospatiaux, par exemple. En revanche, Memcached est plus adapté pour des scénarios où la simplicité et la rapidité sont primordiales. Il est idéal pour des besoins de mise en cache basiques où les données sont principalement des chaînes de caractères. Memcached est un choix judicieux pour les applications peu complexes, avec un cache simple et où la persistance des informations n'est pas impérative. (ScaleGrid 2024)

Apache avec *mod_cache*, quant à lui, est surtout conseillé lorsqu'un serveur Web Apache est déjà présent dans l'infrastructure, afin d'améliorer les performances en ajoutant simplement un module de mise en cache.

4.5.3 Cache distribué

4.5.3.1 Évaluation

Tableau 5 : Évaluation de technologies de cache distribué

	Performance /10	Sécurité /10	Total /20
Redis	9	7	16
Memcached	8	6	14
Hazelcast	8	8	16
Apache Ignite	9	8	17

4.5.3.2 Explications et recommandations

Pour Redis et Memcached, les notes en performance restent identiques à celles du cache sur serveur unique. Redis, en mode cluster, maintient des performances élevées dans un environnement distribué, et Memcached, créé à l'origine pour le cache distribué, fonctionne donc très bien dans cette configuration. Néanmoins, en termes de sécurité, un point a été retiré. Même si la distribution des données sur plusieurs nœuds réduit le risque de point unique de défaillance, la complexité de la configuration manuelle des règles de sécurité justifient cette déduction.

Hazelcast offre de plutôt bonnes performances. Il fonctionne en mémoire, il peut gérer des structures de données complexes et une grande scalabilité. Cependant, la dépendance à la Java Virtual Machine (JVM) peut entraîner une utilisation élevée de la mémoire et une surcharge lors du démarrage et de l'exécution. De plus, malgré l'option de persistance sur disque proposée par Hazelcast, celle-ci n'est pas aussi performante que les solutions dédiées. Hazelcast possède de bonnes fonctionnalités de sécurité, particulièrement dans son édition *Enterprise*. Pour la confidentialité des données, il propose des communications chiffrées via SSL/TLS, avec des options de configuration. Par ailleurs, il propose des mécanismes pour l'authentification, afin de vérifier l'identité des utilisateurs et des systèmes de manière flexible, ainsi que pour l'autorisation, en contrôlant les accès et les actions au sein du cluster. (Moeinnia 2023; Hazelcast 2024)

Apache Ignite, quant à lui, se distingue par de très bonnes performances, grâce à son architecture en mémoire et son stockage multi-niveaux, qui combine la rapidité de la mémoire vive avec la persistance des bases de données externes. En plus de cela, Ignite prend en charge l'exécution de requêtes SQL complexes. En matière de sécurité,

Apache Ignite intègre des fonctionnalités d'authentification pour contrôler l'accès des nœuds et des clients au cluster. Il permet aussi de définir des plugins de sécurité personnalisés pour une gestion des droits d'accès et de l'autorisation. Enfin, pour sécuriser les communications, il prend en charge des connexions chiffrées via SSL/TLS. (Moeinnia 2023; Gajbhiye 2020)

Redis Cluster est recommandé pour des applications nécessitant une faible latence et une scalabilité horizontale et Memcached convient bien pour des caches simples et efficaces, comme le stockage de sessions Web. Toutefois, ces deux technologies sont restreintes en termes de sécurité si aucune configuration n'est mise en place. Hazelcast est adapté pour des applications exigeant une grande disponibilité et un traitement en mémoire complexe, comme les systèmes de transactions en temps réel. Apache Ignite excelle dans les environnements nécessitant des fonctionnalités comme le traitement en temps réel et les requêtes SQL complexes, tout en nécessitant une configuration minutieuse en raison de la complexité de certaines fonctionnalités.

4.5.4 Cache proxy

4.5.4.1 Évaluation

Tableau 6 : Évaluation de technologies de cache proxy

	Performance /10	Sécurité /10	Total /20
Nginx	9	8	17
Varnish	8	7	15

4.5.4.2 Explications et recommandations

Nginx est à l'origine un serveur Web mais possède aussi des capacités de reverse proxy et de mise en cache. Il traite rapidement les requêtes grâce à son architecture asynchrone, qui gère plusieurs connexions simultanément. Sa rapidité le rend tolérant aux situations de fort trafic. En matière de sécurité, il intègre plusieurs fonctionnalités dont le traitement des connexions SSL/TLS. Contrairement à Varnish, qui est exclusivement un reverse proxy, Nginx combine ces fonctions avec celles d'un serveur Web complet pour fournir une performance et une sécurité efficaces. (Garland 2021)

Varnish a été créé spécifiquement comme un accélérateur HTTP, ce qui lui permet de se concentrer pleinement sur cette tâche et de l'exécuter de façon efficace. Il possède une fonctionnalité pour accéder à des données mises en cache même après leur expiration, ce qui est particulièrement utile lors de maintenances ou de pannes. Côté sécurité, Varnish fournit également des outils de protection intégrés, bien qu'il ne gère pas le traitement des connexions SSL/TLS, à la différence de Nginx. (Garland 2021)

Pour les sites nécessitant une gestion du cache flexible, personnalisable et conviviale, il est judicieux de choisir Varnish. En revanche, si la vitesse et la stabilité sont davantage recherchées, il est préférable de se tourner vers Nginx. Il est également important de noter que Nginx peut servir de solution complète en tant que serveur Web, alors que Varnish, n'étant pas autonome, doit être utilisé avec un serveur Web. (Garland 2021)

4.5.5 Cache CDN

4.5.5.1 Évaluation

Tableau 7 : Évaluation de technologies de cache CDN

	Performance /10	**Sécurité /10**	**Total /20**
Cloudflare	10	10	20
Amazon CloudFront	9	9	18

4.5.5.2 Explications et recommandations

Cloudflare est réputé pour être très performant et très sécurisé. Il possède un vaste réseau de centres de données répartis à l'échelle mondiale. Ces points de présence réduisent considérablement la latence pour les utilisateurs du monde entier. Le service utilise des techniques de multiplexage, ce qui signifie qu'il peut traiter plusieurs requêtes simultanément. Cloudflare se concentre principalement sur la sécurité. Il propose une suite complète de fonctionnalités de sécurité, comme une protection contre les attaques DDoS, un pare-feu d'application Web et des certificats SSL. (Hung 2021)

Amazon CloudFront possède également un large réseau de points de présence. CloudFront utilise des *buckets S3*, qui sont des espaces de stockage en ligne (dans le Cloud) où les fichiers sont conservés. Lorsqu'un utilisateur demande un fichier, CloudFront le récupère depuis ces *buckets S3* et le met en cache sur ses serveurs proches des utilisateurs. De façon générale, CloudFront est considéré comme légèrement inférieur à Cloudflare. Il est sérieux en termes de sécurité, en fournissant, entre autres, des certificats SSL. Néanmoins, il ne propose pas de protection intégrée contre les attaques DDoS comme Cloudflare. Les fonctionnalités de sécurité de CloudFront sont solides mais moins complètes en comparaison avec celles de Cloudflare. (Hung 2021)

Cloudflare convient bien aux environnements nécessitant un contrôle élevé sur les options de mise en cache et une protection renforcée contre les menaces de sécurité. D'autre part, CloudFront est parfait dans des contextes spécifiques, comme la diffusion de vidéos en streaming (Hung 2021). Il peut être facilement intégré dans les infrastructures utilisant déjà les services AWS.

5. Mise en pratique

Passer de la théorie à la pratique est nécessaire pour évaluer concrètement l'efficacité des méthodes de mise en cache discutées dans la section précédente. Cette partie se concentre sur l'implémentation concrète des concepts analysés en construisant une application spécifique permettant de tester et de valider les approches de mise en cache dans un environnement local et contrôlé. De cette façon, leur impact direct sur la performance est observable et des données empiriques peuvent être recueillies. Cette démarche viendra compléter l'évaluation théorique par des résultats tangibles, enrichissant la compréhension des méthodes de mise en cache.

5.1 Construction de l'application

L'idée de ce mini-projet est de créer une application d'e-commerce axée sur les vêtements, qui exploite des données fictives suffisantes pour simuler un site de taille moyenne. L'objectif principal est de mettre en place une interface affichant une liste de vêtements classés par genre (homme ou femme) et par catégorie (pantalon, robe, etc.), tout en autorisant la sélection de filtres sur cette liste. L'application intègre à la fois des données dynamiques, qui nécessitent des appels au serveur à chaque requête, et des données statiques. Le code source de l'application est disponible sur GitHub via le lien ci-après : https://github.com/Pamouu/TravailDeBachelor.

MySQL a été choisi comme base de données et Vue.js pour le développement du frontend. J'ai déjà eu l'occasion d'utiliser ces technologies par le passé, ce qui a facilité et accéléré le développement. Bootstrap a également été intégré pour améliorer l'interface utilisateur. Concernant le backend, j'ai sélectionné Node.js et Express. Node.js permet l'exécution de JavaScript côté serveur. Express, quant à lui, est construit sur Node.js et simplifie le développement en offrant un certain nombre de fonctionnalités.

La réalisation a débuté par la récupération de données depuis l'API de l'entreprise ASOS, via le site rapidapi.com. Au total, 770 produits au format JSON ont pu être recueillis. Ils ont ensuite été insérées, grâce à un script, dans la base de données MySQL, préalablement initialisée. La structure de la base de données comprend six tables, avec la table principale des produits reliée à d'autres informations (marque, couleur, taille et catégorie).

Pour le backend, une API a été créée pour interagir avec la base de données. Elle est organisée autour de trois fichiers : le fichier de routes, le contrôleur et le modèle. Le fichier de routes définit les URLs accessibles, le contrôleur gère la logique métier et la réponse aux requêtes, tandis que le modèle interagit avec la base de données.

Le développement du frontend a impliqué la création de quelques composants essentiels. Il s'agit de la barre de navigation, la page d'accueil, la liste des produits et le détail d'un produit. Étant donné que l'application est conçue principalement à des fins de tests, seules ces pages ont été développées. Les autres sections typiques d'un site d'e-commerce, comme les FAQs, les conditions générales de vente ou le profil utilisateur, n'étaient pas utiles dans ce contexte. Le composant avec le plus de fonctionnalités est celui affichant la liste des produits. Il nécessite l'intégration des filtres, des tris et de la pagination, et a entraîné des adaptations du backend pour répondre à ces requêtes spécifiques.

Malgré le fait que la mise en cache soit le mécanisme clé testé dans ce mini-projet, il reste important d'adopter des pratiques visant à optimiser les appels à la base de données. Par exemple, un système de pagination a été mis en place pour charger un nombre maximum de produits à la fois, soit 16 produits. Le catalogue complet n'a ainsi pas besoin d'être téléchargé à chaque appel.

Quelques captures d'écran de l'application Web sont disponibles dans l'annexe 1 pour montrer le visuel et les fonctionnalités.

5.2 Mise en cache

5.2.1 Choix des technologies

Le choix des technologies de mise en cache a été guidé par les besoins spécifiques de cette application. Dans un premier temps, il n'était pas pertinent d'utiliser des technologies de cache à grande échelle comme les CDN ou le cache distribué, qui sont plus adaptés aux sites de grande envergure. Dans ce projet, il était plus logique de se concentrer sur le cache côté client et côté serveur, qui correspondent mieux aux objectifs de test et à la structure de l'application.

Concernant le cache côté client, le service worker a été préféré au cache traditionnel du navigateur. En effet, le service worker offre des fonctionnalités supplémentaires, notamment la possibilité de faire fonctionner l'application en mode hors ligne. Pour le cache côté serveur, il sera mis en place grâce à Redis. Ce choix repose sur les résultats d'analyse menés dans la section 4, où Redis a été identifié comme l'option la mieux notée pour les performances.

L'objectif est donc de tester le service worker pour le cache des données statiques (fichiers HTML, CSS, JavaScript et images), et d'utiliser Redis pour la mise en cache des requêtes dynamiques vers la base de données. Bien que le service worker soit également capable de gérer le cache des requêtes dynamiques, il est plus complexe de

gérer l'invalidation des données dans ce contexte. Par conséquent, les tests sépareront clairement ces deux approches : le service worker sera dédié aux fichiers statiques, et Redis aux données dynamiques.

5.2.2 Implémentation

La mise en place du service worker s'est révélée relativement simple. Cependant, une difficulté est apparue lors des tests : l'application comportait trop peu de contenu statique pour que le service worker puisse réellement démontrer son utilité.

Pour répondre à ce problème, l'idée a été d'ajouter le plus d'éléments statiques possible sur la page d'accueil, même des éléments sans lien direct avec le thème de l'application. Des images ont été récupérées depuis le site picsum.photos et du texte Lorem Ipsum a été généré aléatoirement sur loremipsum.io. Ensuite, pour complexifier davantage le code HTML et CSS de la page d'accueil, un prompt spécifique a été soumis à ChatGPT : *"J'aimerais changer ma homepage, avec si possible beaucoup de HTML, CSS et JS pour avoir une page lourde qui aura beaucoup de contenu statique à mettre en cache par la suite."*

Concernant l'implémentation de Redis, l'installation et la configuration pour lier Redis à l'application se sont déroulées sans problème. La méthode de cache utilisée est celle du *cache-aside*, où les données sont d'abord recherchées dans le cache. Si elles sont présentes, elles sont directement renvoyées à l'utilisateur. Si elles sont absentes, une requête est effectuée vers la base de données. Les données récupérées sont alors stockées dans Redis pour de futures requêtes, puis, elles sont renvoyées à l'utilisateur.

Toutefois, une complication est survenue lors des tests : Redis se retrouvait en conflit avec le cache du navigateur, qui est automatiquement géré par Express. Le problème résidait dans le fait que les en-têtes HTTP *ETag* et *Last-Modified* étaient automatiquement générés par Express, provoquant une mise en cache côté client non désirée. Les données étaient uniquement récupérées depuis le cache du navigateur et non depuis Redis. De ce fait, la difficulté a été de trouver le code approprié pour supprimer ces en-têtes et pour modifier les en-têtes *Cache-Control*, *Pragma* et *Expires*. Cette étape a permis d'assurer que seul Redis gérait les données dynamiques et que les résultats des tests seraient exclusivement liés à l'utilisation de Redis, sans interférence du cache du navigateur.

5.2.3 Déroulement des tests

La première étape a consisté à identifier des outils pour tester efficacement les performances de l'application. Lors des recherches théoriques, l'outil Grafana k6 a retenu mon attention, car il peut simuler un nombre donné d'utilisateurs effectuant des requêtes sur le site, en simultané et pendant une période déterminée.

Différents scénarios ont été mis en place en simulant successivement 1, 25, 50, 75, et 100 utilisateurs virtuels, une fois avec Redis activé, et une fois sans. Cette approche s'arrête à 100 utilisateurs virtuels, car l'application n'est pas destinée à des charges plus élevées. Pour des scénarios dépassant ce seuil, il serait plus pertinent d'envisager d'autres types de cache, tels que les CDN. Chaque test suivait le même schéma : le nombre d'utilisateurs était augmenté progressivement sur une période de 30 secondes, puis maintenu à son maximum pendant 1 minute, avant de revenir à 0 utilisateur. Cinq URLs différentes ont été testées, pouvant être appelées au hasard par les utilisateurs virtuels. Les requêtes incluaient des appels classiques affichant les listes des vêtements par catégorie, ainsi que des requêtes plus complexes simulant la sélection de filtres sur la marque, la couleur et la taille.

L'un des avantages de k6 est sa capacité à générer les graphiques des performances. Il permet également de comparer les données issues de deux tests différents sur un même graphique, ce qui est plutôt utile pour observer les différences avec et sans l'utilisation du cache Redis.

Cependant, k6 ne simule que les requêtes effectuées directement sur le serveur, sans prendre en compte le comportement d'un navigateur ni l'impact du cache côté client. C'est pourquoi il a été nécessaire de trouver un autre outil pour mesurer les performances liées au service worker. La solution la plus simple et efficace est Google Lighthouse, un outil intégré directement aux DevTools de Chrome. Lighthouse est également disponible en ligne de commande. Il audite plusieurs aspects d'une page Web, notamment les performances, l'accessibilité, les meilleures pratiques, et le référencement. Dans le cadre de ces tests, seule la catégorie « performance » a été utilisée.

Après avoir testé les deux méthodes – DevTools et ligne de commande – et effectué de nombreux essais, il a été constaté que les tests réalisés via les DevTools étaient plus stables. Même s'il est parfois conseillé de réaliser les tests en navigation privée, ceux effectués en navigation normale n'ont montré aucune différence notable, malgré la présence de certaines extensions du navigateur qui auraient pu potentiellement interférer avec l'audit.

Ainsi, tous les tests pour le service worker ont été réalisés via Chrome, avec un rafraîchissement complet du cache et du stockage avant chaque audit pour supprimer les données superflues et obtenir les résultats les plus précis possible. Le mode « Desktop » a été sélectionné et la case « Clear Storage » n'a pas été activée, car les anciens caches étaient vidés manuellement juste avant. Les audits ont été effectués à la suite pour maintenir des conditions de réseau similaires entre chaque test afin de comparer les performances de manière équitable. De plus, chaque test a été répété au moins cinq fois pour vérifier d'éventuelles fluctuations dans les résultats.

5.3 Résultats et discussion

5.3.1 Service worker

Tous les résultats des tests de performance avec Google Lighthouse sont disponibles dans l'annexe 2. Ils ont aussi été reportés dans le tableau de synthèse ci-dessous.

Les résultats montrent un score de 100/100 avec le service worker activé, contre un score de 93/100 sans cache, avec parfois des fluctuations à 73/100. Il est important de ne pas se focaliser uniquement sur le score total le plus élevé. Pour une analyse complète, il faut aussi s'intéresser à comment Lighthouse procède à son calcul.

Lighthouse utilise plusieurs critères pondérés pour évaluer les performances :

- First Contentful Paint (FCP) - 10% : Mesure le temps nécessaire pour que le premier élément de contenu (texte, image, etc.) soit affiché à l'écran.
- Total Blocking Time (TBT) - 30% : Évalue la durée pendant laquelle le fil d'exécution principal est bloqué, empêchant l'utilisateur d'interagir avec la page.
- Speed Index (SI) - 10% : Évalue la rapidité avec laquelle le contenu visuel d'une page se charge.
- Largest Contentful Paint (LCP) - 25% : Mesure le temps que prend le plus grand élément de contenu visible (comme une image ou un bloc de texte) à s'afficher.
- Cumulative Layout Shift (CLS) - 25% : Mesure la stabilité visuelle d'une page. Un score faible indique que les éléments de la page ne se déplacent pas de manière inattendue pendant le chargement.

(Chrome 2023)

Le tableau suivant montre les résultats obtenus dans chacune des catégories ci-dessus. Il aide à comprendre dans quels domaines spécifiques des améliorations ont eu lieu avec le service worker. Il y a deux lignes consacrées aux mesures prises sans cache, à cause de fluctuations survenues à quelques reprises.

Tableau 8 : Performances sans cache VS service worker

	FCP	TBT	SI	LCP	CLS	Score
Sans cache	0.6s	80ms	0.7s	1.7s	0.036	93
Sans cache, fluctuation	0.5s	70ms	0.7s	1.6s	0.48	73
Service worker	0.4s	40ms	0.6s	0.6s	0	100

Tout est effectivement un peu moins performant sans cache, mais les différences restent modérées dans la plupart des catégories. La différence majeure réside dans le LCP et le CLS, deux critères relativement importants dans le calcul du score.

Le LCP sans cache est de 1,7 seconde, contre seulement 0,6 seconde avec le service worker. Sans cache, il est normal que le plus grand élément de la page prenne plus de temps à charger. Le CLS sans cache fluctue, avec un score parfois de 0.48, alors qu'il est parfait (0) avec le service worker. Cette variation pourrait être due à des éléments qui se déplacent pendant le chargement des ressources non mises en cache, comme les images.

Il est également intéressant de noter que, lors de plusieurs essais, les résultats avec le service worker ont toujours été stables. En revanche, environ une fois sur quatre, les tests sans cache ont donné un score total de 73, principalement en raison de la hausse du CLS. Cette stabilité avec le service worker pourrait être attribuée au fait que, lorsque les ressources sont mises en cache localement, elles ne dépendent plus de la qualité de la connexion réseau pour être récupérées. Sans cache, chaque requête doit passer par le réseau, et toute variation dans la vitesse ou la stabilité de la connexion peut affecter le chargement des ressources.

Les scores de performance sont élevés dans les deux cas, avec ou sans cache, mais cela ne signifie pas que la mise en cache est inutile. En réalité, l'un des principaux défis rencontrés était de trouver du contenu statique à mettre en cache. Il a fallu beaucoup d'efforts pour déterminer quels éléments méritaient d'être mis en cache. C'est pourquoi le contenu mis en cache dans ce projet ne reflète probablement pas un site véritablement statique. Ces résultats sont à traiter avec prudence. Par ailleurs, des tests ont aussi été réalisés avec des images bien plus lourdes que celles présentes sur la page, et cela a significativement diminué les performances. Bien que la mise en cache n'ait pas eu un impact direct sur les résultats dans ce cas précis, la baisse de performance s'expliquait surtout par la taille inadaptée des images, qui ralentissait le chargement.

Ces résultats n'ont pas été retenus dans le cadre de cette analyse car ils ne montrent en rien l'utilité ou non du service worker. Cela démontre plutôt que, plus que la mise en cache elle-même, ce sont les bonnes pratiques, comme l'optimisation des images, qui impactent les performances dans tous les cas. Par conséquent, il devient important de ne pas se limiter à la mise en cache des ressources, mais aussi de garantir que chaque élément de la page soit optimisé. Cela permet d'assurer de bonnes performances, que le cache soit activé ou non, et de maximiser l'impact positif de la mise en cache.

L'effet sur l'utilisateur peut donc sembler minime. Dans la pratique, les utilisateurs ne perçoivent pas de différences significatives lors de leur navigation sur l'application de test. L'utilisation d'un service worker offre l'avantage de pouvoir fonctionner hors ligne, ce qui est toujours un plus par rapport à un système sans mise en cache.

5.3.2 Redis

Tous les résultats des tests de performance avec Grafana k6 sont disponibles dans l'annexe 3 et un résumé est retranscrit dans les tableaux ci-après.

L'analyse des performances de Redis a été réalisée, dans un premier temps, en examinant les temps de réponse en fonction du nombre d'utilisateurs virtuels (VU). Les résultats peuvent être classés en trois catégories selon le temps de réponse :

- Moins de 200 ms : Réponse très rapide, sensation d'instantanéité.
- Entre 200 ms et 1 seconde : Réponse acceptable, l'utilisateur ne remarquera probablement pas de latence.
- Plus de 1 seconde : Réponse trop lente, l'utilisateur perd la sensation d'instantanéité.

(Sematext 2023)

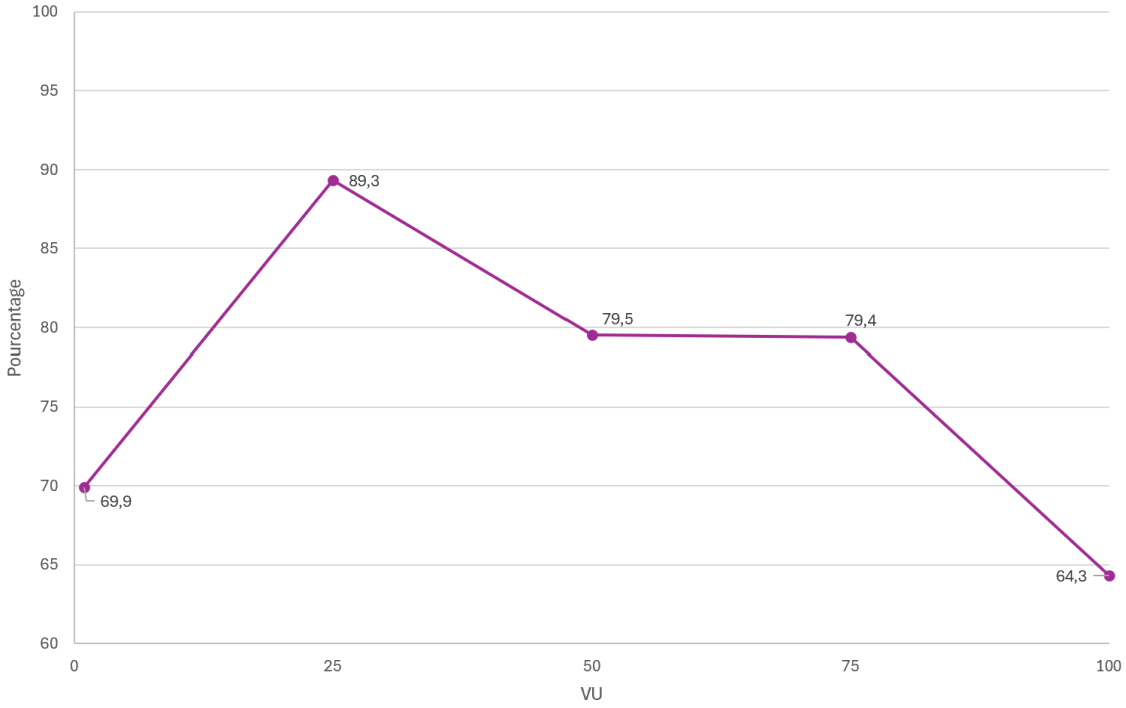
Tableau 9 : Temps de réponse (en ms) selon le nombre de VU

	1VU	25VU	50VU	75VU	100VU
Sans cache	143	487	2032	3097	3899
Redis	43	52	417	639	1393

Avec la formule suivante, il est aussi possible de calculer l'amélioration des performances obtenues en utilisant Redis :

$$\frac{Temps\ sans\ cache - Temps\ avec\ Redis}{Temps\ sans\ cache} * 100$$

Figure 5 : Amélioration des performances grâce à Redis par VU (en %)



Sans cache, les performances se dégradent rapidement : dès 50 utilisateurs, les résultats deviennent insatisfaisants. D'un autre côté, avec Redis, l'application parvient à maintenir des temps de réponse acceptables même sous une charge d'utilisateurs plus élevée. Les pourcentages d'amélioration montrent que Redis a le plus grand impact positif à 25 VU, où il atteint presque 90% d'amélioration des performances. Ensuite, ses performances commencent à diminuer. Avec 100 utilisateurs, bien que le seuil d'une seconde soit franchi, Redis réduit le temps de réponse de 64.3%, soit 2.5 secondes, ce qui est nettement perceptible pour un utilisateur lambda.

Dans un deuxième temps, l'analyse se penche sur le taux de requêtes, qui sert à mesurer le nombre de requêtes traitées par seconde par le serveur. Un taux de requêtes élevé indique une capacité à gérer efficacement un plus grand nombre d'utilisateurs, tandis qu'un taux de requêtes bas suggère que le serveur pourrait rencontrer des difficultés à traiter les demandes.

Tableau 10 : Taux de requêtes (req/s) selon le nombre de VU

	1VU	**25VU**	**50VU**	**75VU**	**100VU**
Sans cache	1	22	22.67	22.33	25.67
Redis	1	24.67	48	58.67	52.33

En analysant les taux de requêtes, il apparaît que, sans cache, le taux reste presque constant. Le système a donc une capacité fixe, atteinte autour de 25 utilisateurs virtuels.

Cela explique l'augmentation soudaine des temps de réponse à partir de 50 VU. En revanche, avec Redis, le taux de requêtes augmente avec le nombre d'utilisateurs, indiquant que le système parvient à gérer efficacement les requêtes supplémentaires. Cependant, une diminution du taux de requêtes à partir de 100 VU apparaît. Elle correspond à l'augmentation des temps de réponse à cette limite. Malgré le fait que Redis améliore les performances, le système commence à atteindre ses limites sous une charge de 100 utilisateurs.

Ainsi, pour notre application, Redis se positionne comme une solution efficace pour optimiser les performances. Les résultats montrent une grande amélioration des temps de réponse, d'au moins 64%, avec un nombre d'utilisateurs virtuels allant jusqu'à 100. Dans cette plage d'utilisation, la différence est clairement visible, notamment à 25 VU. Au-delà de 100 utilisateurs, les résultats baissent et il faudrait envisager d'autres méthodes de mise en cache pour garder une application performante.

5.3.3 Limites

Les tests ont été réalisés en local, ce qui peut influencer les résultats, notamment en raison de la distance réduite entre le serveur et le client. En condition de déploiement sur un serveur distant, les performances pourraient varier, mais les différences observées entre les configurations avec et sans cache restent révélatrices.

De plus, ces tests ne prennent pas en compte d'autres aspects liés à l'optimisation des performances avec la mise en cache, comme la gestion des invalidations des données. Par exemple, notre application ne simule pas de changements dans la base de données, ce qui aurait permis de vérifier comment le cache se comporte pour servir des informations fraîches au lieu de données périmées.

Une autre limite réside dans le fait que les deux méthodes de cache n'ont pas été testées ensemble. En effet, il est possible d'utiliser conjointement un service worker et Redis mais cela soulève des questions sur les invalidations. Si le service worker est utilisé, il priorisera les données mises en cache localement, même pour les données dynamiques. Même si cela peut être plus rapide, la gestion des caches client pour des données dynamiques est compliquée car le serveur n'a pas de contrôle direct sur les informations stockées dans le navigateur. De plus, le stockage côté client est limité en espace, ce qui implique que le service worker devrait être adapté pour ne pas récupérer toutes les données dynamiques, surtout lorsqu'il y a déjà Redis en charge de cette tâche. Dans ce cas, il aurait fallu mettre en place une logique pour vider le cache client une fois qu'il atteint sa capacité maximale ou après un certain temps.

5.3.4 Synthèse

L'utilisation du service worker n'a pas montré d'effets impressionnants. Pour réellement comprendre comment il optimise les performances, je pense qu'il serait plus judicieux de le tester sur un site Web plus grand et plus complet que la page statique de mon application. Il ne faut pas en tirer de bonnes ou de mauvaises conclusions, mais plutôt comprendre que le scénario de test n'était pas forcément adapté au service worker.

À l'inverse, Redis s'est présenté comme une très bonne technologie de mise en cache. Avec 25 utilisateurs simultanés, ses performances sont bien mieux que celles auxquelles je m'attendais. Néanmoins, si le but est d'avoir une application ultra performante, j'estime que la limite de Redis se situe autour de 75 utilisateurs. Dans un cas d'une utilisation réelle, elle pourrait même être baissée à 50 ou 25 utilisateurs. Ceci implique que le passage à une autre méthode de mise en cache est une bonne solution dans le cas d'un site d'e-commerce en essor. Redis pourrait utilisé en mode cluster, c'est-à-dire dans le cadre d'un cache distribué. Je pense que c'est une bonne alternative pour conserver de bonnes performances.

Dans tous les cas, les tests de performances effectués en local ne suffisent pas. Il peut y avoir de très grandes différences avec des tests exécutés dans un contexte réel et l'analyse de ces derniers n'est pas à mettre de côté.

6. Conclusion

Arrivée à la fin de ce travail, je peux affirmer avoir atteint l'objectif initial qui était d'analyser et comprendre les différentes méthodes de mise en cache, leur importance, leurs avantages et inconvénients, ainsi que de tester en pratique leur impact sur les performances.

Chaque méthode présente ses propres avantages et aucune d'elles n'est universellement meilleure. Pour choisir la méthode de mise en cache appropriée, il faut commencer par bien comprendre les besoins et les contraintes du projet. L'analyse comparative menée à travers deux critères principaux – la performance et la sécurité – a permis de déterminer quelles technologies sont les plus efficaces selon ces aspects. Toutefois, ces critères ne sont pas toujours les seuls à prendre en compte lorsqu'il s'agit de mettre en place une stratégie de mise en cache. En fonction des besoins d'une application, d'autres facteurs comme la scalabilité, la facilité de mise en œuvre ou encore le coût peuvent entrer en jeu. Ce n'est pas nécessairement la technologie la mieux notée qui conviendra, mais celle qui répondra le mieux aux objectifs précis, que ce soit en fonction des données à gérer, du volume de trafic attendu ou des contraintes de sécurité. Il n'existe pas de solution unique qui soit idéale pour tous les cas, et les comparaisons doivent être vues comme des indicateurs à adapter à chaque situation avant toute implémentation.

Les tests effectués ont permis de mettre en évidence les gains de performance obtenus grâce à la mise en cache, dans un environnement local et contrôlé. La mise en cache avec Redis a donné d'excellents résultats, en améliorant de plus de 60% les performances, que ce soit avec 1 ou 100 utilisateurs connectés simultanément. Les résultats du service worker sont plus modérés, ce qui peut s'expliquer par un éventuel manque de pertinence dans les données mises en cache. Ces résultats montrent l'importance de maîtriser les outils de test pour mieux comprendre comment la mise en cache améliore concrètement les performances d'une application. En ce sens, il est recommandé à tout développeur de réaliser des tests réguliers afin de mieux cerner pourquoi certaines attentes ne sont pas toujours atteintes, et d'envisager d'autres solutions ou technologies si nécessaire.

Pourtant, il est important de noter que, même si la mise en cache contribue significativement à l'optimisation, elle ne résoudra pas à elle seule des problèmes de performance si le système sous-jacent n'est pas optimisé. Il est donc préconisé de vérifier d'abord d'autres aspects du système avant de se tourner vers la mise en cache. Il existe plusieurs pistes à explorer en cas de mauvaises performances, comme par

exemple la qualité du code, la complexité des requêtes vers la base de données, l'équilibrage de charge dans un contexte distribué, l'optimisation des images, ou encore la compression et la minification des fichiers.

En conclusion, il est important de rappeler que l'optimisation des performances Web continuera d'avoir un rôle central dans la conception de systèmes toujours plus rapides et efficaces. Les temps de chargement des pages Web ont un impact direct sur l'expérience utilisateur. En effet, ils représentent la première impression qu'un utilisateur a d'une application ou d'un site. Un utilisateur confronté à des temps d'attente trop longs est plus susceptible de quitter la plateforme sans interagir davantage.

Bibliographie

AFFONSO, Gabriel, 2024. Demystifying the Web: A Deep Dive into HTTP, TCP and Beyond — Part 1. *Medium* [en ligne]. 8 avril 2024. Disponible à l'adresse : https://medium.com/@gabrielaffonso/demystifying-the-web-a-deep-dive-into-http-tcp-and-beyond-part-1-1c3c1efaa3e7 [consulté le 30 juillet 2024].

AGARWAL, Ankit, 2024. Client Server Architecture : System Design. *Medium* [en ligne]. 2 juin 2024. Disponible à l'adresse : https://medium.com/@ankitagarwal702/client-server-architecture-system-design-433d97e6c72d [consulté le 22 juillet 2024].

APACHE, 2024. Guide de la mise en cache. *Apache* [en ligne]. 2 avril 2024. Disponible à l'adresse : https://httpd.apache.org/docs/2.4/fr/caching.html [consulté le 25 août 2024].

APURVA, Agrawal, 2023. Different Caching Techniques. *Medium* [en ligne]. 21 septembre 2023. Disponible à l'adresse : https://medium.com/@aggarwalapurva89/different-caching-techniques-660ad8d97a8a [consulté le 10 août 2024].

AWS, 2024. Réseau de diffusion de contenu. *Amazon Web Services, Inc.* [en ligne]. 2024. Disponible à l'adresse : https://aws.amazon.com/fr/cloudfront/ [consulté le 23 août 2024].

BINIELI, Moshe, 2023. Cache Strategies. *Medium* [en ligne]. 22 avril 2023. Disponible à l'adresse : https://medium.com/@mmoshikoo/cache-strategies-996e91c80303 [consulté le 16 août 2024].

BYTEBYTEGO, 2022. *Proxy vs Reverse Proxy (Real-world Examples)* [en ligne]. 25 octobre 2022. Disponible à l'adresse : https://www.youtube.com/watch?app=desktop&v=4NB0NDtOwIQ [consulté le 13 août 2024].

BYTESIZEDPIECES, 2023. Cache Pros and Cons. *ByteSizedPieces* [en ligne]. 26 mai 2023. Disponible à l'adresse : https://www.bytesizedpieces.com/posts/cache-pro-con [consulté le 10 août 2024].

CHROME, 2023. Lighthouse performance scoring. *Chrome for Developers* [en ligne]. 2023. Disponible à l'adresse : https://developer.chrome.com/docs/lighthouse/performance/performance-scoring [consulté le 19 septembre 2024].

CLOUDFLARE, 2023. Qu'est-ce que la durée de vie (TTL) ? | Définition du TTL. *Cloudflare* [en ligne]. 2023. Disponible à l'adresse : https://www.cloudflare.com/fr-fr/learning/cdn/glossary/time-to-live-ttl/ [consulté le 11 septembre 2024].

CLOUDFLARE, 2024. Qu'est-ce que la mise en cache ? *Cloudflare* [en ligne]. 2024. Disponible à l'adresse : https://www.cloudflare.com/fr-fr/learning/cdn/what-is-caching/ [consulté le 3 août 2024].

DANG, Thinh, 2023. Mastering Caching Strategies: A Comprehensive Guide. *ThinhDA* [en ligne]. 8 septembre 2023. Disponible à l'adresse : https://thinhdanggroup.github.io//caching-stategies/ [consulté le 10 août 2024].

D'ARROS, Thibaud, 2024. Démystifier la Mise en Cache sur le Web. *laConsole* [en ligne]. 14 juin 2024. Disponible à l'adresse : https://laconsole.dev/blog/demystifier-mise-en-cache-web/ [consulté le 24 août 2024].

DELOITTE IRELAND, 2020. Milliseconds Make Millions. *Deloitte* [en ligne]. 24 mars 2020. Disponible à l'adresse : https://www.deloitte.com/ie/en/services/consulting/research/milliseconds-make-millions.html [consulté le 3 août 2024].

GAJBHIYE, Amar, 2020. How to Secure Apache Ignite cluster? *Medium* [en ligne]. 5 octobre 2020. Disponible à l'adresse : https://aamargajbhiye.medium.com/how-to-secure-apache-ignite-cluster-cd595b99ec5e [consulté le 27 août 2024].

GARLAND, Frank, 2021. Varnish vs. Nginx. *Azion Technologies* [en ligne]. 10 février 2021. Disponible à l'adresse : https://www.azion.com/en/blog/varnish-vs-nginx/ [consulté le 26 août 2024].

HAZELCAST, 2024. Security Overview. *Hazelcast documentation* [en ligne]. août 2024. Disponible à l'adresse : https://docs.hazelcast.com/hazelcast/5.5/security/overview [consulté le 27 août 2024].

HEAVY.AI, 2022. What is Client-Server? Definition and FAQs. *HEAVY.AI* [en ligne]. 2022. Disponible à l'adresse : https://www.heavy.ai/technical-glossary/client-server [consulté le 22 juillet 2024].

HUNG, Jimmy, 2021. [Cloudflare vs Cloudfront] What exactly are the differences between them? *Free Web Stack* [en ligne]. 22 juillet 2021. Disponible à l'adresse : https://www.freewebstack.com/cloudflare-vs-cloudfront-what-exactly-are-the-differences-between-them/ [consulté le 28 août 2024].

IBM, 2024. Caching Proxy. *IBM* [en ligne]. 26 mars 2024. Disponible à l'adresse : https://www.ibm.com/docs/en/was-nd/9.0.5?topic=overview-caching-proxy [consulté le 13 août 2024].

LOADFORGE, 2023. Optimizing Apache Server Response Times with Effective Caching Techniques. *LoadForge* [en ligne]. 2023. Disponible à l'adresse : https://loadforge.com/guides/leveraging-caching-to-accelerate-apache-server-response-times [consulté le 25 août 2024].

MDN, 2023. Mise en cache HTTP. [en ligne]. 3 août 2023. Disponible à l'adresse : https://developer.mozilla.org/fr/docs/Web/HTTP/Caching [consulté le 10 août 2024].

MDN, 2024a. La relation Client-Serveur - Apprendre le développement web. [en ligne]. 11 avril 2024. Disponible à l'adresse : https://developer.mozilla.org/fr/docs/Learn/Server-side/First_steps/Client-Server_overview [consulté le 22 juillet 2024].

MDN, 2024b. Utiliser les service workers - Les API Web. *MDN* [en ligne]. 8 août 2024. Disponible à l'adresse : https://developer.mozilla.org/fr/docs/Web/API/Service_Worker_API/Using_Service_Workers [consulté le 10 août 2024].

MENDEZ, Nicholas, 2021. A « Brief » History of the Web Part 2. *DEV Community* [en ligne]. 2 octobre 2021. Disponible à l'adresse : https://dev.to/snickdx/a-brief-history-of-the-web-9c3 [consulté le 25 juillet 2024].

MICROSOFT, 2016. Introduction to Web Storage (Windows). [en ligne]. 20 octobre 2016. Disponible à l'adresse : https://learn.microsoft.com/en-us/previous-versions/windows/desktop/legacy/bg142799(v=vs.85) [consulté le 24 juillet 2024].

MIRYALA, Supraja, 2023. Caching vs Local Storage vs Session Storage vs Cookie. *Medium* [en ligne]. 30 décembre 2023. Disponible à l'adresse : https://medium.com/@supraja_miryala/caching-vs-local-storage-vs-session-storage-vs-cookie-ef60fbb7aa71 [consulté le 26 juillet 2024].

MOEINNIA, Moein, 2023. Exploring In-Memory Databases: Redis, Memcached, and Beyond. *Medium* [en ligne]. 6 septembre 2023. Disponible à l'adresse : https://javascript.plainenglish.io/exploring-in-memory-databases-redis-memcached-and-beyond-bd940380ee1 [consulté le 27 août 2024].

ORACLE, 2020. Qu'est-ce qu'une base de données ? *Oracle* [en ligne]. 24 novembre 2020. Disponible à l'adresse : https://www.oracle.com/ch-fr/database/what-is-database/ [consulté le 30 juillet 2024].

PATIL, Rohit, 2020. HTTP Request, HTTP Response, Context and Headers : Part III. *Medium* [en ligne]. 29 mai 2020. Disponible à l'adresse : https://medium.com/@rohitpatil97/http-request-http-response-context-and-headers-part-iii-5c37bd4cb06b [consulté le 2 août 2024].

PEÑA, Tiffany, 2019. Client Side Session vs Server Side Session. *Medium* [en ligne]. 10 mai 2019. Disponible à l'adresse : https://medium.com/@tiff.sage/client-side-session-vs-server-side-session-d506f5408e8c [consulté le 2 août 2024].

RAMOTION, 2023. What is Web Storage - Types, Tips & Use Cases. *Ramotion* [en ligne]. 30 octobre 2023. Disponible à l'adresse : https://www.ramotion.com/blog/what-is-web-storage/ [consulté le 22 juillet 2024].

REDIS, 2024a. Distributed Caching. *Redis* [en ligne]. 7 février 2024. Disponible à l'adresse : https://redis.io/glossary/distributed-caching/ [consulté le 15 août 2024].

REDIS, 2024b. Redis security. *Redis* [en ligne]. 2024. Disponible à l'adresse : https://redis.io/docs/latest/operate/oss_and_stack/management/security/ [consulté le 26 août 2024].

SCALEGRID, 2024. Redis Vs Memcached In 2024. *ScaleGrid* [en ligne]. 1 avril 2024. Disponible à l'adresse : https://scalegrid.io/blog/redis-vs-memcached/ [consulté le 25 août 2024].

SEMATEXT, 2023. What Is Response Time & How to Reduce It. *Sematext* [en ligne]. 2023. Disponible à l'adresse : https://sematext.com/glossary/response-time/ [consulté le 19 septembre 2024].

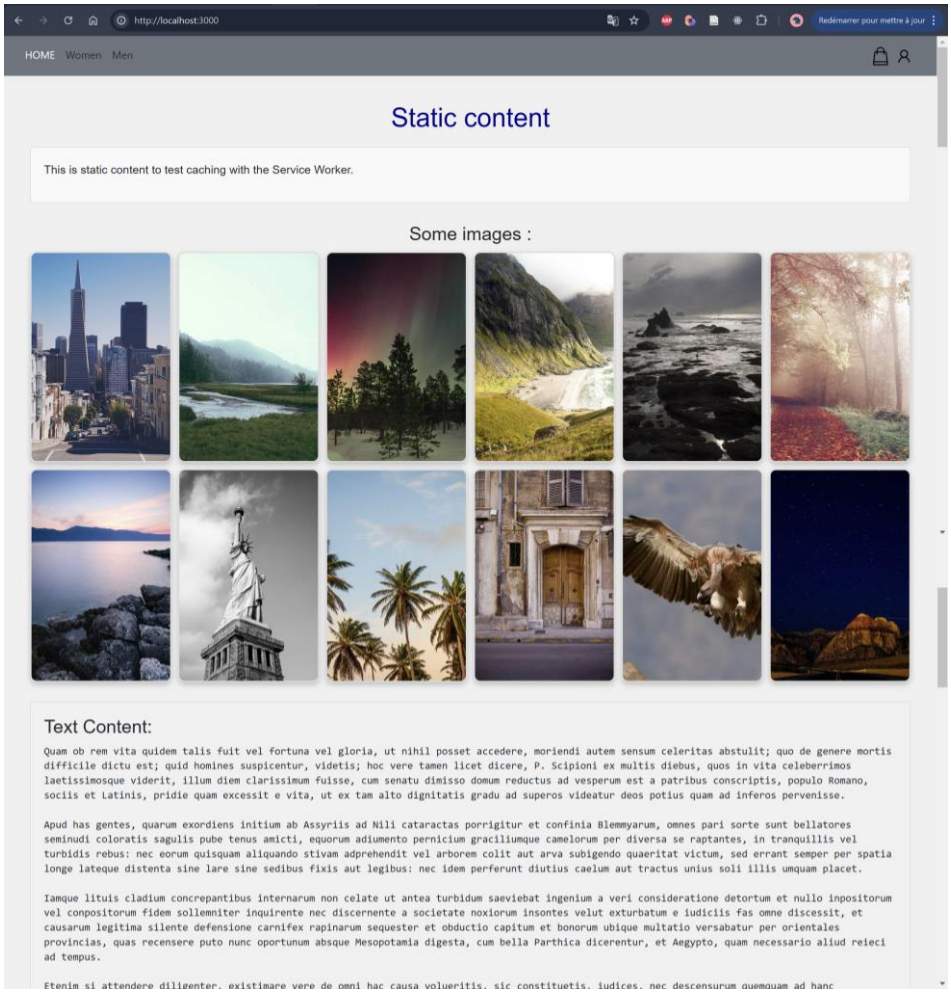
WIKIPEDIA, 2024a. Navigateur web. *Wikipedia* [en ligne]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Navigateur_web&oldid=217195873#Exemples_de_navigateurs [consulté le 2 août 2024]. Page Version ID: 217195873

WIKIPEDIA, 2024b. Encapsulation (networking). *Wikipedia* [en ligne]. Disponible à l'adresse : https://en.wikipedia.org/w/index.php?title=Encapsulation_(networking)&oldid=1237876021 [consulté le 30 juillet 2024]. Page Version ID: 1237876021

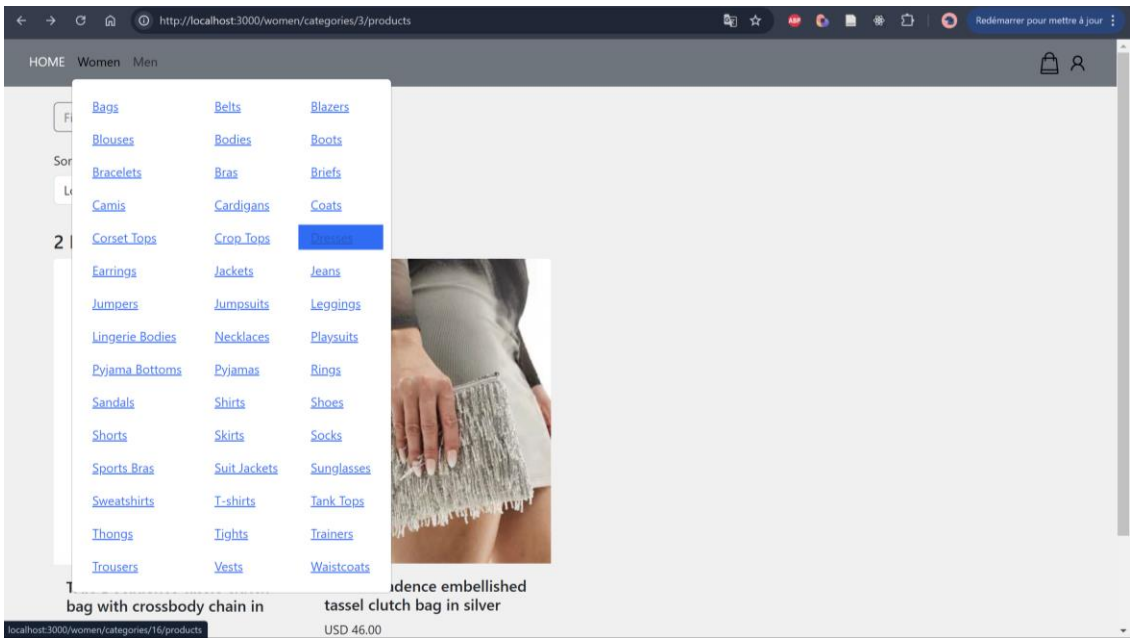
ZEE PALM, 2024. Service Worker Security Best Practices - 2024 Guide. *Zee Palm* [en ligne]. 7 mai 2024. Disponible à l'adresse : https://www.zeepalm.com/blog/service-worker-security-best-practices-2024-guide [consulté le 25 août 2024].

Annexe 1 : Captures d’écran de l’application

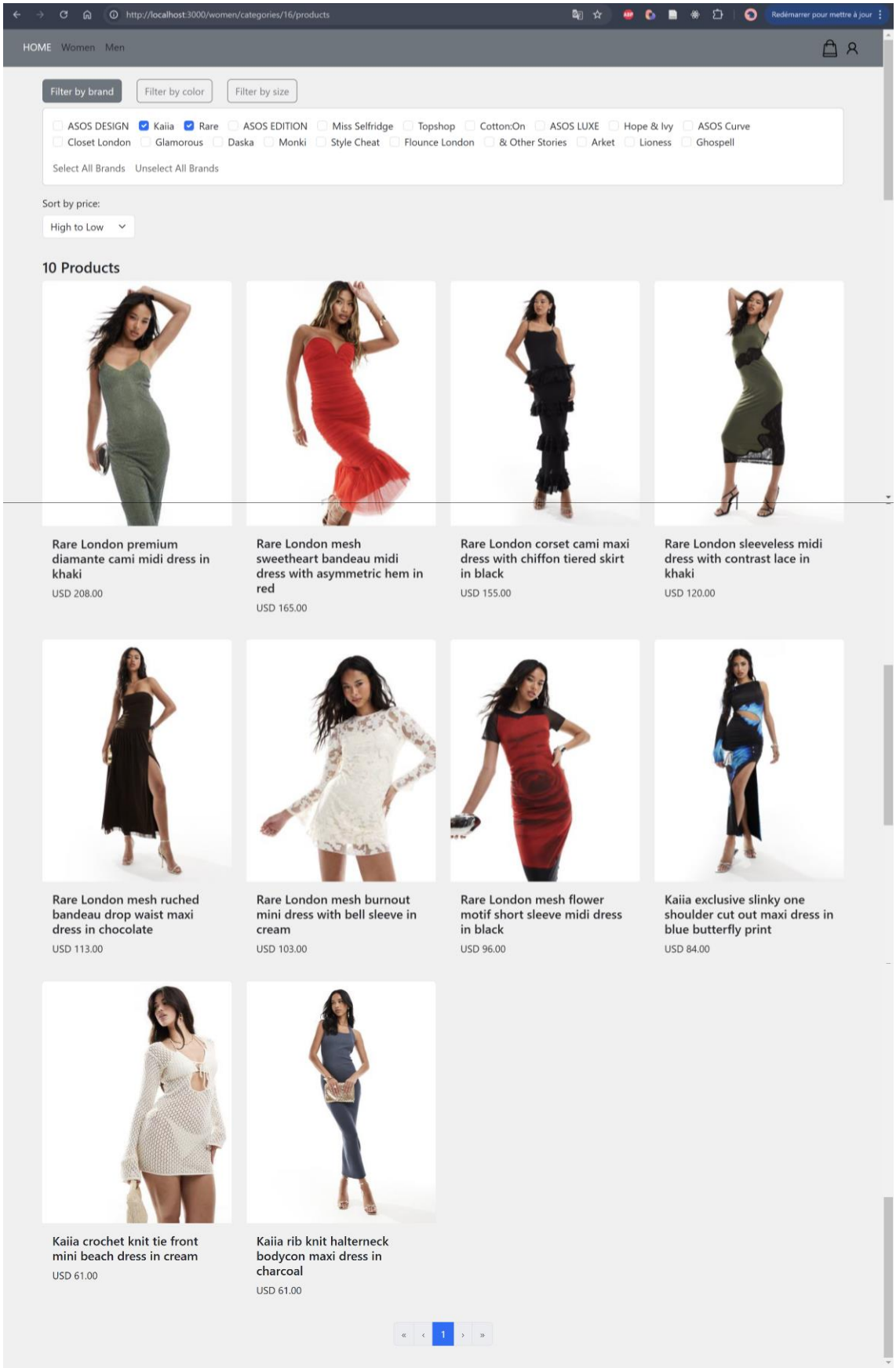
Page d’accueil (contenu statique) :



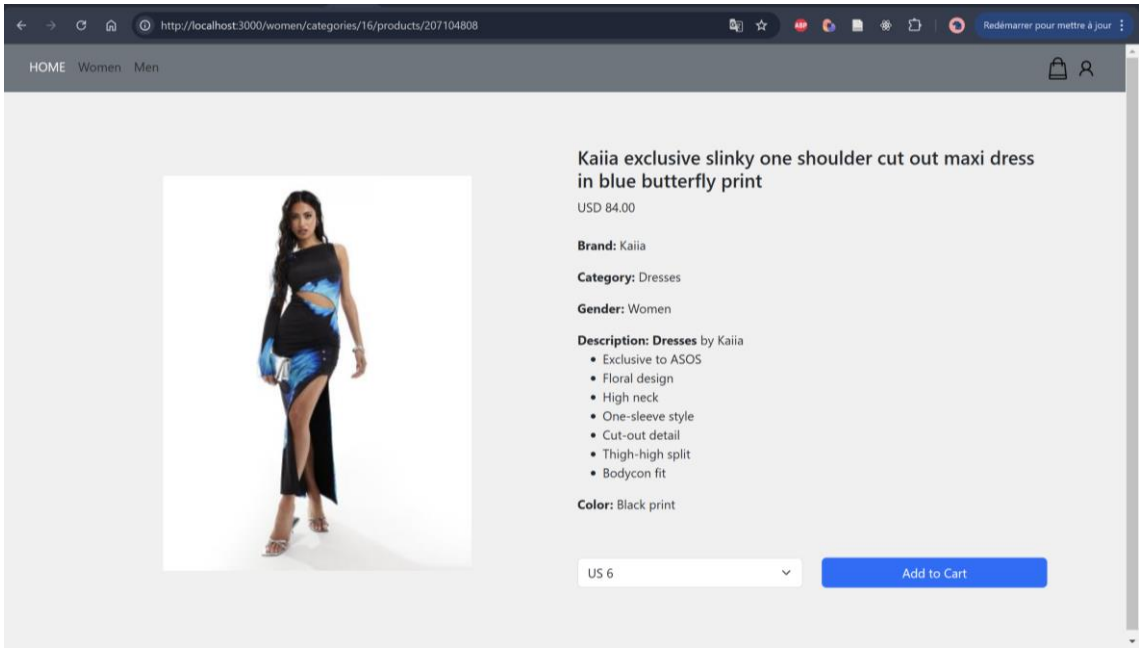
Sélection d’une catégorie femme :



Vêtements de la catégorie robes pour femmes, avec un filtre sélectionné :



Détail d’un produit :



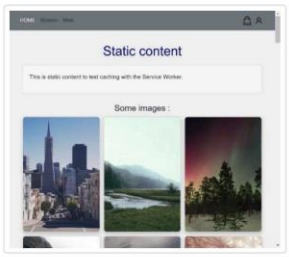
Annexe 2 : Résultats des performances du service worker (Lighthouse)

Sans cache :

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 50–89 90–100



METRICS Expand view

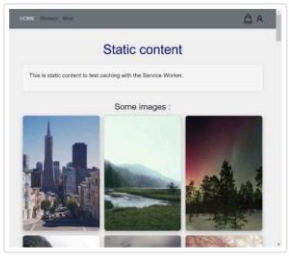
First Contentful Paint 0.6 s	Largest Contentful Paint 1.7 s
Total Blocking Time 80 ms	Cumulative Layout Shift 0.036
Speed Index 0.7 s	

Sans cache, fluctuation :

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

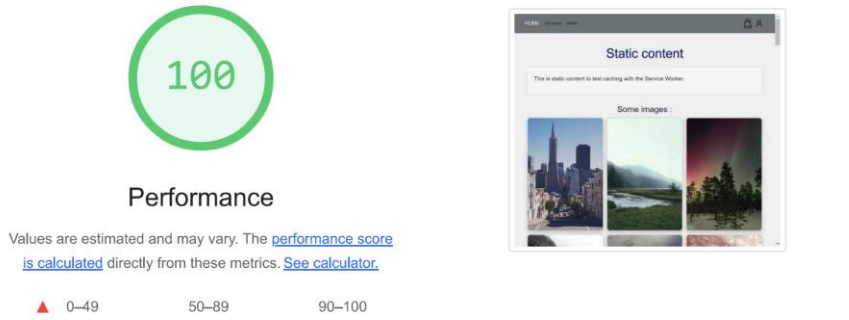
▲ 0–49 50–89 90–100



METRICS Expand view

First Contentful Paint 0.5 s	Largest Contentful Paint 1.6 s
Total Blocking Time 70 ms	▲ Cumulative Layout Shift 0.48
Speed Index 0.7 s	

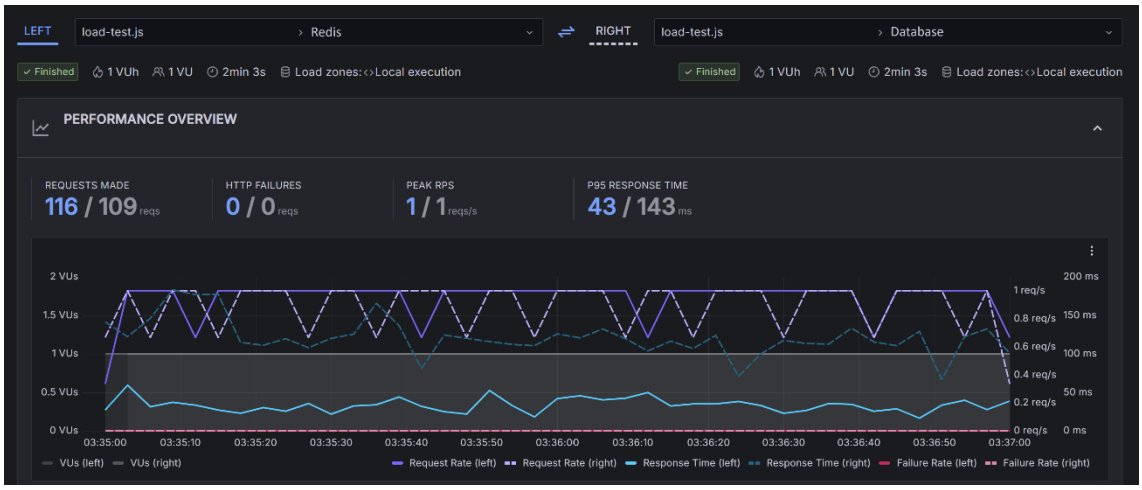
Avec service worker :



Annexe 3 : Résultats des performances de Redis (Grafana k6)

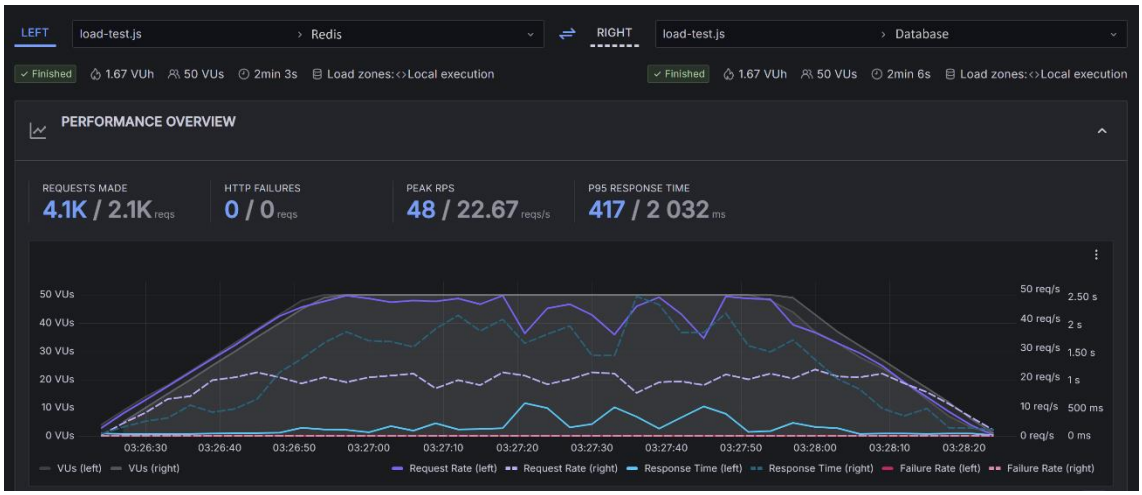
Sur tous les graphiques, les traits pleins représentent les performances avec Redis et les traitillés représentent les performances sans cache (appels à la base de données directement). En gris, ce sont le nombre d'utilisateurs virtuels, en violet les taux de requêtes, en bleu les temps de réponse et en rouge le taux d'échec.

1 utilisateur virtuel :

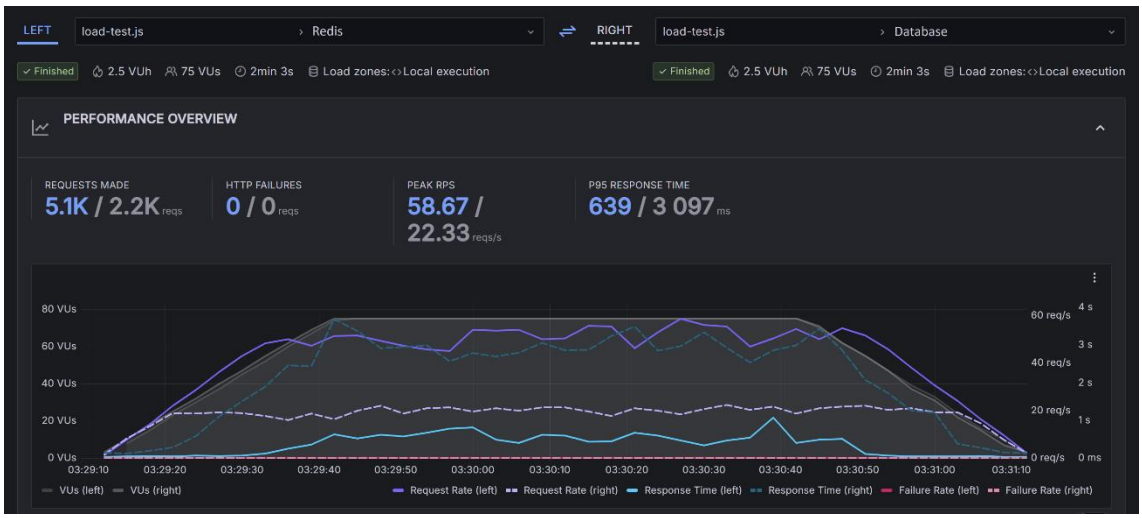


25 utilisateurs virtuels :

50 utilisateurs virtuels :



75 utilisateurs virtuels :



100 utilisateurs virtuels :