Baptiste Ballesteros 4a wd²
Why performance is important

La performance est essentielle pour plusieurs raisons :

- **Expérience utilisateur améliorée** : Une application rapide permet aux utilisateurs de naviguer sans frustration. Un site lent engendre une augmentation du taux de rebond et peut réduire significativement les conversions, comme le montre le cours (« une application lente va frustrer puis repousser les utilisateurs »).
- **Optimisation des coûts et des ressources** : En optimisant la performance, on réduit l'utilisation des ressources (CPU, RAM, bande passante) ce qui se traduit par une diminution des coûts d'hébergement et de maintenance.
- **Meilleur référencement** : Les moteurs de recherche privilégient les sites qui se chargent rapidement. Une bonne performance contribue donc à un meilleur classement dans les résultats de recherche, augmentant la visibilité et le trafic.
- **Scalabilité et fiabilité** : Une application performante est capable de supporter une montée en charge sans dégrader l'expérience utilisateur. Cela est crucial pour assurer la stabilité et la fiabilité du service, même lors de pics de trafic.
- **Expérience développeur et maintenance** : Des performances bien optimisées rendent l'application plus facile à maintenir et à faire évoluer.

En résumé, optimiser la performance d'une application web n'est pas seulement une question de rapidité, c'est un enjeu stratégique qui impacte directement la satisfaction des utilisateurs, les coûts opérationnels, le référencement et la pérennité du service.

What is wrong with the application

L'application présente plusieurs problèmes de performance qui impactent directement son temps de chargement et l'expérience utilisateur :

1. **Assets non optimisés**

   Les images utilisées ne sont pas compressées ni redimensionnées correctement en ne sont pas en format webp. Cela ralentit le chargement de la page.

   Les fichiers CSS et JavaScript ne sont pas minifiés, ce qui augmente leur poids en incluant des espaces, commentaires et caractères superflus.

2. **Chargement des assets**

   Les images et autres ressources ne sont pas chargées au moment où elles deviennent visibles (lazy loading), ce qui entraîne un chargement initial plus lent et une utilisation plus élevée de la bande passante dès l'ouverture de la page.

   Les en-têtes de cache ne sont pas mis en place, ce qui oblige le navigateur à re-télécharger les images à chaque visite, au lieu de les récupérer depuis son cache.

3. **Code inefficace**

   Certaines fonctions de l'application semblent être écrites de manière inefficace, avec un code qui pourrait être réécrit pour améliorer la rapidité d'exécution (par exemple, en évitant des boucles inutiles ou des traitements redondants).

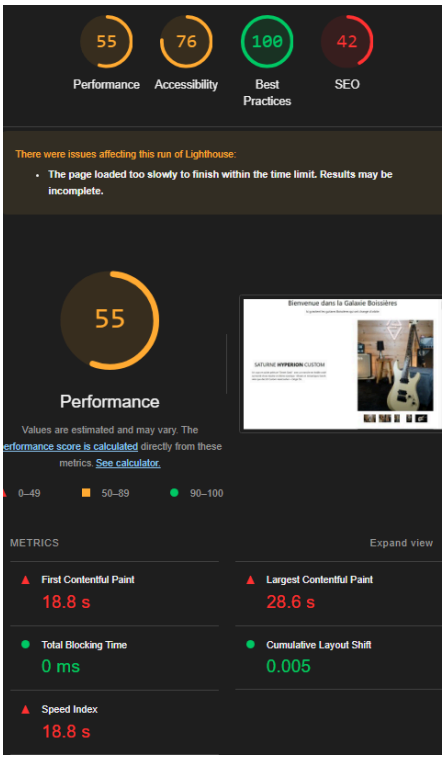
   La manière dont les données sont récupérées depuis la base de données (nombre de requêtes, jointures non optimisées, absence d'index appropriés) contribue à un temps de réponse plus long, on doit pouvoir éviter les requêtes n+1.

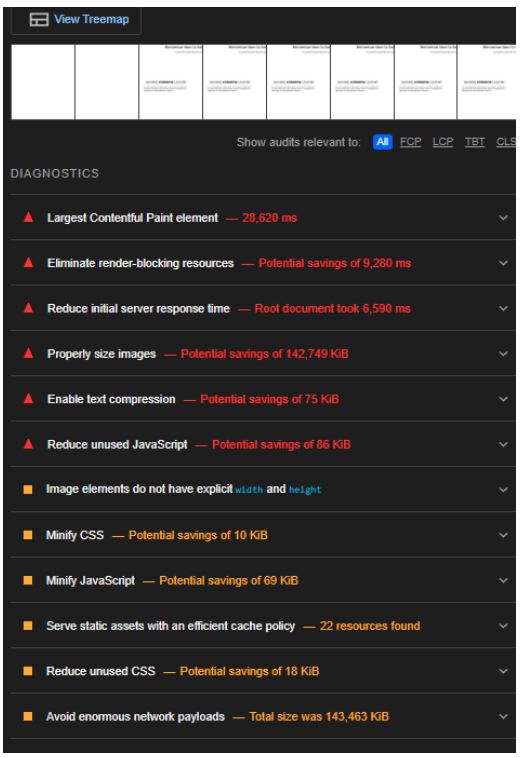
En résumé, l'application souffre d'un manque d'optimisation global : les ressources (images, CSS, JS) sont trop lourdes et ne sont pas servies de manière intelligente (lazy loading, cache), le code et les requêtes SQL ne sont pas optimisés pour la performance. Une amélioration sur ces points permettrait de réduire significativement les temps de chargement et d'améliorer l'expérience utilisateur.

Tests and measurements

Pour tester et mesurer les performances de l’application, j'utiliserai Lighthouse. Ce choix s'explique par le fait qu'il est crucial d'avoir des mesures quantitatives pour confirmer nos hypothèses. Lighthouse fournit un ensemble de métriques clés, notamment le Largest Contentful Paint, le First Contentful Paint, le Cumulative Layout Shift, et bien d'autres indicateurs des Core Web Vitals. Ces données chiffrées permettent d'identifier précisément les points faibles de pouvoir voir l’évolution après les différentes optimisations.

Analyse lighthouse :





Nous remarquons que le site souffre d'une lenteur, en effet le largest Contentful Paint est très élevé (plus de 28s), cela est mauvais pour l'expérience utilisateur
Il y a beaucoup de place à gagner (142 KB) en compressant les images, les redimensionnements et en les passant en webp. Les scripts et feuilles de style bloquent le rendu initial, allongeant la phase avant que l'utilisateur ne voie quoi que ce soit (9s).

En résumé, le temps de chargement reste trop élevé, principalement à cause d'images lourdes et mal optimisées, de ressources également non optimisées. Des améliorations ciblées sur ces points devraient permettre de réduire significativement les temps de rendu et d'augmenter le score global.

What are the solutions ?

Voici un résumé des solutions de programmation immédiates pour améliorer l'application :

1. **Optimiser les requêtes SQL**

   Réduire les requêtes N+1 en regroupant la récupération des données (jointures et fetch en une seule requête).
   Centraliser la logique d'accès aux données dans les repositories pour éviter les multiples appels à la base de données.

2. **Optimiser le template Twig**

   Mettre en place le lazy loading pour les images.
   Gérer correctement la taille et l'affichage conditionnel du contenu.

3. **Améliorer la gestion des images**

   Convertir en WebP, compresser et redimensionner pour diminuer le poids des images (utilisation de imagemagick) .
   Utiliser le lazy loading pour différer le chargement des images hors écran.

4. **Compresser et minifier les ressources CSS/JS**

   Trouver pourquoi les feuilles de style bloquent le rendu initial, sûrement preload le css/minifier le css et voir pour garder en cache les données avec un htacces.

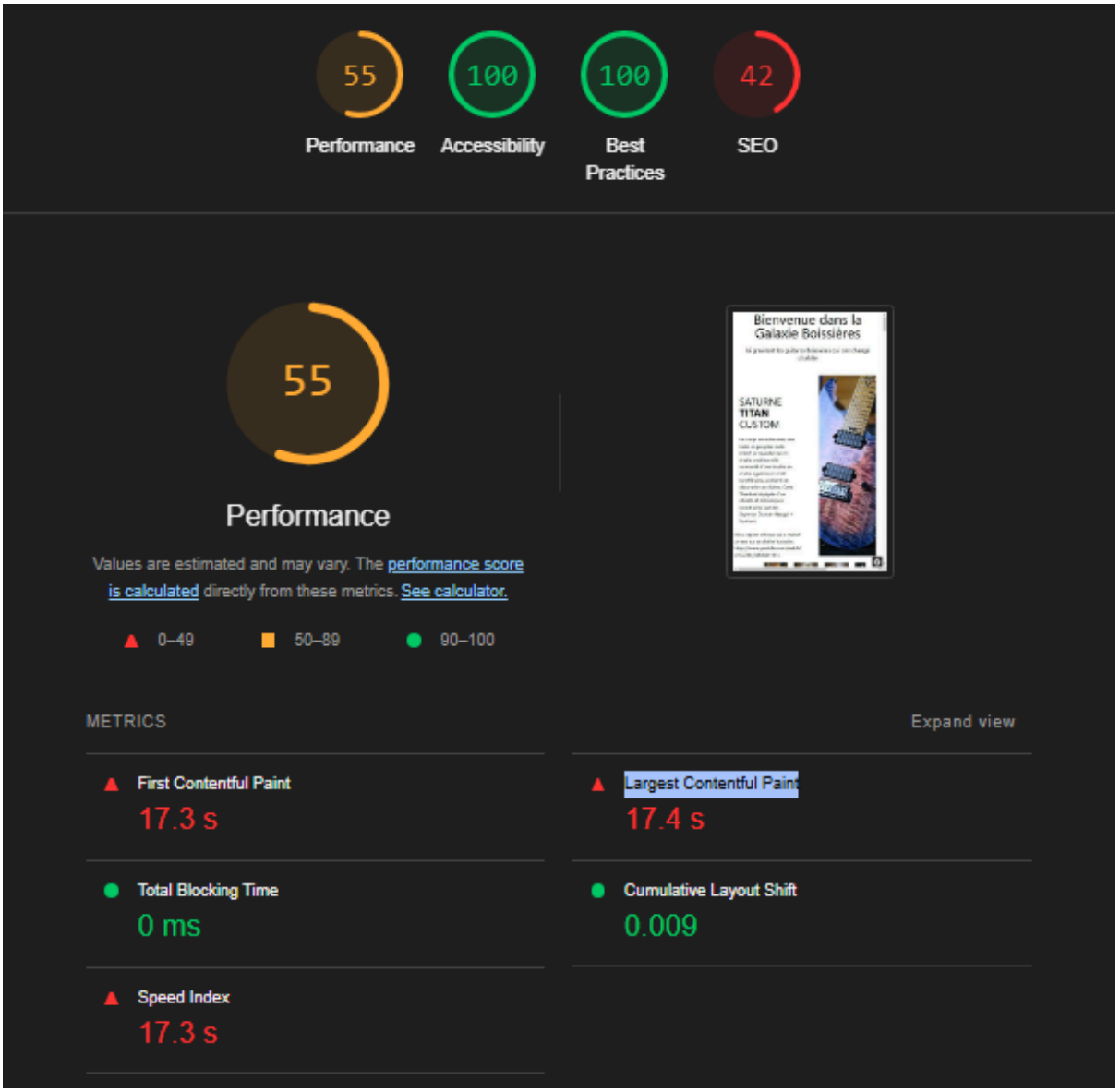
En mettant en place ces optimisations, nous réduisons significativement les temps de chargement, tout en améliorant l'expérience utilisateur et la performance globale de l'application.

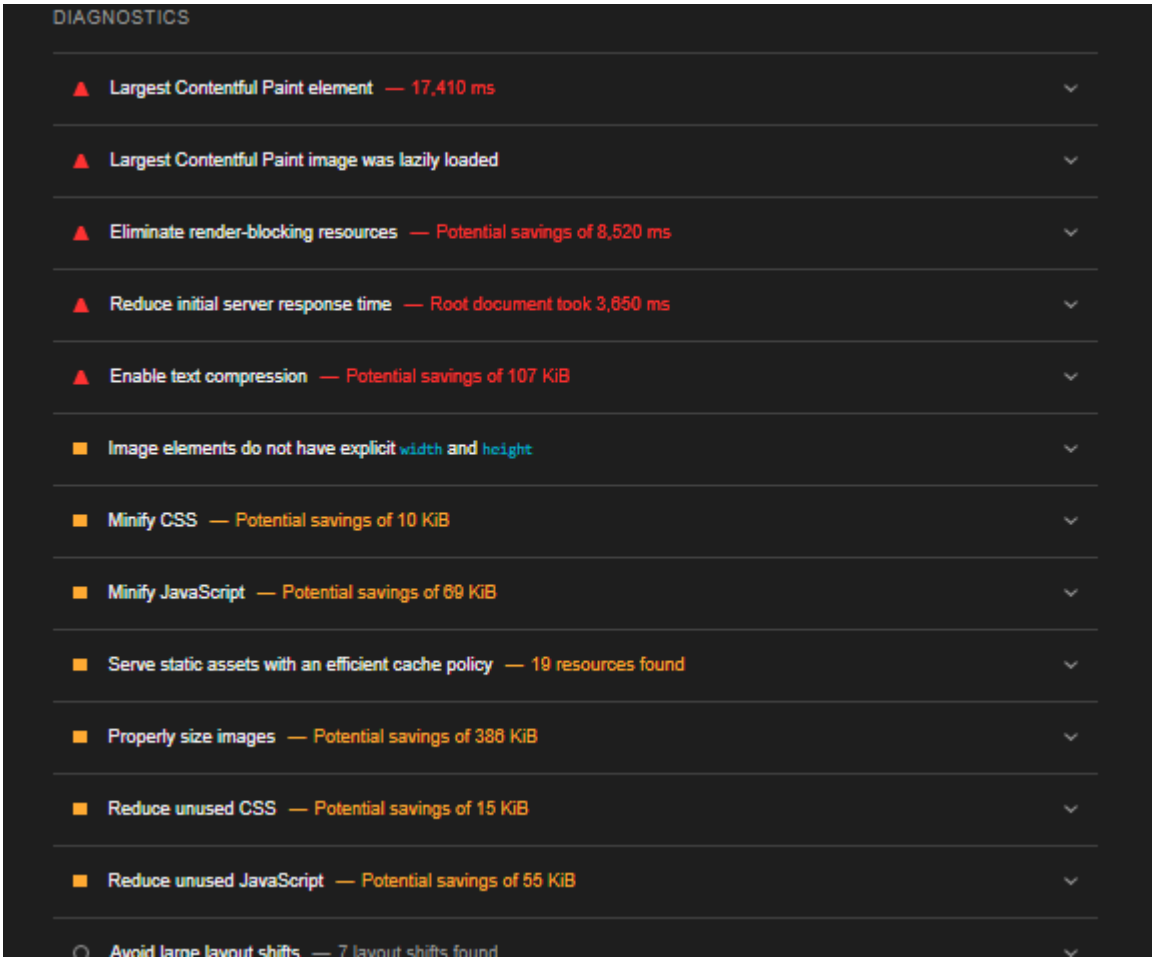
Conclusion

Ici on peut voir que le largest contentful Paint est beaucoup plus bas (de 11 secondes) ce qui montre un amélioration du la vitesse du site et donc d'une meilleure expérience utilisateur. Cela est dû au modification sur le site de mise en place, optimisation des requêtes sql dans le repository (jointures et fetch en une seule requête), optimisation du front avec des lazy loading sur les images.

Ce qui a également beaucoup aidé est d’avoir compressé, redimensionner et d’avoir converti les images en format webp avec ImageMagick. Ce qui à permis de gagner de la place et de la rapidité.

À l'avenir, il sera crucial de conserver nos ressources en cache afin de réduire les temps de chargement et d'améliorer l'expérience utilisateur. Mettre en place un CDN pour nos ressources statiques permettra de les servir depuis des serveurs géographiquement proches des utilisateurs, diminuant ainsi la latence. De plus, l'implémentation de techniques comme le prefetch contribuera à anticiper les besoins des utilisateurs en pré chargeant les ressources avant même qu'elles ne soient demandées, garantissant ainsi une navigation fluide et réactive.




DIAGNOSTICS
Largest Contentful Paint element — 17,410 ms
Largest Contentful Paint image was lazily loaded
Eliminate render-blocking resources — Potential savings of 8,520 ms
Reduce initial server response time — Root document took 3,650 ms
Enable text compression — Potential savings of 107 KiB
Image elements do not have explicit width and height
Minify CSS — Potential savings of 10 KiB
Minify JavaScript — Potential savings of 69 KiB
Serve static assets with an efficient cache policy — 19 resources found
Properly size images — Potential savings of 386 KiB
Reduce unused CSS — Potential savings of 15 KiB
Reduce unused JavaScript — Potential savings of 55 KiB
Avoid large layout shifts — 7 layout shifts found