




# SYNTHESE DU PROJET CHALLENGE DESIGN4GREEN 2020 REPORT

Numéro d'équipe / Team Number : 59



## GT MTERIX

SCORE (PageSpeed Score) : 97% (only percentage)  
SCREENSHOT 18h30 05/11



### Latest Performance Report for:

<https://59design4green.tech/>

Report generated: Thu, Nov 5, 2020 9:27 AM -0800  
Test Server Region:  Vancouver, Canada  
Using:  Chrome (Desktop) 75.0.3770.100, PageSpeed 1.15-gt1.3, YSlow 3.1.8

#### Performance Scores

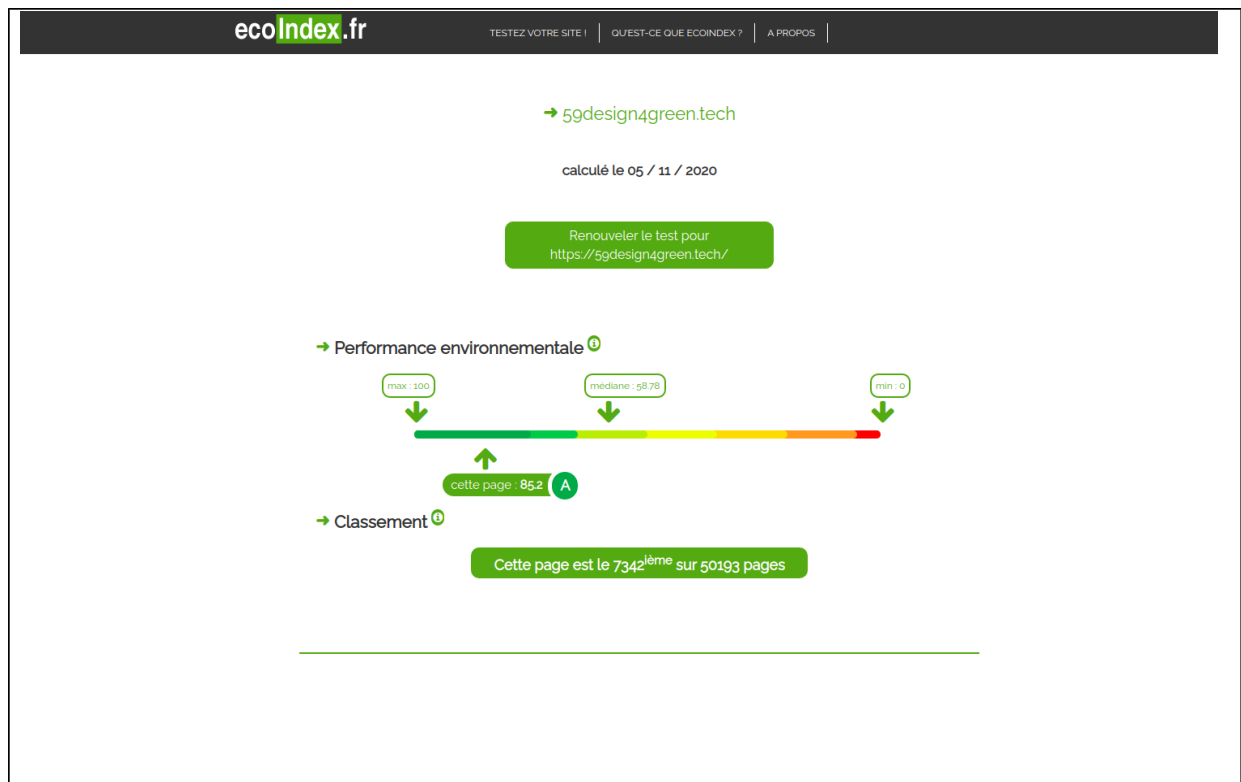
PageSpeed Score	YSlow Score
<b>A (97%)</b> ^	<b>A (95%)</b> ^

#### Page Details

Fully Loaded Time	Total Page Size	Requests
<b>2.8s</b> ^	<b>92.0KB</b> ^	<b>7</b> ^

## ECOINDEX

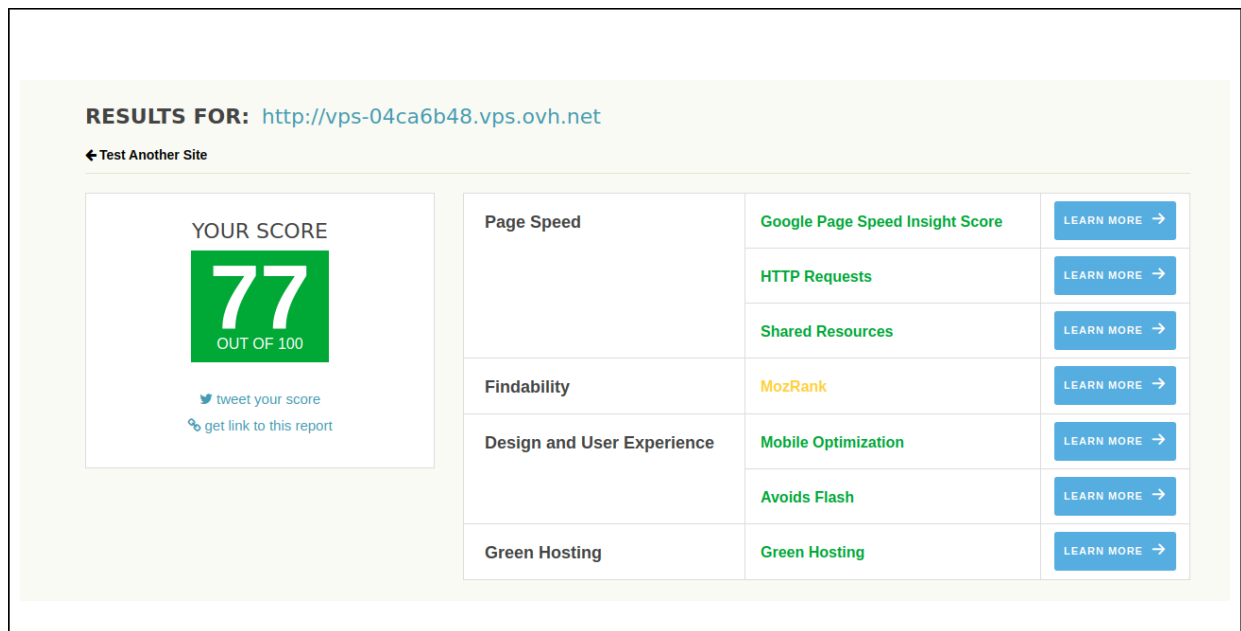
SCORE (Performance environnementale / Environmental performance ) : 85,2/100  
SCREENSHOT 18h30 05/11



## ECOGRADER

SCORE : 77 / 100

SCREENSHOT 18h30 05/11



## SONARQUBE

GITHUB URL : <https://github.com/Abdelhamidhenni/59>

## Conception générale - General conception

Avez-vous réussi à finaliser votre projet ? Did you manage to finish your project ?  
Oui Yes / Non No

Si non, pourquoi et quels éléments sont manquants ? if not, why and what is missing ?

bcp de tps à identifier les données à consommer et les agréger

- dimension accès information manquante
- comparaisons par département
- comparaisons par région

## Conception technique - Technical conception

Quel langage avez-vous choisi et pourquoi ? which language did you use and why ?

Technologies :

### **Backend : NestJs**

La technologies javascript est asynchrone. (pas de blocage lors d'un calcul par exemple).

L'utilisation du CPU est donc optimisé par design.

Nodejs nécessite peu de ressource. Il n'est donc pas nécessaire d'avoir un processeur performant pour que l'application fonctionne. Il est donc possible d'héberger des applications sur un environnement minimaliste. Une infrastructure vieillissante devrait quand même être en mesure d'exécuter l'application. (Moins d'obsolescence)

### **Frontend : ReactJs**

L'utilisation d'une architecture SPA (single page application) permet de faire transiter séparément les données d'affichages statiques et les données dynamiques. Ainsi il n'y a pas de donnée redondante de desing qui transite lors d'une nouvelle requête. Notons que cela limite aussi la sollicitation du server et donc réduit potentiellement la consommation du cpu.

### **Design : MaterialUI**

L'utilisation d'un librairie minifier permet de répondre à moindre empreinte un design dans l'air du temps à l'application. La librairie gère nativement les aspects responsive design. Elle permet aussi de répondre aux questions d'accessibilité pour le daltonien par exemple. L'utilisation d'un charte graphique neutre renforce cette aspect.

### **Stockage : Postgres :**

Les informations stockées en base ont subis des transformations avant leurs insertions. Ainsi une partie des calculs n'est réalisée qu'une seul fois. Moins de calcul c'est donc moins de consommation cup.

Le typage des champs de la base de donnée ont été choisis

- La volumétrie des tables est donc optimisée => baisse de la volumétrie

- Les requêtes sont plus performantes => gain de temps de calcul cpu

Architecture :

Docker container :

L'utilisation de docker facilite la virtualisation des environnements.

Elle est propice à l'utilisation d'un environnement cloud. La loi de Moore n'explique pas à elle seule que la courbe de l'empreinte carbone de notre environnement informatique ne soit pas exponentielle. L'utilisation d'infrastructure mutualisée est un des axes qui permet de réduire cette empreinte.

Server Node

Reverse Proxy NGinx

Le reverse proxy permet de compresser les flux envoyés.

Il est aussi configuré pour :

- garder en cache les requêtes afin d'éviter les re-calculs
- indiquer au navigateur de conserver les éléments statiques du site pour limiter le trafic réseau

Comment avez-vous optimisé vos requêtes ? How did you optimize the query ?

Cet encart est en partie répondu ci dessus :

- compression des requêtes transmises
- découpage de l'architecture de la solution pour ne renvoyer que le nécessaire
- mise en cache côté client et côté serveur ainsi : un client ne demandera que les informations qu'il n'a pas reçus. Le serveur conserve en mémoire ce qu'il a déjà utilisé pour limiter le calcul et les accès bases.

Par ailleurs, l'utilisation du principe de lazy loading permet de limiter les informations qui transitent avec le client web. Ainsi le client n'a pas besoin d'un serveur performant pour consulter le site (moins d'obsolescence matériel) et le trafic réseau est aussi diminué (moins de serveurs sollicités)

Dans le cas présent, la recherche d'une ville est en auto complète côté client. La recherche ne s'exécute qu'après la saisie des 3 premiers caractères et seuls 10 propositions sont disponibles à l'utilisateur. Il y a ici un gain en terme d'expérience utilisateur mais aussi sur le trafic réseau.

## Conception fonctionnelle - Functional conception

Avez-vous choisi d'utiliser un outil de représentation graphique ? Did you use a graphical representation ? Oui Yes / Non No

Si oui pourquoi ? if yes, why ?

Si non pourquoi ? if not Why ?

## Design

Expliquez en quelques mots les choix réalisés au niveau du design du site?  
Explain your design choices ?

## Accessibilité

Qu'avez-vous mis en place pour le respect de l'accessibilité du site? How did you manage the accessibility of your site ?

## QUESTIONS GÉNÉRALES - GENERAL QUESTIONS

Qu'est ce qui fait que votre site est éco-conçu? Why your solution is ecodesign ?

De part l'architecture et les technologies utilisées, le site limite au maximum son empreinte carbone global.  
Le site nécessite peu de ressource aussi bien côté client que côté server.

Avez-vous d'autres remarques pertinentes sur votre projet ? others comments on your project ?

