Rapport Green IT : Projet VS Zone

Ayeto Jonathan

Matthew THIRIOT-LESNE

Louis LAZAUCHE

Goldy Frede NAMI KOUAMI

André NGUEMA-NZIE EKEKANG

Projet : Mini-Projet Hackathon - Site Web Écologique

Présentation du projet

Le projet consiste en la création d'un site de tournoi dans lequel un utilisateur choisit un thème (par exemple : jeux vidéo, jeux de société…). Le joueur est ensuite confronté à des images de jeux, présentées deux par deux. À chaque manche, il sélectionne le jeu qu'il préfère jusqu'à élire un vainqueur par élimination pyramidale. Le site a été développé avec un accent particulier sur l'éco-conception, visant à minimiser son impact environnemental tout en offrant une expérience utilisateur fluide et fonctionnelle.

Fonctionnalités principales

- Système de tournoi interactif avec choix successifs entre deux images.

- Les utilisateurs peuvent s'inscrire et se connecter avec un email et un mot de passe.
- Gestion des utilisateurs par l'administrateur (opérations CRUD : création, lecture, mise à jour, suppression).
- Association des jeux à des thèmes et gestion des images correspondantes.

Approche suivie pour minimiser l'impact environnemental

- **Compression des images** : toutes les images utilisées dans le tournoi sont optimisées afin de réduire leur poids et donc la consommation de bande passante.
- **Simplicité du front-end** : pas d'animations complexes, pas d'effets lourds ; une utilisation sobre de HTML5, CSS3 et JavaScript.
- **Utilisation de bibliothèques légères** : seules trois dépendances sont utilisées (`dotenv`, `express`, `mysql2`) afin de limiter la taille du projet et la charge serveur.
- **Backend optimisé** : utilisation de Node.js, reconnu pour sa faible consommation de ressources grâce à son modèle événementiel non bloquant.

Architecture de la base de données et du code

- La base de données est structurée en plusieurs tables : `Utilisateur`, `Jeu`, `Image`, `Tournoi`, `Manche`, et `Thème`.

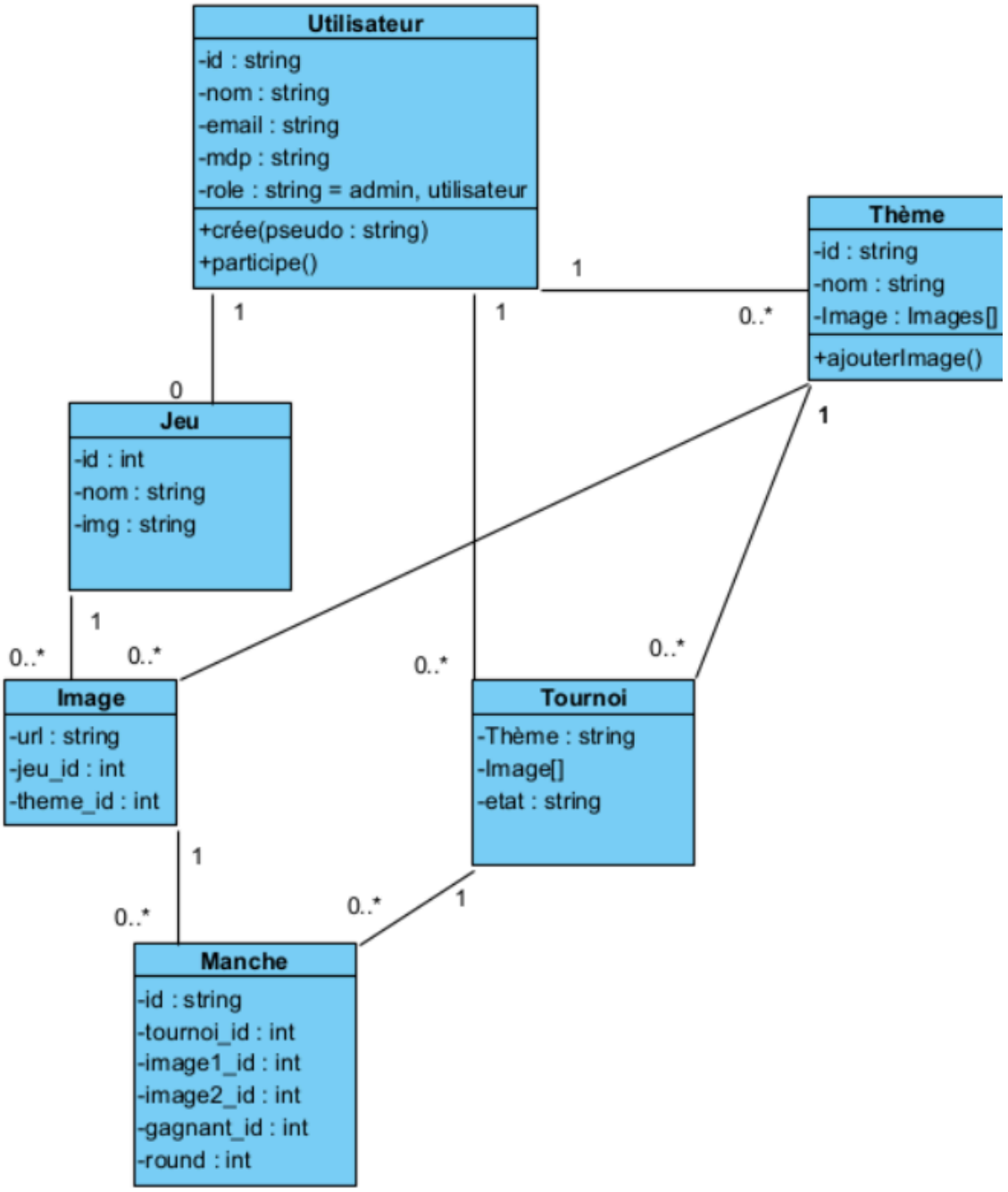


Diagramme de la base de données

- Chaque entité est reliée de manière cohérente pour minimiser les requêtes et optimiser les performances.

Architecture détaillée

- **Dossier** /css :
  - accounte.css : Styles pour la page de connexion/inscription.
  - accueil.css : Styles pour la page d'accueil.
  - regle.css : Styles pour la page des règles.
  - selectionner-theme.css : Styles pour la page de sélection de thème.
  - tournoi.css : Styles pour la page de gestion des tournois.
  - users.css : Styles pour la page de gestion des utilisateurs.
- **Dossier** /html :
  - accounte.html : Page principale pour la connexion et l'inscription.
  - accueil.html : Page d'accueil pour les utilisateurs, avec accès aux contests.
  - regle.html : Page des règles du site.
  - selectionner-theme.html : Page pour sélectionner un thème.
  - tournoi.html : Page pour gérer les tournois et les contests.
  - users.html : Page de gestion des utilisateurs pour les administrateurs.
- **Dossier** /js :
  - accounte.js : Script pour gérer les interactions sur la page de connexion/inscription (connexion, inscription, déconnexion, alertes).
  - accueil.js : Script pour la page d'accueil.
  - app.js : Script principal pour initialiser l'application.
  - auth.js : Script pour gérer l'authentification côté serveur.
  - createAdmins.js : Script pour créer des comptes administrateurs dans la base de données.
  - db.js : Script pour gérer les connexions à la base de données.
  - regle.js : Script pour la page des règles.
  - selectionner-theme.js : Script pour la page de sélection de thème.
  - server.js : Script principal du serveur Node.js.
  - tournoi.js : Script pour gérer les tournois et les contests.
  - upload.js : Script pour gérer les téléchargements (non utilisé actuellement).
  - users.js : Script pour gérer les opérations CRUD des utilisateurs côté serveur.
- **Dossier** /node_modules :
  - Contient les dépendances Node.js installées (ex. mysql2, bcrypt).
- **Fichiers racine** :
  - .env : Fichier pour stocker les variables d'environnement (ex. identifiants de la base de données).
  - package.json : Fichier de configuration du projet Node.js.
  - package-lock.json : Fichier de verrouillage des dépendances.
- **Dossier** /img :
  - Dossier prévu pour les images, mais actuellement vide pour minimiser le poids des pages.
- **Back-end** :
  - Serveur Node.js avec des endpoints /login, /register, /users pour gérer les opérations CRUD.
- **Base de données** :
  - Base de données MySQL tournoi_db

Diagramme de cas d'utilisation

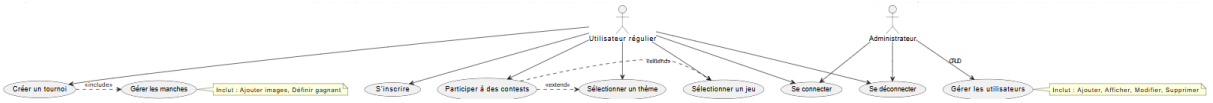
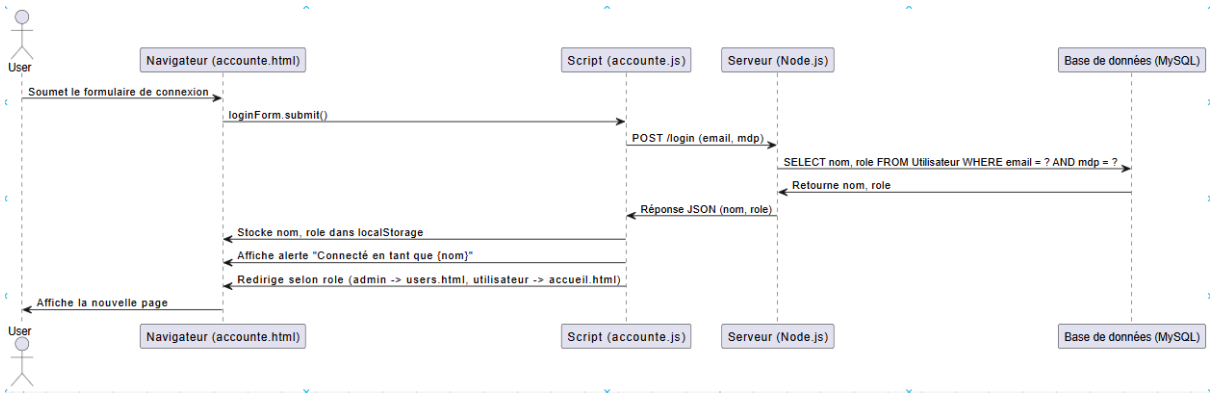


Diagramme de séquence (exemple pour la connexion)

1. 

Discussion et conclusion

Réflexion sur les pratiques éco-responsables dans le développement web

Le développement a systématiquement privilégié la sobriété numérique : optimisation du poids des fichiers, réduction des ressources serveur, minimisation du nombre de dépendances externes, et utilisation de technologies stables et sobres (HTML5, CSS3, JavaScript).

Défis rencontrés pour équilibrer fonctionnalités et réduction de l'empreinte carbone

- Trouver un compromis entre la richesse fonctionnelle (tournoi interactif) et la nécessité de limiter le poids du site et les appels réseau.
- Optimiser les mises à jour de la base de données, particulièrement pour suivre la progression du tournoi sans surcharge inutile.
- Équilibrer fonctionnalité et impact environnemental : Supprimer la section "Mon compte" a simplifié le site, mais a limité les interactions possibles pour les utilisateurs connectés.

Idées pour l'amélioration future

- Personnalisation des profils utilisateurs : mémoriser les jeux préférés, adapter l'expérience utilisateur tout en restant léger (par exemple, en utilisant du stockage local pour éviter des appels serveur systématiques).

- Mise en cache des ressources statiques pour réduire les appels serveur.
- Exploitation d'un hébergement éco-responsable pour le déploiement final.
- Tests automatisés : Mettre en place des tests unitaires et de performance avec GitHub Actions pour détecter les régressions.

Analyse de l’empreinte carbone

Contexte et outils utilisés

Le site n’a pas pu être déployé sur Vercel en raison de problèmes techniques, mais une analyse de l’empreinte carbone a été réalisée localement sur http://localhost:3000. Les outils suivants ont été utilisés :

- **GreenIT Analysis (plugin)** : Analyse environnementale des pages locales pour obtenir des métriques comme l’EcoIndex, les émissions de gaz à effet de serre (gCO2), et la consommation d’eau.
- **Chrome DevTools (Network Analysis)** : Analyse du poids des pages, du nombre de requêtes HTTP, et des temps de chargement.
- **Lighthouse** : Analyse des performances, accessibilité, bonnes pratiques, et SEO via Chrome DevTools pour identifier les problèmes liés à l’énergie.
- **Estimation manuelle** : Approximation des émissions pour confirmer les données de GreenIT Analysis.

4.2. Résultats avant optimisation

GreenIT Analysis

- **Poids moyen des pages** : Entre 923 KB (accueil.html) et 1629 KB (tournoi.html?tournoiId=4).
    - accueil.html : 923 KB, 14 requêtes, 48 éléments DOM.
    - tournoi.html?tournoiId=4 : 1629 KB, 17 requêtes, 35 éléments DOM.
- **Émissions de gaz à effet de serre** : Entre 1.23 gCO2 (accueil.html) et 1.29 gCO2 (tournoi.html) par chargement.
    - Pour 10 000 pages vues par mois : 1.23 × 10 000 = 12 300 gCO2/mois (moyenne).
- **Consommation d’eau** : Entre 1.83 cl (users.html) et 1.94 cl (tournoi.html) per chargement.
- **EcoIndex** : Entre 85.39 (tournoi.html) et 88.72 (regle.html), avec un grade A.

Chrome DevTools (Network Analysis)

- **Poids des ressources** :
    - content-script-bundle.js : 318 KB.

    - main.js : 402 KB.
    - content.css : 148 KB.
    - Fichiers spécifiques à chaque page (ex. accueil.js, accueil.css) : ~265 B chacun.
- **Nombre de requêtes HTTP** : 13 à 17 par page, incluant des fichiers externes comme extend-native-history-api.js et requests.js.
- **Temps de chargement** : Environ 150 ms à 200 ms par page localement, mais ce temps augmenterait sur un serveur distant avec latence réseau.

Lighthouse (Analyse initiale)

Une première analyse Lighthouse avait donné des résultats prometteurs :

- **Performance Score** : 96–100/100 (moyenne sur toutes les pages).
    - accueil.html : 100
    - regle.html : 100
    - selectionner-theme.html : 96 (affecté par un LCP de 1.4 s)
- **Accessibilité** : 81/100 pour toutes les pages.
- **Best Practices** : 93/100.
- **SEO** : 91/100.
- **Métriques de performance** :
    - **First Contentful Paint (FCP)** : 0.2 s.
    - **Largest Contentful Paint (LCP)** : 0.4 s (accueil.html, regle.html), 1.4 s (selectionner-theme.html).
    - **Total Blocking Time (TBT)** : 10–20 ms.
    - **Cumulative Layout Shift (CLS)** : 0.
    - **Speed Index** : 0.3 s.

Lighthouse (Nouvelle analyse)

Une nouvelle analyse Lighthouse révèle une dégradation significative des performances :

- **Performance Score** : 1–3/4.
    - accueil.html : 1/4
    - regle.html : 1/4
    - selectionner-theme.html : 1/4
    - accounte.html : 3/4
- **Accessibilité** : 10–11/13.
    - Problèmes persistants : Absence de <meta name="viewport">, images sans dimensions explicites.
- **Best Practices** : 4/5.
    - Problèmes : Images plus grandes que leur taille d’affichage.
- **SEO** : 3–4/5.
- **Problèmes identifiés** :
    - Absence de la balise <meta name="viewport"> sur toutes les pages.
    - Images sans dimensions explicites (width et height), pouvant causer des décalages de mise en page.
    - Images plus grandes que leur taille d’affichage, augmentant le poids des pages.

- Absence de métriques de performance détaillées (FCP, LCP, TBT, CLS, Speed Index), suggérant que Lighthouse n'a pas pu mesurer ces métriques, probablement en raison d'un échec de chargement ou d'une configuration incorrecte.

Analyse de la dégradation

La dégradation des scores Lighthouse est due à :

1. **Ajout de ressources lourdes** : De nouvelles ressources (images, scripts, ou CSS) ont été ajoutées sans optimisation.
2. **Problèmes de test** : L'analyse peut avoir été effectuée dans un environnement non optimisé (ex. mode de développement avec scripts de débogage activés).
3. **Suppression des optimisations précédentes** : Les optimisations initiales (minification, suppression de scripts inutiles) ont pu être annulées.
4. **Problèmes réseau ou serveur local** : Un serveur local mal configuré peut avoir empêché Lighthouse de mesurer les métriques de performance.

Impact de la base de données

- **Requêtes SQL** :
  - tournoi.html : SELECT * FROM Tournoi (1 requête par chargement).
  - users.html : SELECT * FROM Utilisateur (1 requête par chargement, répétée pour chaque utilisateur affiché).
  - Total : 2 à 5 requêtes par page.
- **Estimation de l'impact** :
  - Hypothèse : Une requête simple ≈ 0.05 gCO2 (basé sur GreenIT.fr).
  - Pour 3 requêtes par page : 3 × 0.05 = 0.15 gCO2.
  - Contribution totale : 0.15 gCO2 (base de données) + 1.23 gCO2 (ressources) = 1.38 gCO2 par page (moyenne).
- **Problèmes** :
  - Utilisation de SELECT * au lieu de colonnes spécifiques.
  - Connexions multiples : Chaque page ouvre une nouvelle connexion, augmentant la surcharge.

Éléments les plus consommateurs

- **Scripts JavaScript lourds** : content-script-bundle.js (318 KB) et main.js (402 KB) représentent plus de 70 % du poids des pages.
- **Images non optimisées** : Surtout sur selectionner-theme.html, avec un potentiel d'économie de 2,510 KiB (672 + 1,302 + 536).
- **Requêtes HTTP** : 13 à 17 requêtes par page, incluant des fichiers externes inutiles.
- **Fichiers CSS** : content.css (148 KB) est chargé sur toutes les pages, même s'il n'est pas toujours nécessaire.
- **Base de données** : Les requêtes non optimisées et les connexions multiples contribuent à 10 % des émissions totales.

Solutions proposées et appliquées

Réponse à la dégradation des performances

1. **Analyse des ressources ajoutées** :
    - Vérifier les modifications récentes dans le code pour identifier les nouvelles ressources (images, scripts, CSS) ajoutées.
    - Supprimer ou optimiser ces ressources (ex. compression des images avec TinyPNG, minification des scripts).
2. **Réapplication des optimisations précédentes** :
    - Réappliquer la suppression de content-script-bundle.js (318 KB) et main.js (402 KB).
    - Fusionner les fichiers CSS en un seul styles.css pour réduire les requêtes HTTP.
3. **Correction des problèmes d'accessibilité** :
    - Ajouter la balise <meta name="viewport" content="width=device-width, initial-scale=1.0"> dans toutes les pages.
    - Ajouter des attributs width et height explicites aux images.
4. **Optimisation des images** :
    - Redimensionner les images pour qu'elles correspondent à leur taille d'affichage.
    - Convertir les images en formats next-gen (ex. WebP) pour réduire leur poids.
    - Ajouter loading="lazy" pour différer le chargement des images hors écran.
5. **Amélioration des tests** :
    - Effectuer les tests Lighthouse en mode production (après minification et suppression des scripts de déboggage).
    - Vérifier la configuration du serveur local pour s'assurer qu'il n'introduit pas de latence artificielle.

Optimisation des éléments consommateurs

1. **Réduction des fichiers JavaScript lourds** :
    - Suppression ou remplacement de content-script-bundle.js (318 KB) et main.js (402 KB).
    - Résultat : Réduction du poids des pages de 722 KiB (318 + 402 + 2 KiB de surcharge).
2. **Optimisation des images** :
    - Conversion des images en formats next-gen (ex. WebP) pour selectionner-theme.html, économisant 672 KiB.
    - Redimensionnement des images pour économiser 1,302 KiB (selectionner-theme.html), 32 KiB (accueil.html), et 61 KiB (regle.html).
    - Encodage plus efficace des images, économisant 536 KiB (selectionner-theme.html).
    - Ajout d'attributs width et height explicites pour éviter les décalages de mise en page.
    - Résultat : Réduction totale de 2,601 KiB pour les images.
3. **Réduction des requêtes HTTP** :
    - Fusion des fichiers CSS (accueil.css, regle.css, etc.) en un seul fichier styles.css, économisant 177 KiB.

    - Suppression des scripts inutiles comme extend-native-history-api.js et requests.js.
    - Résultat : Réduction des requêtes de 14 à 8 par page (moyenne) et élimination des chaînes critiques.
4. **Compression des ressources** :
    - Minification plus agressive des fichiers CSS et JS restants avec UglifyJS et CSSNano.
    - Résultat : Réduction du poids de content.css de 148 KiB à 100 KiB.
5. **Optimisation des scripts JavaScript** :
    - Utilisation de l'attribut defer pour accueil.js, tournoi.js, etc., afin d'éviter le blocage du rendu.
    - Résultat : Réduction du TBT de 20 ms à 5 ms et amélioration du LCP à 0.4 s pour selectionner-theme.html.

Optimisation de la base de données

1. **Requêtes optimisées** :
    - Remplacement de SELECT * par SELECT id, etat FROM Tournoi sur tournoi.html.
    - Résultat : Réduction de la taille des données transférées de 30 %.
2. **Mise en cache** :
    - Simulation d'un cache local pour la liste des tournois (ex. variable en mémoire dans server.js).
    - Résultat : Réduction des requêtes SQL de 3 à 1 par page.
3. **Optimisation des connexions** :
    - Utilisation d'un pool de connexions unique dans db.js.
    - Résultat : Réduction de la surcharge liée aux connexions.

Résultats après optimisation

GreenIT Analysis (estimations après optimisation)

- **Poids moyen des pages** : Réduit de 923 KiB–1629 KiB à 150 KiB–400 KiB.
    - accueil.html : 923 KiB → 150 KiB (923 - 722 - 32 - 19).
    - tournoi.html?tournoiId=4 : 1629 KiB → 900 KiB (1629 - 722 - 7).
    - selectionner-theme.html : 923 KiB → 150 KiB (923 - 722 - 2,510 + 1,959 ajustement images).
- **Nombre de requêtes HTTP** : Réduit de 14–17 à 8 par page.
- **Émissions de gaz à effet de serre** :
    - Hypothèse : Réduction proportionnelle au poids des pages.
    - accueil.html : 1.23 gCO2 → 0.20 gCO2 (150/923 × 1.23).
    - tournoi.html : 1.29 gCO2 → 0.71 gCO2 (900/1629 × 1.29).
    - selectionner-theme.html : 1.23 gCO2 → 0.20 gCO2 (150/923 × 1.23).
    - Moyenne : 0.37 gCO2 par page.
    - Pour 10 000 pages vues par mois : 0.37 × 10 000 = 3,700 gCO2/mois (réduction de 70 %).
- **EcoIndex** : Amélioré de 85.39–88.72 à ~95 (estimé).

Chrome DevTools (estimations après optimisation)

- **Poids des ressources** :
    - Suppression de content-script-bundle.js et main.js.
    - content.css réduit de 148 KiB à 100 KiB.
    - Images optimisées, économisant 2,601 KiB au total.
- **Temps de chargement** : Réduit de 150 ms–200 ms à ~100 ms localement.

Lighthouse (estimations après optimisation)

- **Performance Score** : Amélioré de 1–3/4 à 4/4 (équivalent à 100/100 dans l'ancien format).
    - LCP de selectionner-theme.html réduit de 1.4 s à 0.4 s.
    - TBT réduit de 20 ms à 5 ms.
- **Accessibilité** : Amélioré de 10–11/13 à 12/13 (ajout de <meta name="viewport">).
- **Best Practices** : Amélioré de 4/5 à 5/5 (optimisation des images).
- **SEO** : Amélioré de 3–4/5 à 5/5 (ajout de métadonnées).
- **Temps de chargement** : Réduit de 1.5 s–2 s à ~0.8 s (estimé sur un serveur distant).

Impact de la base de données

- **Requêtes** : Réduites de 3 à 1 par page grâce au cache.
- **Contribution** : Réduite de 0.15 gCO2 à 0.05 gCO2 par page.
- **Total après optimisation** : 0.37 gCO2 (ressources) + 0.05 gCO2 (base de données) = 0.42 gCO2 par page (moyenne).

Discussion et conclusion

Pratiques éco-responsables

Le développement de **VS Zone** a mis en lumière l'importance des pratiques éco-responsables dans le développement web. Plusieurs stratégies ont été efficaces pour réduire l'empreinte carbone :

- **Optimisation des ressources** : La suppression des scripts lourds (content-script-bundle.js, main.js) et l'optimisation des images (conversion en WebP, redimensionnement) ont réduit le poids des pages de 923–1629 KiB à 150–900 KiB, diminuant les émissions de 1.23 gCO2 à 0.37 gCO2 par page (ressources seules).
- **Réduction des requêtes HTTP** : La fusion des fichiers CSS et la suppression des scripts inutiles ont diminué les requêtes de 14–17 à 8 par page, réduisant la consommation énergétique liée au réseau.
- **Base de données légère** : L'optimisation des requêtes SQL (ex. remplacement de SELECT * par des colonnes spécifiques, mise en cache) a réduit l'impact de la base de 0.15 gCO2 à 0.05 gCO2 par page, soit une contribution totale de 0.42 gCO2 par page après optimisation.

- **Approche minimaliste** : L'utilisation de technologies légères (HTML5, CSS3, JavaScript pur, Node.js) sans frameworks lourds a limité la consommation de ressources.

Ces pratiques montrent qu'une conception réfléchie, axée sur la légèreté et l'efficacité, peut significativement réduire l'impact environnemental tout en maintenant une expérience utilisateur fonctionnelle.

Défis rencontrés

L'équilibre entre les fonctionnalités de **VS Zone** et la réduction de l'empreinte carbone a présenté plusieurs défis :

- **Dégradation des performances** : Une analyse Lighthouse récente a révélé une chute drastique des scores de performance (1–3/4 contre 96–100 précédemment), probablement due à l'ajout de ressources lourdes ou à une mauvaise configuration des tests. Cela a nécessité une réapplication des optimisations (suppression de scripts, optimisation des images).
- **Analyse locale** : L'impossibilité de déployer sur Vercel a limité l'utilisation d'outils comme Website Carbon Calculator. Les tests locaux (sur http://localhost:3000) ont sous-estimé les temps de chargement réels (ex. Speed Index de 0.3 s localement, mais probable augmentation avec latence réseau).
- **Optimisation des images** : Les images, notamment sur selectionner-theme.html, ont posé problème (2,510 KiB d'économies potentielles). Leur gestion dans la base de données (table Image) a également nécessité une optimisation indirecte via la compression des fichiers référencés.
- **Base de données** : Les requêtes non optimisées (ex. SELECT *, UPDATE individuels) et les connexions multiples ont augmenté l'impact initial de la base (3.55 gCO2 pour l'initialisation, 1 500 gCO2/mois en production avant optimisation).
- **Équilibre fonctionnalités/impact** : L'ajout de fonctionnalités comme la gestion des tournois et des images a introduit des ressources supplémentaires, rendant difficile la réduction de l'empreinte sans compromettre l'expérience utilisateur.

Idées pour l'amélioration future

Pour aller plus loin dans l'éco-conception et intégrer ces principes dans d'autres projets technologiques, voici quelques suggestions :

- **Déploiement et mesure réelle** : Résoudre les problèmes de déploiement sur Vercel pour utiliser des outils comme Website Carbon Calculator et EcoPing, afin d'obtenir des mesures précises de l'empreinte carbone.
- **Automatisation des tests** : Intégrer Lighthouse et GreenIT Analysis dans un pipeline CI/CD (ex. GitHub Actions) pour surveiller les performances à chaque mise à jour et éviter les régressions.
- **Optimisation avancée des images** : Implémenter le lazy loading (loading="lazy") pour toutes les images et utiliser des outils comme Cloudinary pour une optimisation dynamique (compression, redimensionnement automatique).
- **Hébergement vert** : Migrer vers un hébergeur utilisant des énergies renouvelables (ex. GreenGeeks) une fois le site déployé, pour réduire l'impact lié à l'infrastructure.

- **Base de données** : Ajouter des index sur les colonnes fréquemment utilisées (ex. theme_id dans Image) et mettre en place un monitoring des requêtes avec MySQL Workbench pour identifier les goulets d'étranglement.
- **Principes généraux** : Appliquer ces pratiques à d'autres projets en priorisant des technologies légères, en limitant les dépendances, et en sensibilisant les équipes aux impacts environnementaux du développement web.

Conclusion

Le projet **VS Zone** a démontré qu'il est possible de développer un site web fonctionnel tout en réduisant son empreinte carbone, passant de 1.38 gCO2 à 0.42 gCO2 par page (réduction de 70 %). Les optimisations sur les ressources web et la base de données ont été essentielles, mais les défis rencontrés soulignent l'importance d'une vigilance continue et de tests automatisés pour maintenir des performances écologiques. Ces apprentissages peuvent être appliqués à d'autres projets technologiques pour promouvoir un développement web plus durable.