Contraintes techniques

Technologies

Ces technologies permettent de couvrir tous les aspects du développement full-stack (frontend, backend, base de données, API).

Frontend

SPA : React + Typescript: composants réutilisables, écosystème riche, compatibilité avec le mobile-first. CSS : DaisyUI inclut une bonne base d'accessibilité. Les composants sont construits avec les bonnes pratiques (balises sémantiques, contraste). Il faut néanmoins ajouter les attributs "aria-label" et les "alt".

Backend (API)

Node.js + Express: Légereté, rapidité de développement, cohérence avec Typescript en frontend. l'API Restful en backend permet une séparation claire des responsabilités backend/frontend et une séparation du déploiement. On peut faire évoluer backend et frontend séparément. Plus facile à maintenir et à débugger. Pour tester les endpoints, nous utiliserons Postman ou RestClient. ORM : Un ORM permet de gérér les opérations CRUD simplement. Nous utiliserons Sequelize.

Base de données

PostgreSQL : robuste, sécurisé, gère bien les relations (utilisateurs, challenges, votes, participations).

Sécurité

Authentification

JWT pour gérer les sessions utilisateurs. Standardisé, compatible avec React/Node. Pas de stockage côté serveur : idéal pour les API REST. Chaque requête est indépendante. Facile pour le frontend : le token est stocké en local et envoyé dans chaque requête. Utilisation de tokenCSRF pout contrer les attaques CSRF.

Protection des données

- Hashage des mots de passes: argon2
- Variables d'environnement pour les secrets (clés API, etc.) via un fichier .env

Sécurité des API

- Limitation des requêtes (express-rate-limit); CORS configuré pour n'autoriser que le domaine du frontend.
- Validation des données entrantes (ne pas faire confiance aux données venant du client) avec ZOD et sanitizer-HTML pour la protection contre les attaques XSS.

Failles courantes

Contrôle d’accès défaillant

Risque : un utilisateur accède à des ressources non autorisées.

Mesures: vérifier dans le backend que l'ID de l'utilisateur dans le token JWT correspond à l'ID de la ressource demandée.

Défaillances cryptographiques

Risque : exposition de données sensibles (mots de passe, tokens, données personnelles) en raison d'un chiffrement insuffisant ou inexistant.

Mesures: utiliser argon2 (via la bibliothèque argon2) pour hacher les mots de passe.

Injection SQL

Risque : exécution de code malveillant via des entrées utilisateur non validées.

Mesures:

- Utiliser un ORM Sequelize ou des requêtes paramétrées pour éviter les injections SQL.
- Valider et sanitizer les entrées avec zod.

Défaillances d'identification et d'authentification

Risque : faiblesses dans les mécanismes de connexion (ex : mots de passe faibles, session hijacking).

Mesures:

- implémenter JWT avec une durée de vie courte (ex : 15 min) et un refresh token.
- Exiger une complexité minimale des mots de passe (8+ caractères, majuscules, chiffres).

Uploads sécurisés

Ajout d'un lien vers une plateforme sécurisée (Youtube)

Déploiement

Le déploiement du projet s'effectue selon les étapes suivantes :

Frontend (Vercel)

- Le code du front est hébergé sur GitHub et connecté à Vercel.
- Chaque push sur la branche principale déclenche automatiquement un build et déploiement sur Vercel.
- Les variables d'environnement (API_URL, etc.) sont configurées dans le dashboard Vercel.
- L'URL du front est générée et mise à jour à chaque déploiement.

Backend (Render)

- Le backend (Node/Express/API) est hébergé sur Render.
- Les sources sont synchronisées depuis le dépôt GitHub pour chaque mise à jour.
- Les variables d'environnement (DATABASE_URL, etc.) sont configurées dans Render.

- Le déploiement se fait automatiquement à chaque push sur la branche main (ou manuellement via le dashboard).

Responsive

DaisyUI : bibliothèque simple et rapide à utiliser, basée sur Tailwind CSS, qui intègre automatiquement les bonnes pratiques du responsive design. Les composants sont déjà optimisés pour s'adapter parfaitement à toutes les tailles d'écran (mobile, tablette, ordinateur).

Accessibilité

- Utiliser des balises sémantiques HTML correctes (`<button>, <label>, <header>, <nav>, <main>, <article>` etc.).
- Respecter l'ordre des balises `<h1>, <h2>,<h3>`etc.(une seule balise `<h1>` et une seule balise `<main>` par page). En effet, les lecteurs d'écran utilisent les titres pour naviguer dans la page et la sémantique HTML.
- Attributs aria-label pour les boutons/liens sans texte visible (ex: icônes seules) et alt pour les images et les videos.
- Contraste suffisant entre texte et fond : ratio de contraste d'au moins 4,5:1 pour le texte normal et 3:1 pour le texte large (WCAG AA).
- Formulaires bien structurés avec `<label>` associés aux `<input>`(via attribut for/id). Ajouter des messages d'erreur clairs et accessibles..
- Vidéos : ajouter des sous-titres.
- Tests : Lighthouse et utiliser un lecteur d'écran (NVDA sur Windows, VoiceOver sur Mac).

RGPD et mentions légales

Dans le footer, un lien vers la page des mentions légales :

- Identité de l'éditeur du site (projet o'clock) + email de contact

Politique de confidentialité (conforme RGPD)

C'est le document clé du RGPD : Qui traite les données : identité de l'entreprise Quelles données collectées : formulaires, cookies, analytics, etc. Pourquoi les collecter : finalités (contact, statistiques, marketing...) Durée de conservation : combien de temps on garde les données Droits des utilisateurs : accès, rectification, suppression, opposition, portabilité Bases légales : consentement, intérêt légitime, obligation légale, etc. Sous-traitants : si vous utilisez des services tiers (Google Analytics, Mailchimp...) Cookies : consentement avec choix d'accepter/refuser

Pratique

Créer deux pages distinctes et facilement accessibles (en pied de page). Des générateurs gratuits existent à adapter à notre situation. La CNIL propose aussi des modèles.

Versionning

Utilisation de gitHub avec plusieurs branches à l'ajout de nouvelles fonctionnalités. Travail collaboratif facilité. Convention de nommage des commit : en anglais avec préfixe Feat (fonctionnalité), Fix (debugg),

Doc (ajout de documents).

API

https://www.igdb.com/api L'api IGDB, gérée par Twitch, permet d'accéder à une large base de données sur les jeux vidéo : titres, genres, plateformes, éditeurs, studios, notes, vidéos, captures d'écran, etc.

SEO

- Structure du site optimisée : balises sémantiques HTML (

  ,

  ,

  ,

  ,

  ), titres clairs et hiérarchisés.

- URLs lisibles et pertinentes, contenant les mots-clés importants.
- Rapidité de chargement des pages.
- Site 100 % responsive et adapté au mobile (mobile-first).
- Utilisation de balises meta :