État de l'Art Frontend pour R3MOB - Publications Scientifiques

Analyse réalisée en 2025 pour l'amélioration et l'optimisation du frontend

Contexte du Projet

R3MOB est un réseau de recherche pluridisciplinaire porté par Bordeaux INP, avec un site web existant développé en **React.js** depuis 2023.

Architecture Actuelle (2024)

- **Frontend** : React.js avec architecture MVC
- **Backend** : Express.js + MySQL
- **Authentification** : Système de tokens
- **Fonctionnalités** : Gestion d'événements, projets, chercheurs, ressources (publications + vidéos)

Problématiques Identifiées

D'après l'audit en cours, plusieurs points critiques nécessitent une refonte :

Problèmes de Performance

- Trop de re-rendus (notamment dans les filtres)
- Appels API non optimisés (boucles infinies dans `useEffect`)
- Filtrage côté client lent sur gros datasets
- Pas de mémorisation (`useMemo`) des calculs coûteux

Problèmes de Code

- Gestion d'erreurs insuffisante (alert au lieu d'UI)
- Code redondant dans les handlers
- `useEffect` mal optimisés
- Pas de validation globale des formulaires

Nouveaux Besoins (Publications Scientifiques)

- Import de publications via APIs (Crossref, Semantic Scholar)
- Classification automatique par thèmes/sous-thèmes
- Export vers TheBrain pour la visualisation en graphe de connaissances
- Interface de recherche avancée

1. Technologies Frontend Principales

1.1 React.js (Recommandé)

Avantages :

- Écosystème mature et très riche en composants
- Excellent pour les interfaces complexes avec beaucoup d'interactions
- Intégration native avec les APIs REST (Crossref, Semantic Scholar)
- Composants réutilisables pour l'affichage des publications
- Performance élevée avec le Virtual DOM
- Support TypeScript natif pour la robustesse du code
- Nombreuses librairies spécialisées (`react-virtualized` pour les grandes listes)

Inconvénients :

- Courbe d'apprentissage plus élevée
- Bundle size plus important
- Nécessite une configuration webpack/build plus complexe

1.2 Vue.js 3 (Alternative viable)

Avantages :

- Syntaxe plus simple et intuitive
- Excellent système de réactivité pour les filtres/recherches
- Composition API parfaite pour les composants de publication
- Performance légèrement supérieure à React
- Taille de bundle plus petite

Inconvénients :

- Écosystème moins mature pour les composants scientifiques
- Moins de ressources/exemples pour les interfaces académiques
- Communauté plus petite

Mais je préfére qu'on reste sur de la syntaxe Javascript/React puisqu'une grande base a déja été faite depuis 2023

En ce qui concerne l'optimisation des appels API existants , on pourrait intégrer *React Query*

Avantages de son utilisation:

- Cache automatique des requêtes APIs
- Synchronisation automatique avec Crossref/Semantic Scholar
- Gestion optimisée des états de chargement
- Retry automatique et gestion d'erreurs
- Parfait pour les données académiques souvent volumineuses

Inconvénients :

- Dépendance supplémentaire

Pistes d'amélioration faisables

Numéros de port

- Utiliser un fichier de configuration (ex. `.env`) pour définir les ports, par exemple : `PORT_SERVEUR=3001`, afin de faciliter la maintenance sans modifier le code source.
- Importer cette variable dans le code (ex. via `process.env.PORT_SERVEUR`) plutôt que d'utiliser une valeur codée en dur (hardcoded).

Affichage

- Certaines images apparaissent déformées (étirées) : prévoir un redimensionnement ou une contrainte CSS adaptée.
- Certaines rubriques ne s'affichent pas correctement .
- Des éléments dans l'en-tête du site sont mal alignés : à revoir avec un système de grille ou flexbox.

Documentation

- Ajouter des instructions claires pour le démarrage de la base de données **MySQL** (ex. commandes de création, configuration des accès).
- Fournir un conteneur **Docker** prêt à l'emploi avec configuration de ports dédiée pour les environnements de développement/test.

Langues

- Proposer une version anglaise du site pour toucher un public international, comme le font déjà R3IA et R3 Tesna (référence : état de l'art 2023).
- Ajouter un sélecteur de langue **FR/EN** visible (par exemple en haut de page).
- S'assurer que la traduction est complète et fidèle pour l'ensemble du contenu.

SEO (Search Engine Optimization)

- Optimiser le référencement naturel du site en :
    - intégrant des **mots-clés pertinents** dans les titres, descriptions et contenus,
    - ajoutant des balises `meta` bien structurées.
- Créer un **sitemap** pour aider les moteurs de recherche à indexer les pages du site :
    - **Sitemap XML** (recommandé) : facile à lire par les robots, généré automatiquement par des outils ou frameworks.
    - **Sitemap HTML** : plus difficile à maintenir manuellement.
- Pour lister les pages accessibles dynamiquement (routes internes), ce script JavaScript peut être utilisé directement dans la console du navigateur :

```
(function() {
  const links = Array.from(document.querySelectorAll('a[href]'));
  const internalLinks = links
    .map(a => a.href)
    .filter(href => href.includes(window.location.hostname))
    .map(href => new URL(href).pathname)
    .filter((path, index, arr) => arr.indexOf(path) === index)
    .sort();
  console.log('Routes trouvées:');
  internalLinks.forEach(path => console.log(path));
})();
```

Importation et gestion des publications côté frontend

Parsing de fichiers

Le frontend doit permettre l'import de publications via des fichiers aux formats **BibTeX**, **XML**, ou **JSON**.

- Utiliser des modules JavaScript comme `bibtex-js` ou `xml-js` pour parser ces fichiers directement **côté client**.
- Une fois parsés en **JSON**, les données peuvent être envoyées à la base SQL via l'**API backend**.

Recherche et enrichissement

Le frontend peut permettre à l'utilisateur :

- de saisir un **DOI** ou des **mots-clés**,
- puis d'interroger l'API **Crossref** (et éventuellement **Semantic Scholar**) pour récupérer les **métadonnées** des publications.

> *⚙ L'enrichissement (récupération des auteurs, thématiques, etc.) peut être déclenché côté serveur, mais une partie de la logique de récupération et **d'affichage** s'effectue côté **client**.*

Affichage des publications

Les publications importées sont affichées :

- sous forme de **listes filtrables** (par **titre**, **auteur**, **thématique**, etc.),
- avec **pagination** et **fonctionnalité de tri**,
- suivant le modèle déjà utilisé pour les projets et chercheurs.

⚠ Gestion des erreurs et cas limites

L'interface doit prendre en compte plusieurs scénarios :

- DOI invalide ou publication introuvable
- Métadonnées manquantes ou incomplètes
- Affichage de **messages d'erreur clairs** et gestion de **fallbacks** pour garantir une bonne expérience utilisateur

Solutions pour la mise à jour des dépendances

Dans le cadre de mon analyse pour le projet **R3MOB**, j'ai identifié plusieurs outils et pratiques permettant d'automatiser la gestion des dépendances et de sécuriser l'environnement JavaScript.

Outils automatisés recommandés

1. `npm audit` (solution native)

- Analyse des vulnérabilités connues
- Rapport d'évaluation avec recommandations

```
npm audit          # identifier les failles
npm audit fix      # corrections automatiques
npm audit --production  # cibler uniquement les dépendances de prod
```

2. Outils complémentaires

- `npm-check-updates` : met à jour les versions dans `package.json`
- **Snyk** : monitoring continu des failles de sécurité
- **Dependabot** : détection et mise à jour automatique des dépendances

---

Solutions pour la gestion des accès (Sécurité React moderne 2025)

1. Amélioration de l’authentification existante

- Mise en place de **refresh tokens** pour prolonger les sessions de manière sécurisée
- Intégration **OAuth 2.0 via ORCID**, standard dans le monde de la recherche
- Implémentation d'un **RBAC** (Role-Based Access Control)

2. Protection contre les vulnérabilités

- Utilisation de JSX avec `{}` pour prévenir les attaques **XSS**
- Validation des données **côté client ET serveur**
- Intercepteurs Axios pour la sécurisation des appels API
