Rapport Technique

Tableau Kanban Next.js/Redis

Application haute performance avec synchronisation en temps réel

Résumé Exécutif

Ce document présente une analyse technique détaillée du projet **Next.js/Redis Kanban Board**, une application web interactive de gestion de tâches exploitant une architecture moderne basée sur Next.js 16 et Redis comme base de données primaire. Le projet illustre l'utilisation efficace de Redis comme magasin d'état principal, capitalisant sur sa rapidité pour offrir une expérience utilisateur réactive.

1 Architecture Technique

1.1 Stack Technologique

1.1.1 Frontend

— **Next.js 16** avec App Router pour le rendu côté serveur

— **TypeScript** pour la sécurité des types

— **Tailwind CSS** pour le stylisme utilitaire

— **Radix UI** (via shadcn/ui) pour les composants accessibles

— Gestion d'état avec `useState`/`useEffect` de React

1.1.2 Backend

— **Redis** comme base de données en mémoire

— **ioredis** comme client Redis

— **API Routes** de Next.js pour les endpoints backend

1.2 Structure du Projet

```
app/
 api/                  # Route Handlers Next.js
    columns/         # API pour les colonnes
    tasks/           # API pour les tâches
 page.tsx             # Page principale
 layout.tsx           # Layout racine
components/
 kanban-board.tsx     # Logique du tableau (Drag \& Drop)
 kanban-column.tsx    # UI des colonnes
 ...                  # Autres composants
lib/
 redis.ts             # Configuration du client Redis
types/                   # Définitions TypeScript
```

1.3 Modélisation des Données Redis

L'application utilise deux structures de données principales dans Redis :

1.3.1 Colonnes

— **Structure** : Hash Redis `kanban:columns`

— **Clé** : ID de la colonne (ex : `col1, col2, col3`)

— **Valeur** : Objet JSON sérialisé contenant :

  — `id` : Identifiant unique

  — `title` : Titre de la colonne

  — `taskIds` : Tableau des IDs de tâches

1.3.2 Tâches

— **Structure** : Hash Redis `kanban:tasks`

— **Clé** : ID de la tâche (ex : `task1, task2`)

— **Valeur** : Objet JSON sérialisé contenant :

  — `id` : Identifiant unique

  — `content` : Contenu de la tâche

  — `columnId` : Référence à la colonne parente

Cette modélisation permet des opérations en O(1) pour la récupération et la mise à jour des données, garantissant des performances optimales.

2 Mécanisme de Synchronisation en Temps Réel

2.1 Stratégie Actuelle : Polling

2.1.1 Implémentation

Le client (`KanbanBoard`) récupère automatiquement les données les plus récentes de l'API toutes les **2 secondes** :

```
useEffect(() => {
    const interval = setInterval(() => {
        fetchColumns();
        fetchTasks();
    }, 2000);
    return () => clearInterval(interval);
}, []);
```

2.1.2 Avantages

— **Simplicité** : Aucune infrastructure complexe requise

— **Robustesse** : Tolérant aux déconnexions réseau

— **Serverless** : Compatible avec les environnements sans serveur

— **Performance** : Redis répond en <1ms, rendant le polling fréquent viable

2.1.3 Flux de Données

1. Client envoie une requête GET vers `/api/columns`
2. Handler Next.js interroge Redis via `ioredis`
3. Redis retourne les données en mémoire

4. Les données sont sérialisées en JSON et renvoyées au client
5. Mise à jour de l'interface utilisateur avec les nouvelles données

2.2 Architecture pour les Futures Améliorations

2.2.1 Pub/Sub Redis

Pour une vraie synchronisation en temps réel :

```
// Serveur publie un événement
redis.publish('kanban:updates', JSON.stringify({
    event: 'task:moved',
    data: { taskId, fromColumnId, toColumnId }
}));

// Clients souscrits reçoivent les mises à jour
redis.subscribe('kanban:updates', (message) => {
    // Mettre à jour l'UI en temps réel
});
```

2.2.2 WebSockets

— Serveur WebSocket séparé ou service comme Pusher/Ably
— Abonnement aux canaux Redis Pub/Sub
— Diffusion des mises à jour aux clients connectés

3 Performance et Évolutivité

3.1 Métriques Clés

Métrique	Valeur
Temps de réponse Redis	< 1 ms
Fréquence de synchronisation	2000 ms
Taille maximale des données	Mémoire Redis disponible
Concurrence	Supportée via transactions Redis

Table 1 – Métriques de performance

3.2 Avantages de l'Architecture

1. **Vitesse** : Redis en mémoire offre des lectures/écritures ultra-rapides
2. **Simplicité** : Architecture monolithique avec séparation claire des responsabilités
3. **Maintenabilité** : Code TypeScript avec types stricts
4. **Portabilité** : Fonctionne sur tout environnement supportant Node.js et Redis

3.3 Limitations Actuelles

— **Persistance** : Données perdues en cas de redémarrage de Redis (sans RDB/AOF)
— **Évolutivité horizontale** : Redis monolithique (non clusterisé)
— **Sécurité** : Absence d'authentification/autorisation
— **Consommation mémoire** : Toutes les données résident en mémoire

4 Roadmap d'Amélioration

4.1 Court Terme (1-3 mois)

— Implémenter la persistance Redis avec RDB et AOF

— Ajouter l'authentification utilisateur via NextAuth.js

— Intégrer `dnd-kit` pour un drag-and-drop plus accessible

4.2 Moyen Terme (3-6 mois)

— Migrer vers Redis Cluster pour l'évolutivité horizontale

— Implémenter Redis Pub/Sub pour les mises à jour en temps réel

— Ajouter des WebSockets pour une synchronisation bidirectionnelle

4.3 Long Terme (6+ mois)

— Introduire une base de données relationnelle (PostgreSQL) pour la persistance à long terme

— Implémenter un cache multi-niveaux (Redis + CDN)

— Développer une application mobile React Native

5 Conclusion

Le projet **Next.js/Redis Kanban Board** démontre avec succès comment combiner les technologies modernes du web (Next.js, React, TypeScript) avec une base de données en mémoire performante (Redis) pour créer une application interactive et réactive.

L'architecture actuelle, bien que simple, fournit une base solide pour l'évolution vers une solution de production complète. La stratégie de polling, bien que n'étant pas du vrai temps réel, offre un compromis excellent entre simplicité, performance et maintenabilité.

Les améliorations futures, notamment l'intégration de Redis Pub/Sub et des WebSockets, positionneront l'application comme une solution compétitive dans le domaine des outils de collaboration en temps réel.