Stratégie de Test pour RealWorld

1. Contexte et Objectifs

Application sous test

- Nom : RealWorld (Conduit) - Clone de Medium
- Type : Application web full-stack avec API REST
- Stack : React + Redux / Node.js + Express / PostgreSQL

Objectifs de test

- Valider toutes les fonctionnalités selon les spécifications RealWorld
- Assurer la qualité de l'expérience utilisateur
- Vérifier la sécurité (authentification JWT)
- Tester les performances et la robustesse de l'API

2. Périmètre de Test

Fonctionnalités à tester

🔐 Authentification & Utilisateurs

- Inscription d'un nouvel utilisateur
- Connexion / Déconnexion
- Mise à jour du profil utilisateur
- Gestion des tokens JWT

🦸 Profils & Relations Sociales

- Consultation de profils publics
- Follow / Unfollow d'autres utilisateurs
- Affichage des followers/following

📝 Gestion des Articles

- Création d'articles (titre, description, contenu, tags)
- Lecture d'articles individuels
- Mise à jour d'articles existants
- Suppression d'articles
- Liste d'articles (pagination, filtres)
- Articles favoris (ajouter/retirer des favoris)
- Feed personnalisé (articles des utilisateurs suivis)

💬 Système de Commentaires

- Ajout de commentaires sur articles
- Suppression de commentaires
- Affichage des commentaires

🏷️ Tags

- Récupération des tags populaires
- Filtrage d'articles par tags

Environnements de test

- Développement : Docker local (ports 4100, 3004, 5433)
- Staging : À définir si nécessaire
- API de référence : https://api.realworld.show (pour validation)

3. Niveaux de test et types de tests

Tests Unitaires

- Frontend : Composants React, reducers Redux, utilitaires
- Backend : Services métier, contrôleurs, middlewares
- Base de données : Modèles Prisma, requêtes

Tests d'Intégration

- API : Tests des endpoints avec vraie base de données
- Frontend-Backend : Communication via HTTP
- Base de données : Transactions, migrations

Tests End-to-End

- Parcours utilisateur complets
- Tests cross-browser
- Tests de régression

Tests Non-fonctionnels

- Performance : Temps de réponse API, chargement pages
- Sécurité : Authentification, autorisation, injections
- Compatibilité : Navigateurs, responsive design

4. Approche de Test

Méthodologie

- Risk-based testing : Prioriser selon criticité business
- Shift-left : Tests précoces dans le développement
- Test-first : TDD pour composants critiques

Types de tests

- Smoke tests : Fonctions critiques après déploiement
- Tests de régression : Non-régression sur fonctionnalités existantes
- Tests exploratoires : Découverte de bugs non prévus

5. Outils et Frameworks

Tests Unitaires

- Frontend : Jest + React Testing Library
- Backend : Jest + Supertest
- Base données : Jest + Prisma Mock

Tests d'Intégration & API

- Postman : Collection fournie par RealWorld
- Newman : Exécution automatisée Postman
- Insomnia : Tests API manuels

Tests End-to-End

- Playwright : Tests E2E cross-browser (recommandé)
- Cypress : Alternative populaire
- Selenium : Si contraintes spécifiques

CI/CD

- GitHub Actions : Pipeline automatisé
- Docker : Environnements reproductibles

6. Critères d'Acceptation

Couverture de code

- Frontend : > 80% sur composants critiques
- Backend : > 90% sur services métier
- API : 100% des endpoints testés

Performance

- API : < 200ms pour 95% des requêtes
- Frontend : Time to Interactive < 3s
- Base de données : < 100ms pour requêtes simples

Qualité

- Zéro bug critique en production
- < 5% de bugs mineurs par release
- 100% des user stories validées

7. Gestion des Risques

Risques identifiés

- Sécurité : Failles JWT, injections SQL
- Performance : Montée en charge, pagination
- Données : Corruption, cohérence
- Intégration : Frontend-Backend, APIs tierces

Stratégies de mitigation

- Tests de sécurité automatisés (SAST/DAST)
- Tests de charge avec k6 ou Artillery
- Backup/restore de données de test
- Contracts tests (Pact) pour API

8. Métriques et Reporting

Métriques de test

- Test execution rate : % tests passés/échoués
- Defect density : Bugs par fonctionnalité
- Test coverage : Couverture code et fonctionnelle
- Mean Time to Recovery : Temps résolution bugs

Reporting

- Daily : Résultats tests automatisés
- Weekly : Dashboard qualité avec métriques
- Release : Rapport complet avec recommandations

9. Planning et Ressources

Phases

1. Préparation (1 semaine) : Setup environnements, outils
2. Test design (1 semaine) : Cas de test détaillés
3. Implémentation (2 semaines) : Scripts automatisés
4. Exécution (En continu) : Tests dans pipeline CI/CD

Rôles

- QA Lead : Stratégie et coordination
- Test Automation Engineer : Scripts automatisés
- Manual Tester : Tests exploratoires
- DevOps : Infrastructure de test

10. Maintenance et Évolution

Maintenance des tests

- Revue mensuelle : Pertinence des cas de test
- Refactoring : Optimisation scripts obsolètes
- Documentation : Mise à jour suite évolutions

Amélioration continue

- Retrospectives : Leçons apprises
- Veille : Nouveaux outils et pratiques
- Formation : Montée en compétence équipe