Dossier de Validation & Assurance Qualité - EasyBooking

Projet : Application de Gestion de Réservation de Salles

Auteurs : Groupe 11 - M1 Dev G2

Date : 14 Janvier 2026

Version : 1.0 (Release Candidate)

Table des Matières

1. Introduction et Périmètre

1.1 Contexte

Ce document présente la stratégie de qualification logicielle mise en œuvre pour le projet **EasyBooking**. L'objectif est de garantir la fiabilité, la sécurité et la performance de l'application avant sa mise en production, en conformité avec les standards de développement actuels.

1.2 Périmètre Fonctionnel

Les tests couvrent l'intégralité des fonctionnalités critiques de l'application :

- **Authentification** : Inscription, Login, Logout, Protection des routes.

- **Réservation** : Vérification des disponibilités, Création de réservation (avec gestion des conflits), Annulation, Historique.
- **Gestion des Salles** : Catalogue, Recherche, Détails techniques (capacité, équipements).

---

2. Architecture Technique et Outils

2.1 Organisation du Code

L'architecture des tests a été conçue pour séparer clairement les responsabilités, comme visible dans la structure du projet :

- `src/components/**/*.test.tsx` : Tests unitaires co-localisés avec les composants.
- `src/__tests__/integration/` : Tests d'intégration API et flux métiers complexes.
- `src/__tests__/security/` : Tests dédiés aux scénarios OWASP.
- `src/__tests__/perf/` : Scripts K6 pour les tests de charge.
- `src/test/` : Centralisation des Mocks (`mocks.ts`) et Fixtures (`fixtures.ts`) pour la maintenabilité.

2.2 Stack Technologique de Test

- **Moteur de Test** : Vitest 4.0 (Unitaires & Intégration).
- **Simulation DOM** : React Testing Library & jsdom.
- **Performance** : Grafana K6.
- **Qualité Code** : ESLint, Prettier, TypeScript 5.9.
- **Couverture** : @vitest/coverage-v8.

---

3. Stratégie de Test (ISTQB)

Notre approche repose sur la pyramide des tests, divisée en quatre niveaux complémentaires :

1. **Tests Unitaires (White-box)** : Validation isolée des composants UI (Navbar, Listes) et des fonctions utilitaires (`utils.ts`) sans dépendances externes.
2. **Tests d'Intégration (Grey-box)** : Validation des interactions entre le Frontend et l'API (Mockée), assurant que les flux de données (Login -> Redirection) fonctionnent.
3. **Tests de Sécurité (Black-box)** : Validation de la robustesse face au Top 10 OWASP (Injection, XSS, IDOR, Auth).
4. **Tests de Performance (Load Testing)** : Validation de la tenue en charge et des temps de réponse (SLA < 1000ms).

---

4. Plan de Test Détaillé

4.1 Scénarios Couverts

Le projet totalise **91 tests automatisés** couvrant les aspects suivants:

Niveau	Cible	Scénarios Clés	Fichiers Principaux

Niveau	Cible	Scénarios Clés	Fichiers Principaux
Unitaire	Composants UI	Rendu conditionnel, Props, États locaux	`Navbar.test.tsx`, `ReservationsList.test.tsx`
Intégration	API & Flux	Auth Flow, Booking Flow, Gestion erreurs HTTP	`api.test.ts`, `flow.test.tsx`
Sécurité	Backend	Routes 401/403, Injections, Dates passées	`protection.test.ts`
Performance	Infrastructure	Stress test login, dashboard, listing	`script.js` (K6)

4.2 Données de Test

Afin de garantir l'indépendance des tests, nous utilisons des jeux de données (Fixtures) et des Mocks centralisés pour simuler :

- L'authentification Supabase.
- Les réponses API (Succès/Erreur).
- La navigation (Next/Router).

---

5. Guide d'Exécution

Prérequis

```
npm install
# Installation de K6 requise pour les tests de perf (ex: winget install k6)
```

Commandes Principales

Objectif	Commande	Description
Qualification Complète	`npm run test`	Lance Unitaires + Intégration + Sécurité
Tests Unitaires/Inté	`npx vitest`	Mode watch pour le développement
Sécurité uniquement	`npx vitest security`	Focus sur les failles OWASP
Performance	`k6 run src/__tests__/perf/script.js`	Lance les 10 scénarios de charge
Rapport Couverture	`npm run coverage`	Génère le rapport HTML/JSON

---

6. Synthèse des Résultats (Rapport Qualité)

À la date du **14 jan. 2026**, voici l'état de la qualité logicielle d'EasyBooking :

6.1 Indicateurs Clés de Performance (KPI)

- ☑ **Taux de succès global** : **100%** (91 tests passés sur 91).
- ☑ **Couverture de code** : **90.74%** (Objectif > 80% atteint).
- ☑ **Sécurité** : **10/10** scénarios critiques validés.

6.2 Analyse de la Couverture

La couverture est excellente sur les parties critiques :

- **Composants UI** : 100%
- **Utils** : 100%
- **API Critiques** : > 80%

6.3 Performance (K6)

- **Latence p95** : **329.47ms** (Objectif < 1000ms respecté) ☑.
- **Taux d'erreur HTTP** : **0.00%** (Aucune erreur 500/400 sur les flux nominaux) ☑.
-

Note : Certains checks de validation de contenu K6 (parsing HTML/JSON) échouent partiellement et nécessiteront un ajustement des scripts, mais la performance serveur est validée.

---

7. Bilan et Recommandation

7.1 Verdict

Sur la base des résultats obtenus (stabilité fonctionnelle parfaite, sécurité validée, performance serveur conforme), le verdict est :

🚀 **GO pour la mise en production**.

7.2 Points d'Attention & Risques

Bien que l'application soit stable, deux points mineurs sont à surveiller :

1. **Scripts K6** : Affiner les assertions de contenu pour éliminer les faux positifs dans les rapports de performance.
2. **API Secondaires** : Augmenter la couverture de test sur les routes administratives (actuellement ~65%).

7.3 Conclusion

Le projet EasyBooking respecte les standards de qualité industrielle. La stratégie de test mise en place (séparation Unit/Intégration, automatisation, couverture élevée) assure une maintenabilité forte pour les évolutions futures.

Groupe 11 - M1 Dev G2