Rapport de Synthèse Qualité Logicielle

Projet : EasyBooking - Application de Réservation de Salles
Groupe: Arthur ZACHARY, Alexy-Alan HEMERY et Jack YE

Lien github: https://github.com/K0ola/EasyBooking

1. Introduction et Stratégie de Test

Ce rapport synthétise les activités d'assurance qualité menées sur l'application de réservation de salles. L'objectif était de garantir la fiabilité du module critique d'authentification, la cohérence des API, la fluidité des parcours utilisateurs et la tenue à la charge du serveur.

Nous avons adopté une approche en quatre niveaux :

1. **Tests Unitaires :** Isolation de la logique métier critique (Authentification JWT).
2. **Tests d'Intégration :** Validation des contrats d'interface (API HTTP).
3. **Tests Fonctionnels :** Simulation de scénarios utilisateurs complets (E2E).
4. **Tests de Performance :** Vérification des temps de réponse et de la stabilité.

2. Analyse Détaillée par Niveau de Test

2.1. Tests Unitaires (Module Auth) - (Jack YE)

Focus : Sécurité et logique interne du token JWT.
Outils : Jest, Mocks.

Le module d'authentification a été testé en isolation totale. Nous avons validé 15 points de contrôle (UNIT-1 à UNIT-15) répartis en deux blocs :

- **Génération de Token (generateToken) :**
  - Vérification de la structure du JWT (Header, Payload, Signature).
  - Validation de l'inclusion des données utilisateurs (ID, Email).
  - **Contrôle strict de l'expiration** (calcul précis de la durée de 7 jours).

- **Middleware d'Authentification (authenticateToken) :**
  - Rejet des requêtes sans header Authorization.
  - Rejet des tokens invalides, expirés ou mal signés.
  - Vérification du format Bearer.
  - Attachement correct du profil utilisateur à l'objet req.

Conclusion : La logique de sécurité gère correctement les cas d'erreurs (401/403).

2.2. Tests d'Intégration API - (Alexy-Alan HEMERY)

Focus : Codes de statut HTTP, Headers, Sécurité API.

Quantité : 22 tests (API-1 à API-22).

Ces tests valident que l'API respecte les standards REST et réagit correctement aux entrées externes :

- **Conformité HTTP :** Validation des codes retours (200 OK, 201 Created, 400 Bad Request, 404 Not Found).
- **Sécurité des Données :**
    - Vérification que les mots de passe ne sont **jamais** renvoyés dans les réponses JSON.
    - Validation du blocage des injections SQL basiques.
    - Vérification des validations de format (Email invalide, mot de passe trop court).

- **Structure des Réponses :** Garantie que les endpoints renvoient toujours du JSON (Content-Type: application/json).

2.3. Tests Fonctionnels (Scénarios Métier) - (Arthur ZACHARY)

Focus : Parcours utilisateur de bout en bout ("Happy Path" et gestion des erreurs). Outils : Requêtes HTTP (simulation client).

Les tests fonctionnels ont permis de valider le comportement global de l'application à travers 4 grands scénarios réalistes, couvrant l'ensemble des flux critiques :

- **Scénario 1** : Gestion de Compte Utilisateur
    - Validation du flux d'inscription (`POST /register`) et de la création des ressources.
    - Vérification de la connexion (`POST /login`) et de la récupération du Token.
    - Confirmation de l'accès aux routes protégées (Profil utilisateur) avec le token généré.

- **Scénario 2** : Cycle de Réservation Complet
    - Navigation complète : Liste des salles ➔ Détails ➔ Vérification disponibilité.
    - Création réussie d'une réservation sur un créneau libre.
    - Consultation de l'historique personnel ("Mes réservations").
    - Annulation et suppression effective de la réservation.

- **Scénario 3** : Gestion des Conflits (Concurrence)
    - Simulation multi-utilisateurs : Deux comptes distincts ciblent le même créneau horaire.
    - Validation de la règle d'intégrité : La première réservation passe (201), la seconde est rejetée avec un code 409 Conflict.
    - Vérification que le second utilisateur peut réserver sur un créneau adjacent sans erreur.

- **Scénario 4** : Robustesse et Cas Limites
    - Sécurité : Rejet des tentatives d'accès sans token ou avec token invalide (401/403).
    - Logique Temporelle : Blocage des incohérences (ex: heure de fin antérieure à l'heure de début).
    - Intégrité : Gestion des erreurs lors de demandes sur des salles inexistantes (404) ou doublons d'emails à l'inscription.

  Conclusion : L'application gère correctement les flux nominaux ainsi que les erreurs utilisateurs classiques, garantissant qu'aucune réservation invalide ou chevauchante ne peut être créée.

2.4. Tests de Performance - (Jack YE, Alexy-Alan HEMERY)

Focus : Latence et stabilité sous charge.
Métriques Clés (SLA) :

Les tests ont simulé des charges séquentielles et parallèles pour vérifier la réactivité :

- **Health Check :** Moyenne < 100ms (Excellent).
- **Listing des Salles :** Moyenne < 500ms (Conforme).
- **Login :** Moyenne < 1000ms (Acceptable pour une fonction de hachage sécurisée).
- **Concurrence :** Le serveur reste stable lors de 10 requêtes parallèles simultanées.
- **Stabilité :** Aucune dégradation de performance observée après des séries de requêtes répétées (variance faible).

3. Bilan Quantitatif et Couverture

Type de Test	Quantité	Couverture estimée	Résultat Global
Unitaire	15	100% (Auth Module)	PASS
Intégration	22	~90% (Routes API)	PASS
Fonctionnel	4 Scénarios	Principaux flux métier	PASS
Performance	11 Métriques	Latence & Charge	PASS

4. Conclusion Générale

La campagne de tests démontre que l'application de réservation de salles atteint un **niveau de maturité élevé** pour une mise en production.

Points forts identifiés :

1. **Sécurité :** La gestion des tokens JWT et la protection des données sensibles (mots de passe) sont rigoureusement testées.
2. **Robustesse API :** L'API gère proprement les erreurs clients (4xx), évitant les crashs serveur.
3. **Intégrité des Données :** Le système de réservation empêche efficacement les doubles réservations (gestion des conflits).

Capture d’éxécution

TESTS DE PERFORMANCE
  Temps de Réponse des Endpoints
    ✓ PERF-1: Health check répond en moins de 100ms (5 ms)
    ✓ PERF-2: Liste des salles répond en moins de 500ms (81 ms)
    ✓ PERF-3: Connexion utilisateur répond en moins de 1000ms (146 ms)
    ✓ PERF-4: Détails d'une salle répondent en moins de 500ms (84 ms)
  Tests de Charge Séquentielle
    ✓ PERF-5: 10 requêtes consécutives au health check (12 ms)
    ✓ PERF-6: 5 requêtes consécutives à la liste des salles (429 ms)
  Tests de Charge Parallèle
    ✓ PERF-7: 10 requêtes parallèles au health check (14 ms)
    ✓ PERF-8: 5 requêtes parallèles à la liste des salles (331 ms)
  Tests de Stabilité
    ✓ PERF-9: Le serveur reste stable après plusieurs requêtes (935 ms)
    ✓ PERF-10: Temps de réponse cohérent sous charge légère (1058 ms)
  Tests de Concurrence
    ✓ PERF-11: Création de multiples réservations simultanées (1 ms)

PASS tests/integration.test.js
 TESTS D'INTEGRATION API
  Tests des Routes d'Authentification
    ✓ API-1: POST /api/auth/register - Structure de réponse correcte (169 ms)
    ✓ API-2: POST /api/auth/login - Retourne un token JWT (248 ms)
    ✓ API-3: GET /api/auth/profile - Headers d'authentification requis (1 ms)
    ✓ API-4: GET /api/auth/profile - Accepte un token valide (1 ms)
  Tests des Routes Salles
    ✓ API-5: GET /api/rooms - Retourne un tableau de salles (81 ms)
    ✓ API-6: GET /api/rooms - Chaque salle a les propriétés requises (91 ms)
    ✓ API-7: GET /api/rooms/:id - Retourne une salle spécifique (97 ms)
    ✓ API-8: GET /api/rooms/:id - 404 pour une salle inexistante (93 ms)
  Tests des Routes Réservations
    ✓ API-9: POST /api/bookings - Authentification requise (1 ms)
    ✓ API-10: POST /api/bookings - Validation des champs requis
    ✓ API-11: GET /api/bookings/my-bookings - Retourne les réservations de l'utilisateur
    ✓ API-12: DELETE /api/bookings/:id - Authentification requise
  Tests des Headers HTTP
    ✓ API-13: Les réponses incluent Content-Type JSON (1 ms)
    ✓ API-14: Les routes protégées vérifient le header Authorization (72 ms)
  Tests des Codes de Statut HTTP
    ✓ API-15: GET /api/health retourne 200
    ✓ API-16: POST avec données invalides retourne 400 (74 ms)
  Tests de Sécurité
    ✓ API-17: Les mots de passe ne sont pas retournés dans les réponses (200 ms)
    ✓ API-18: Les tokens invalides sont rejetés (93 ms)
    ✓ API-19: SQL injection basique est bloquée (77 ms)
  Tests de Validation des Données
    ✓ API-20: Email invalide est rejeté (75 ms)
    ✓ API-21: Mot de passe trop court est rejeté (80 ms)
    ✓ API-22: Nom complet vide est géré (169 ms)

PASS tests/fonctionnel.test.js
 TESTS FONCTIONNELS - Scenarios Utilisateur
  SCENARIO 1: Inscription et connexion d'un nouvel utilisateur
    ✓ 1.1 - L'utilisateur s'inscrit avec succès (173 ms)
    ✓ 1.2 - L'utilisateur se connecte avec ses credentials (80 ms)
    ✓ 1.3 - L'utilisateur accède à son profil (1 ms)

```
  SCENARIO 2: Consultation et réservation de salle
    ✓ 2.1 - L'utilisateur consulte la liste des salles (86 ms)
    ✓ 2.2 - L'utilisateur consulte les détails d'une salle (82 ms)
    ✓ 2.3 - L'utilisateur vérifie la disponibilité de la salle (90 ms)
    ✓ 2.4 - L'utilisateur crée une réservation (1 ms)
    ✓ 2.5 - L'utilisateur consulte ses réservations
    ✓ 2.6 - L'utilisateur annule sa réservation (1 ms)
  SCENARIO 3: Tentatives de réservation en conflit
    ✓ 3.1 - Premier utilisateur réserve un créneau (1 ms)
    ✓ 3.2 - Deuxième utilisateur tente de réserver le même créneau (1 ms)
    ✓ 3.3 - Deuxième utilisateur réserve un créneau différent
  SCENARIO 4: Gestion des erreurs et validations
    ✓ 4.1 - Tentative de réservation sans authentification (3 ms)
    ✓ 4.2 - Tentative de réservation avec token invalide (2 ms)
    ✓ 4.3 - Tentative de réservation avec heure de fin avant heure de début (1 ms)
    ✓ 4.4 - Tentative de réservation avec salle inexistante
    ✓ 4.5 - Tentative de connexion avec mauvais mot de passe (83 ms)
    ✓ 4.6 - Tentative d'inscription avec email existant (166 ms)

 console.log
    at _log (server/node_modules/dotenv/lib/main.js:142:11)

PASS  tests/unit.test.js
 TESTS UNITAIRES
   Module Auth - generateToken()
     ✓ UNIT-1: generateToken() retourne une chaîne non vide (1 ms)
     ✓ UNIT-2: generateToken() crée un JWT valide (3 parties) (1 ms)
     ✓ UNIT-3: Token contient l'ID utilisateur
     ✓ UNIT-4: Token contient l'email utilisateur (1 ms)
     ✓ UNIT-5: Token a une date d'expiration
     ✓ UNIT-6: Token expire après 7 jours
     ✓ UNIT-7: Token invalide est rejeté par jwt.verify() (4 ms)
     ✓ UNIT-8: Token avec mauvaise signature est rejeté (1 ms)
     ✓ UNIT-9: Token expiré est rejeté
   Module Auth - authenticateToken() middleware
     ✓ UNIT-10: authenticateToken() rejette requête sans header Authorization (1 ms)
     ✓ UNIT-11: authenticateToken() rejette token invalide (1 ms)
     ✓ UNIT-12: authenticateToken() accepte token valide et appelle next()
     ✓ UNIT-13: authenticateToken() attache les données utilisateur à req.user (1 ms)
     ✓ UNIT-14: authenticateToken() rejette token expiré
     ✓ UNIT-15: authenticateToken() gère Authorization header sans Bearer

Test Suites: 4 passed, 4 total
Tests:       66 passed, 66 total
Snapshots:   0 total
Time:        8.381 s, estimated 10 s
```

Code des tests automatisés présent dans le dossier **.github/workflows**