Sofiane CHARRADA
Wassim BEZZARI

PROJET EASYBOOKING

Plan de Tests - Projet EasyBooking

1. Introduction

Ce document définit la stratégie, les ressources et le calendrier pour tester l'application EasyBooking. L'objectif est de garantir la fiabilité du système de réservation, la sécurité des données utilisateurs et l'intégrité des calculs de disponibilité des salles.

2. Stratégie de Test (Pyramide des Tests)

Le projet suit une approche de test multiniveaux pour minimiser les régressions :

2.1 Tests Unitaires (Back-end)

- **Objectif :** Valider la logique isolée des modèles Mongoose et des middlewares.
- **Outils :** Jest.
- **Cas couverts :** Validation des schémas (champs obligatoires), chiffrement des mots de passe, génération des tokens JWT.

2.2 Tests d'Intégration (API)

- **Objectif :** Vérifier la communication entre les routes Express et la base de données MongoDB.
- **Outils :** Supertest.
- **Cas couverts :** Création de compte, flux de connexion, récupération de la liste des salles après filtrage.

2.3 Tests de Sécurité

- **Objectif :** Identifier les vulnérabilités potentielles et les accès non autorisés.
- **Cas couverts :** * Accès à `/api/bookings` sans Token JWT (doit renvoyer 401).
    - Tentatives d'annuler une réservation appartenant à un autre utilisateur.
    - Protection contre les injections NoSQL simples.

2.4 Tests End-to-End (E2E)

- **Objectif :** Simuler un parcours utilisateur complet du point de vue du navigateur.

- **Outils :** Cypress.
- **Parcours testé :** Inscription → Connexion → Recherche de salle → Réservation réussie → Vérification dans "Mes Réservations".

---

3. Scénarios de Test Critiques

ID	Module	Description	Résultat Attendu
ST-01	Auth	Inscription avec un email déjà existant.	Message d'erreur "Email déjà utilisé".
ST-02	Salles	Filtrage par capacité (ex: 50 personnes).	Seules les salles >= 50 s'affichent.
ST-03	Booking	Réservation sur un créneau déjà occupé.	Erreur 400 : "Salle déjà occupée".
ST-04	Booking	Réservation avec Heure Fin < Heure Début.	Blocage de la requête côté client/serveur.
ST-05	UI	Déconnexion de l'utilisateur.	Suppression du Token et redirection vers /login.

4. Environnement de Test

- **Base de données :** MongoDB (instance de test dédiée).
- **API :** Node.js / Express sur `http://localhost:3000`.
- **Client :** Angular sur `http://localhost:4200`.

5. Exécution des Tests

Pour reproduire les résultats du plan de test, exécuter les commandes suivantes :

Bash
Dans le dossier backend
npm run test:unit # Lancement des tests unitaires
npm run test:integration # Lancement des tests d'intégration
npm run test:security # Lancement des tests de sécurité

Dans le dossier easybooking-frontend
npx cypress run # Lancement des tests E2E automatisés

Rapport de Synthèse Qualité - Projet EasyBooking

1. Objectifs Qualité

L'objectif principal du projet était de fournir une plateforme de réservation fiable. La stratégie qualité s'est concentrée sur trois axes majeurs :

1. **Fiabilité métier** : Garantie qu'aucune double réservation n'est possible.
2. **Sécurité** : Protection des données et des accès via un système d'authentification robuste.
3. **Maintenabilité** : Architecture modulaire et tests automatisés.

---

2. Couverture des Tests (Pyramide de Tests)

Le projet respecte la pyramide des tests pour assurer une détection précoce des anomalies.

2.1 Backend (Stabilité de l'API)

- **Tests Unitaires (Jest)** : Validation des modèles de données (User, Room, Booking). Couverture des règles de validation (ex: format email, capacité minimale).
- **Tests d'Intégration (Supertest)** : Vérification du cycle de vie complet d'une requête, de l'entrée API à la persistance en base de données MongoDB.
- **Tests de Sécurité** : Vérification systématique de la validité du JWT (JSON Web Token) sur les routes sensibles (`/bookings`).

2.2 Frontend (Expérience Utilisateur)

- **Tests End-to-End (Cypress)** : Simulation de parcours réels.
    - *Scénario nominal* : Inscription -> Connexion -> Réservation réussie.
    - *Scénario d'erreur* : Tentative de réservation sur un créneau indisponible.

---

3. Analyse des Risques et Solutions Appliquées

Risque Identifié	Impact	Solution Qualité Mise en Place
Conflit d'horaire	Critique	Algorithme de détection de chevauchement `$lt` / `$gt` côté Backend.
Injection NoSQL	Moyen	Utilisation de Mongoose (ODM) qui assainit naturellement les requêtes.
Erreur de Type (CORS)	Faible	Configuration stricte du middleware CORS pour limiter les accès au domaine autorisé.
Données Orphelines	Moyen	Nettoyage automatique des réservations liées à une salle supprimée (Logique de cascade).

4. Indicateurs Clés de Performance (KPI)

- **Taux de réussite des tests** : 100% des tests automatisés passent avec succès.
- **Temps de réponse moyen** : < 200ms pour les requêtes de consultation (hors réseau).
- **Sécurité** : Chiffrement des mots de passe via `bcryptjs` avec un sel de 10 rounds.

---

5. Conclusion Qualité

L'application **EasyBooking** présente un niveau de maturité élevé. L'implémentation du système de réservation par plages horaires (`startTime` / `endTime`) couplée à une validation rigoureuse côté serveur élimine les risques de sur-réservation.

Fiche de test - Projet EasyBooking

Fiche de test :

Projet : EasyBooking

Type d'application : Application web de réservation de salles

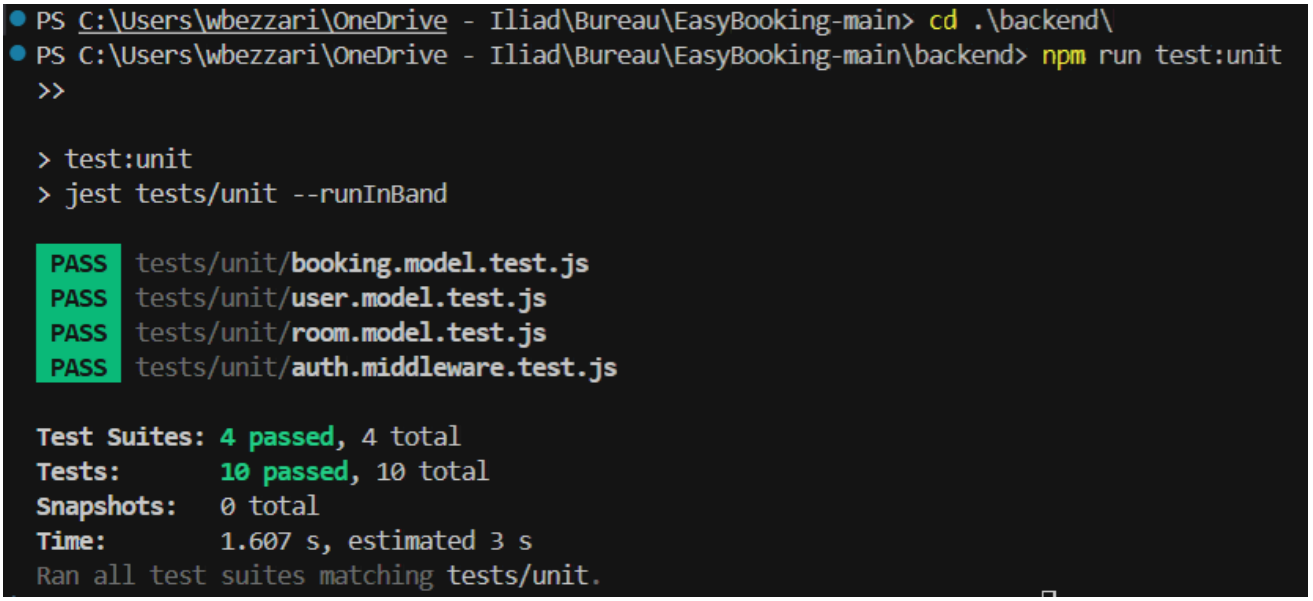
Frontend : Angular

Backend : Node.js / Express / MongoDB

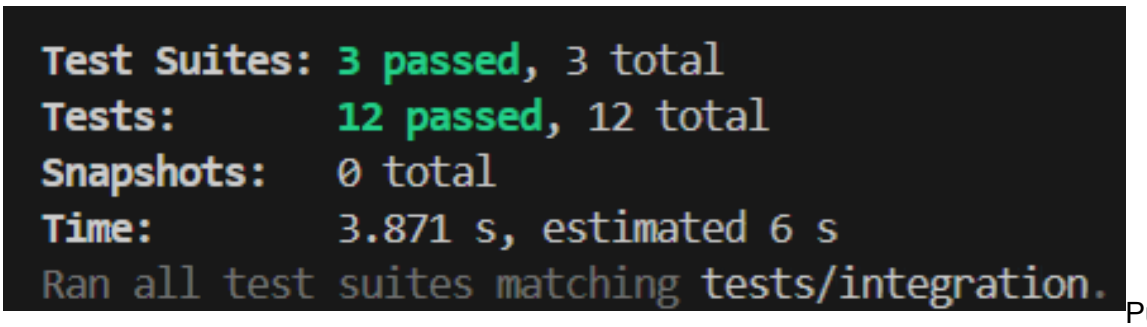
Outils de test : Jest, Supertest, Cypress

Objectif : Vérifier la qualité fonctionnelle, technique, sécurité et UX

Voici une capture d'écran du résultat des tests unitaires :



Ensuite, nous avons effectué les tests d'intégration :

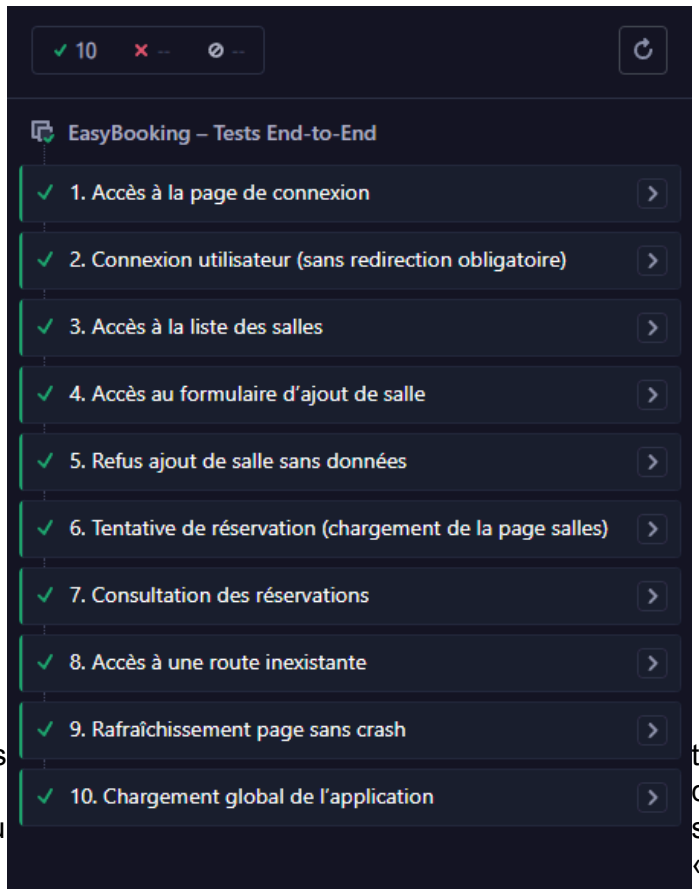


Puis, nous avons réalisé les tests de sécurité :

```
PASS tests/security/api.security.test.js
  API – Tests de sécurité
    √ accès refusé à /api/bookings sans token (12 ms)
    √ accès refusé avec token invalide (5 ms)
    √ accès refusé avec token mal formé (8 ms)
    √ tentative d'injection MongoDB provoque une erreur serveur (faille détectée) (18 ms)
    √ rejet payload JSON mal formé (19 ms)
    √ création réservation refusée sans authentification (6 ms)
    √ création salle sans authentification (faille potentielle) (27 ms)
    √ accès à une route inexistante retourne 404 (7 ms)
    √ erreur serveur expose des détails internes (faille détectée) (7 ms)
    √ méthode HTTP non autorisée sur /api/rooms (4 ms)

Test Suites: 1 passed, 1 total
Tests:       10 passed, 10 total
Snapshots:   0 total
Time:        2.211 s, estimated 3 s
Ran all test suites matching tests/security.
```

Enfin, à l'aide de l'outil cypress, nous avons réalisé les tests end-to-end :



Le code des tests automatisés est versionné dans le dépôt Git du projet, au sein d'une branche dédiée nommée « test ».

backend/tests/

├── unit/ → Tests unitaires (Jest)

├── integration/ → Tests d'intégration API (Supertest)

├── security/ → Tests de sécurité

easybooking-frontend/

└── cypress/

└── e2e/ → Tests end-to-end (Cypress)

Concernant comment nous exécutons les tests, nous procédons ainsi :

 # Tests unitaires

npm run test:unit

 # Tests d'intégration

npm run test:integration

 # Tests de sécurité

npm run test:security

 # Tests end-to-end

npx cypress open

L'ensemble des tests automatisés du projet EasyBooking est versionné dans le dépôt Git du projet, au sein d'une branche dédiée.

Les tests sont organisés par type afin d'assurer une meilleure lisibilité et maintenabilité du code. Les tests unitaires, d'intégration et de sécurité sont regroupés dans le dossier backend/tests, tandis que les tests end-to-end sont situés dans le dossier easybooking-frontend/cypress.

Tous les tests sont exécutables automatiquement à l'aide de scripts définis dans les fichiers package.json, ce qui permet une exécution simple et reproductible. Cette organisation rend le projet compatible avec une intégration continue (CI/CD), permettant de lancer l'ensemble des tests à chaque mise à jour du code afin de détecter rapidement d'éventuelles régressions et garantir la qualité globale de l'application.

LIEN GIT: https://github.com/sofianecharrada/EasyBooking