1. Introduction et Objectifs

L'objectif de ce plan de test est de valider la conformité, la fiabilité, la performance et la sécurité de l'application web "EasyBooking". Ce document servira de référence pour l'exécution des tests et la rédaction des rapports de qualité.

Portée du test (Scope) :

- Fonctionnalités : Inscription, Connexion, Affichage des salles, Réservation des salles, Affichage les réservations d'utilisateurs.
- Non-fonctionnel : Performance (charge), Sécurité (OWASP basique), Robustesse de l'API.

2. Stratégie de Test & Niveaux

Conformément aux exigences, la stratégie repose sur quatre piliers :

A. Tests Unitaires (Code Level)

- **Objectif :** Vérifier les plus petites unités de code (fonctions, méthodes) de manière isolée.
- **Cibles :** Validateurs (email, mot de passe), calculs de dates, formatage des données, composants UI isolés.

B. Tests d'Intégration (API & Database)

- **Objectif :** Vérifier que les modules fonctionnent ensemble (ex: API ↔ Base de données).
- **Cibles :** Endpoints REST (GET /rooms, POST /login), gestion des sessions, flux de réservation complet.

C. Tests de E2E (End-To-End)

- **Objectif :** Vérifier que les parcours utilisateurs complets fonctionnent correctement de bout en bout, depuis l'interface utilisateur jusqu'aux services backend, en conditions proches de la production.
- **Cibles :** Consultation de la liste des chambres, création et validation d'une réservation, gestion des erreurs et confirmation de réservation pour un utilisateur final.

D. Tests de Sécurité

- **Objectif :** Identifier les vulnérabilités critiques.
- **Cibles :** Force brute sur le login, protection des routes privées.

3. Matrice des Scénarios de Test (Cas de Test)

Voici le détail des cas à exécuter. Chaque ID servira de référence dans votre **Fiche de tests**.

Module 1 : Gestion de Compte (Inscription & Connexion)

ID	Type	Fonction / Route	Scénario	Données de Test	Résultat Attendu (Code & Body)
AUTH-01	Unitaire	`isMailValid` (Helper)	Validation format email	`test@domain` (invalide), `test@domain.com` (valide)	Retourne Faux/Vrai selon le format.
AUTH-02	Unitaire	`passwordIsValid` (Helper)	Complexité mot de passe	`password123` (Manque Majuscule/Spécial)	Rejette le faible, accepte le fort.

AUTH-03	Intégration/Unitaire	`POST /auth`
		
		`loginController(Auth)`
			Utilisateur inexistant	Email non enregistré	**404 Not Found** + "user not found".

AUTH-04	Intégration/Unitaire	`POST /auth`
		
		`loginController(Auth)`
			Mot de passe incorrect	Bon email, mauvais MDP	**400 Bad Request** + "invalid password".

AUTH-05	Intégration/Unitaire	`POST /auth`	Connexion succès	Bons identifiants	**200 OK** + JSON `{`

		loginController(Auth)			token: "..." }.
AUTH-06	E2E	front+back	Connexion avec succès	Identifiants valides	Redirection vers Accueil, Token/Session créé.
AUTH-07	E2E	front+back	Connexion échoué	Identifiants invalides	Message d'erreur visible, pas de redirection.
USER-01	Intégration/Unitaire	POST /users createUser(User)	Création avec MDP faible	Body avec MDP: weak	**400 Bad Request** + Message: "Mot de passe invalide...".
USER-02	Intégration/Unitaire	POST /users createUser(User)	Création succès	Body valide (Maje + Min + Chiffre + Spécial)	**201 Created** + User sans le champ password.
USER-03	Intégration/Unitaire	GET /users/me getUser(User)	Récupération profil	Header Authorization: Bearer valid_token	**200 OK** + Données de l'utilisateur.

USER -04	Intégration/Unitaire	GET/users/me getUser(User)	Récupération profil sans auth	Sans token	**401** (Géré par middleware userIsAuth).
USER -05	Intégration/Unitaire	GET/users/me getUser(User)	Récupération profil sans le même id	ID inexistant en BDD	**404 Not found + Message d'erreur**
USER -06	Intégration/Unitaire	GET /users getUsers(User)	Récupération des users	Body valide (email, name, password)	**200 OK** + Données des utilisateurs.
USER -07	Intégration/Unitaire	GET /users getUsers(User)	Récupération des users sans auth admin	Token User standard ou sans token	**401/403** (Géré par middleware isAdmin).
USER -08	Intégration/Unitaire	PUT /users/me updateUser(User)	Modification l'utilisateur sans Auth	Sans token	**401** (Géré par middleware userIsAuth).
USER -09	Intégration/Unitaire	PUT /users/me	Modification l'utilisateur	Identifiant valide, Body valide	**200 OK +** Données de

		updateUser(User)	avec succès		l'utilisateur.
USER-10	Intégration/Unitaire	PUT /users/me updateUser(User)	Modification l'utilisateur avec body invalide	Body invalide (Manque email, name, password)	**400 Bad request +** Message d'erreur de debug
USER-12	Intégration/Unitaire	DELETE /users/:id deleteUser(User)	Suppression l'utilisateur sans auth admin	Token User standard ou sans token	**401/403** (Géré par middleware isAdmin).
USER-13	Intégration/Unitaire	DELETE /users/:id deleteUser(User)	Suppression l'utilisateur avec succès	identifiant valide	**200 OK +** Données de l'utilisateur supprimé
USER-14	Intégration/Unitaire	DELETE /users/:id deleteUser(User)	Suppression l'utilisateur avec identifiant invalide	identifiant invalide	**500 Erreur**

Module 2 : Consultation des salles

ID	Type	Fonction / Route	Scénario	Données de Test	Résultat Attendu (Code & Body)
ROOM-01	Intégration /Unitaire	`GET /rooms`	Récupérer la liste des salles		**200 OK** + Données de la liste des chambres
ROOM-02	Intégration /Unitaire	`GET /rooms/:id`	Récupérer une salle par id avec succès	identifiant valide de chambre	**200 OK** + Données de la chambre
ROOM-03	Intégration /Unitaire	`GET /rooms/:id`	Récupérer une salle par id avec un identifiant invalide	ID inexistant en BDD	**404 Not found** + “Salle non trouvée”
ROOM-04	Intégration /Unitaire	`POST /rooms`	Création sans Auth Admin	Token User standard ou sans token	**401/403** (Géré par middleware `isAdmin`).
ROOM-05	Intégration /Unitaire	`POST /rooms`	Validation Équipement (Type)	`equipment`: "Wifi" (String au lieu d'Array)	**400 Bad Request** + "L'équipement doit être un tableau".

ROOM-06	Intégration /Unitaire	POST /rooms	Validation Équipement (Contenu)	equipment: ["Wifi", ""] (String vide)	**400 Bad Request** + "chaînes de caractères non vides".
ROOM-07	Intégration /Unitaire	POST /rooms	Validation name (Contenu)	name: null ou “”	**400 Bad Request** + "Nom de la salle invalide ".
ROOM-08	Intégration /Unitaire	POST /rooms	Validation capacity (Contenu)	capacity:null ou <= 0	**400 Bad Request** + "Capacité de salle invalide".
ROOM-09	Intégration /Unitaire	POST /rooms	Création Succès	Nom, Capacité > 0, Équipement valide	**201 Created** + Objet Room créé.
ROOM-10	Intégration /Unitaire	PUT/rooms/:id	Modification sans auth Admin	Token User standard ou sans token	**401/403** (Géré par middleware isAdmin).
ROOM-11	Intégration /Unitaire	PUT/rooms/:id	Modification avec identifiant invalide	ID inexistant en BDD	**404 Not found** + “Salle non trouvée”
ROOM-12	Intégration /Unitaire	PUT/rooms/:id	Modification sans succès avec	capacity est nulle ou <= 0	**400 Bad Request** +

			capacity invalide		"Capacité de salle invalide".
ROOM-13	Intégration /Unitaire	PUT/rooms/:id	Modification sans succès avec name invalide	name est null ou “”	**400 Bad Request** + "Nom de la salle invalide ".
ROOM-14	Intégration /Unitaire	PUT /rooms/:id	Mise à jour	Body valide (name, capacity, equipement)	**200 OK +** Données mises à jour
ROOM-15	Intégration /Unitaire	DELETE /rooms/:id	Suppression sans auth admin	Token User standard ou sans token	**401/403** (Géré par middleware isAdmin).
ROOM-15	Intégration /Unitaire	DELETE /rooms/:id	Suppression ID inconnu	ID inexistant en BDD	**404 Not Found** + "Salle non trouvée".
ROOM-16	Intégration /Unitaire	DELETE /rooms/:id	Suppression avec succès	ID valide	**200 OK** + “Salle supprimée avec succès”
ROOM-17	E2E		Affichage UI des chambres		Les cartes des chambres s'affichent (Nom, Capacité).

ROOM-18	E2E		Filtrage par capacité (Optionnel)	Filtre "4 personnes"	Seules les chambres de capacité >= 4 s'affichent.
ROOM-19	Performance		Temps de réponse liste chambres	50 utilisateurs simultanés	Temps de réponse < 200ms.

Module 3 : Réservation

ID	Type	Fonction / Route	Scénario	Données de Test	Résultat Attendu (Code & Body)
BOOK-01	Unitaire	`reservationDateIsValid`	Date fin avant date début	Start: 14h, End: 13h	Retourne `false` (via Mock ou appel direct).
BOOK-02	Intégration/Unitaire	`POST /reservations`	Chevauchement exact	Créneau identique à une résa existante	**400 Bad Request** + "La salle est déjà réservée...".
BOOK-03	Intégration/Unitaire	`POST /reservations`	Chevauchement partiel	Début pendant une autre résa	**400 Bad Request** + "La salle est

					déjà réservée. ..".
BOOK -04	Intégration/Unitaire	POST /reservations	Identifiant utilisateur invalide	userId est null	**400 Bad Request** + "ID utilisateur invalide".
BOOK -05	Intégration/Unitaire	POST /reservations	Identifiant de la salle invalide	roomId est null	**400 Bad Request** + "ID de salle invalide".
BOOK -06	Intégration/Unitaire	POST /reservations	Validation Date réservation	Si startDate > endDate ou startDate < now ou endDate < now	**400 Bad Request** + "Plage horaire invalide".
BOOK -07	Intégration/Unitaire	POST /reservations	Création Succès	Créneau libre, IDs valides	**201 Created** + Objet Reservation.
BOOK -08	Intégration/Unitaire	GET/reservations	Récupération des réservations par User avec userId inconnu	ID inexistant en BDD	**400 Bad request +** “ID utilisateur invalide”

BOOK -09	Intégration/Unitaire	GET/reservations	Récupération des réservations par User avec succès		**200 OK+** Données des réservations
BOOK -10	Intégration/Unitaire	GET/reservations/:date	Récupération des réservations après la date avec l'id inconnu	ID inexistant en BDD	**400 Bad request +** "ID utilisateur invalide"
BOOK -11	Intégration/Unitaire	GET/reservations/:date	Récupération des réservations après la date invalide	Date est sous format invalide (pattern invalide)	**400 Bad request +** "Date invalide"
BOOK -12	Intégration/Unitaire	GET/reservations/:date	Récupération des réservations après la date avec succès	Format Date invalide (URL: /reservations/bonjour)	**200 OK +** Données des réservations
BOOK -13	Sécurité	DELETE /reservations/:id	**Protection IDOR** (Suppression résa autrui)	User A tente de supprimer réservation de User B	**403 Forbidden** + "Vous n'êtes pas autorisé...".

BOOK-14	Intégration/Unitaire	DELETE /reservations/:id	Suppression légitime	User A supprime sa propre réservation	**200 OK** + "Réservation supprimée avec succès".
BOOK-15	Intégration/Unitaire	DELETE /reservations/:id	Aucune réservation	ID inexistant en BDD	**404 Not found** + “Réservation non trouvée avec cet ID”
BOOK-16	Intégration/Unitaire	GET /reservations/room/:id	Accès non Admin	User standard tente de voir toutes les résas d'une salle	**401/403** (Bloqué par isAdmin).
BOOK-17	Intégration/Unitaire	GET /reservations/room/:id	Identifiant de salle invalide	roomId invalide	**400 Bad request** + “ID de salle invalide”
BOOK-18	Intégration/Unitaire	GET /reservations/room/:id	Récupération toutes les réservations d’une salle		**200 OK** + Données des réservation
BOOK-19	E2E		Flux complet de réservation	UI: Choix salle -> Date -> Confirmer	Message de confirmation, mise

					à jour visuelle.

4. Environnement et Outils (Stack Technique Suggérée)

Pour réaliser les livrables demandés (Code Git, Rapports), voici la configuration recommandée :

- **Langage Application :** React, TypeScript/Node.js.
- **Outils de Tests Unitaires :** Vitest(JS).
- **Outils de Tests d'Intégration:** Vitest (API).
- **Outils de E2E :** Playwright
- **Outils de Sécurité** : Vitest (API)
- **Gestion de version :** Git & GitHub.

5. Critères d'Acceptation (Definition of Done)

Le projet sera considéré comme validé si :

1. **Tests Unitaires :** Couverture de **code > 70% sur le backend.**
2. **Tests d'Intégration :** Flux critiques (Login, Réservation) **validés en données réelles** (pas d'erreur).
3. **Tests E2E :** Parcours complet **("Inscription à Historique")** fonctionnel sur l'interface.
4. **Tests de sécurité :** Routes Admin protégées et impossibilité de modifier les données d'autrui (anti-IDOR).
5. **Livrables :** Rapports de preuves (Excel) générés et stockés sur Docs.