Tests Logiciels

Plan de Test Global

Objectif Général du Plan

L'objectif de ce plan de test est de garantir la qualité, la fonctionnalité, la robustesse et la sécurité de l'application de réservation de salles en exécutant une suite complète de tests unitaires, d'intégration, de sécurité et de bout en bout (E2E).

Stratégie de Test

Les tests sont structurés selon la pyramide de test, avec une concentration sur les tests unitaires et d'intégration, complétés par des tests de sécurité et des tests E2E.

Niveau de Test	Outil / Framework	Objectif Principal
Unitaire	Jest	Vérifier la logique interne des fonctions.
Intégration	Supertest / Jest	Valider l'interaction entre les composants et la base de données.
Sécurité (Intégration)	Supertest / Jest	Identifier les vulnérabilités de l'API (Injection, Contrôle d'Accès, Fuite de données).
E2E (Bout en Bout)	Cypress	Simuler le parcours utilisateur complet contre l'API mockée.

Environnement de Test

Catégorie	Détail
Backend	Node.js / Express.js
Base de Données	MongoDB (MongoMemoryServer en environnement de test)
Outil de Test	Jest, Supertest, Cypress
Isolation	Les tests d'intégration utilisent une base de données en mémoire (MongoMemoryServer) et sont réinitialisés après chaque test.

Détail des Catégories de Tests

Tests Unitaires (Total : 54 cas)

Ces tests vérifient le comportement des composants en les isolant.

Composant Testé	IDs de Test (Plage)	Nombre de Cas	Objectifs Clés Vérifiés
Auth Controller	TC-01 à TC-06	6	Gestion des statuts HTTP (201, 200, 4xx, 500) en fonction du service.

Auth Middlewares	TC-07 à TC-13	7	Validation du token JWT (401), Validation des données d'entrée (412).
Auth Service	TC-14 à TC-18	5	Logique métier de l'authentification (unicité, hachage, comparaison de mot de passe).
Booking Controller	TC-19 à TC-27	9	Délégation des requêtes, gestion des *userId*, mapping des erreurs du service.
Booking Service	TC-28 à TC-38	11	Validation des contraintes horaires (passé, ordre, 08:00-18:00, conflits 409).
Room Controller	TC-39 à TC-45	7	Gestion des filtres (capacity), retour des créneaux, gestion des erreurs.
Room Service	TC-46 à TC-54	9	Logique de filtrage, Calcul complexe de disponibilité de salle (créneaux libres).

Tests d'Intégration (Total : 10 cas)

Ces tests vérifient la communication entre l'API et la base de données.

Domaine Testé	IDs de Test (Plage)	Nombre de Cas	Objectifs Clés Vérifiés
Authentification	TC-55 à TC-58	4	Flux Register/Login, Échecs (409 Email existant, 412 Validation).

Gestion des Salles	TC-59 à TC-60	2	Initialisation des données de salle, Récupération des salles par l'utilisateur connecté.
Réservation	TC-61 à TC-64	4	Création, Rejet de conflit (409), Consultation par l'utilisateur, Suppression (annulation).

Tests de Sécurité & Pénétration (Total : 10 cas)

Ces tests vérifient la résilience de l'API face aux attaques courantes.

Domaine Testé	IDs de Test (Plage)	Nombre de Cas	Objectifs Clés Vérifiés
Autorisation	SEC-01	1	Contrôle d'accès : Un utilisateur B ne peut pas manipuler la ressource d'un utilisateur A (*404/403*).
Injection	SEC-02, SEC-04	2	Prévention de l'injection NoSQL et du XSS.
Gestion des Sessions	SEC-03, SEC-06	2	Rejet des tokens JWT forgés ou expirés (*401*).
Résilience / Fuites	SEC-09, SEC-10	2	Absence de traces de pile dans les messages d'erreur (*500*), Rejet des charges utiles trop grandes (*413*).
Validation	SEC-05, SEC-07, SEC-08	3	Non-fuite de données sensibles, Application d'une politique de

			mot de passe fort, Gestion des types de données inattendus.

Tests E2E (Cypress)

Ces tests simulent le comportement de l'utilisateur final à travers l'interface client (front-end) en mockant les réponses de l'API.

Domaine Testé	Objectifs Clés Vérifiés
Flux Complet	Enregistrement, Connexion, Affichage du Dashboard, Navigation vers les réservations, Annulation.
Fonctionnalité UI	Filtrage des salles sur l'UI, Ouverture du modal de réservation, Création de la réservation via l'UI.
Gestion des Erreurs	Affichage du message d'erreur (*401 Invalid Credentials*) et non-redirection en cas d'échec de connexion.

Résumé de la Couverture de Test

Catégorie	Nombre Total de Cas	Description de la Couverture
Unitaires	54	Logique métier et interfaces de bas niveau entièrement testées.
Intégration	10	Validation de la persistance des données et de l'intégration du flux API principal.
Sécurité	10	Couverture des vulnérabilités critiques de l'API pour les routes implémentées.
E2E (Cypress)		Validation du parcours utilisateur complet et de la cohérence de l'interface.

Total Global	**74 cas**	

Fiches de Tests

Contrôleur d'Authentification

Ce document présente les fiches de tests unitaires pour le contrôleur d'authentification, couvrant les fonctions d'enregistrement (`register`) et de connexion (`login`).

I. Objectifs

L'objectif de ces tests est de vérifier que le contrôleur d'authentification (`auth.controller`) interagit correctement avec la couche de service (`auth.service`) et retourne les codes de statut HTTP et les messages d'erreur appropriés dans différents scénarios (succès et échecs).

II. Périmètre des Tests

Domaine	Description du Périmètre
Contrôleur d'Authentification	Fonctions `register` et `login`.
Services Mockés	`registerUser` et `loginUser` de `auth.service`.
Scénarios	Succès (statut 200/201), Échecs (statuts 4xx), Erreurs internes (statut 500).

III. Fiches de Tests Détaillées

ID Test	Fonction Testée	Description du Test	Résultat Attendu	Statut HTTP Attendu
TC-01	`register`	Succès de l'enregistrement d'un nouvel utilisateur.	Un nouvel utilisateur est créé et retourné.	201
TC-02	`register`	Échec de l'enregistrement (ex: email déjà utilisé).	L'erreur du service avec un statusCode est gérée (409).	409

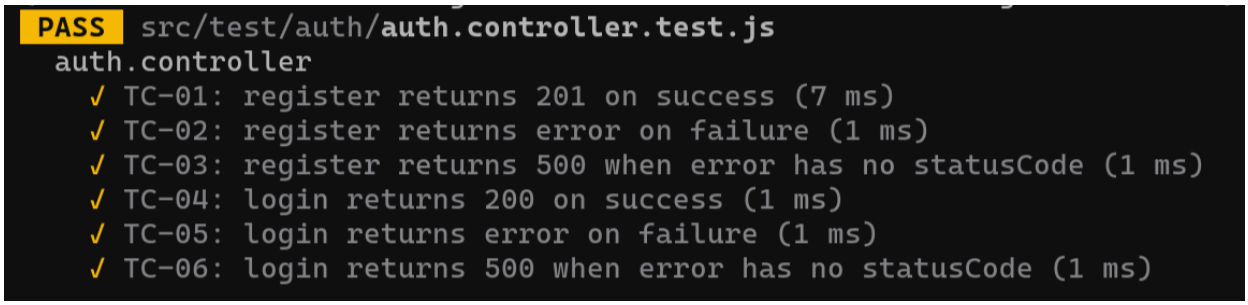
TC-03	`register`	Gestion d'une erreur interne sans statusCode (erreur inattendue).	L'erreur est traitée comme une erreur serveur.	500
TC-04	`login`	Succès de la connexion d'un utilisateur.	Un token d'accès (accessToken) est retourné.	200
TC-05	`login`	Échec de l'authentification (ex: mauvais mot de passe).	L'erreur du service avec un statusCode est gérée (401).	401
TC-06	`login`	Gestion d'une erreur interne sans statusCode (erreur inattendue).	L'erreur est traitée comme une erreur serveur.	500

IV. Exécution des Tests

Les tests unitaires du contrôleur d'authentification ont été exécutés avec **Succès**.

ID Test	Description du Test	Résultat Obtenu (Asserts Clés)	Statut
TC-01	`register` returns 201 on success	`res.status` appelé avec 201 et `res.json` avec les données de l'utilisateur.	Succès
TC-02	`register` returns error on failure	`res.status` appelé avec 409 et `res.json` avec le message d'erreur.	Succès
TC-03	`register` returns 500 when error has no `statusCode`	`res.status` appelé avec 500 et `res.json` avec le message d'erreur non spécifié.	Succès
TC-04	`login` returns 200 on success	`res.status` appelé avec 200 et `res.json` avec le `accessToken`.	Succès

TC-05	`login` returns error on failure	`res.status` appelé avec 401 et `res.json` avec le message "Authentication failed".	Succès
TC-06	`login` returns 500 when error has no `statusCode`	`res.status` appelé avec 500 et `res.json` avec le message d'erreur du service.	Succès



Middlewares d'Authentification

Ce document présente les fiches de tests unitaires pour le middleware d'authentification (`auth.middleware`) et le middleware de validation (`authvalidation.middleware`).

I. Objectifs

L'objectif de ces tests est de vérifier :

- Que le middleware d'authentification (`auth.middleware`) assure que seul un utilisateur avec un token JWT valide peut accéder aux routes protégées.
- Que le middleware de validation (`authvalidation.middleware`) vérifie la conformité des données d'entrée (e-mail, mot de passe, nom d'utilisateur) lors de l'enregistrement et de la connexion, et bloque les requêtes invalides.

II. Périmètre des Tests

Domaine	Description du Périmètre
Middleware d'Authentification	Fonctions de vérification du token JWT et de l'en-tête `Authorization`.
Middleware de Validation	Validation des données d'entrée pour *register* et *login*.

Services Mockés	*jsonwebtoken.verify* est mocké.
Scénarios	Succès (appel de *next*), Échecs d'authentification (401), Échecs de validation (412).

III. Fiches de Tests Détaillées

ID Test	Fonction Testée	Description du Test	Résultat Attendu	Statut HTTP Attendu
TC-07	auth.middleware	Token d'authentification valide.	Le middleware appelle la fonction next() et ajoute l'ID utilisateur à req.user.	Aucun
TC-08	auth.middleware	Token d'authentification invalide.	Le processus est arrêté et une erreur d'authentification est retournée.	401
TC-09	auth.middleware	En-tête d'autorisation manquant.	Le processus est arrêté et une erreur d'authentification est retournée.	401
TC-10	registerValidation	Validation de l'enregistrement avec données valides.	Le middleware appelle la fonction next().	Aucun
TC-11	registerValidation	Validation de l'enregistrement avec données invalides (ex: email/mot de passe/username invalide).	Le processus est arrêté et une erreur de précondition échouée est retournée.	412
TC-12	loginValidation	Validation de la connexion avec données valides.	Le middleware appelle la fonction next().	Aucun

TC-13	loginValidation	Validation de la connexion avec données invalides (ex: email/mot de passe manquant).	Le processus est arrêté et une erreur de précondition échouée est retournée.	412

IV. Exécution des Tests

Les tests unitaires des middlewares d'authentification et de validation ont été exécutés avec **Succès**.

ID Test	Description du Test	Résultat Obtenu (Asserts Clés)	Statut
TC-07	auth.middleware appelle next si le token est valide.	userId est correctement défini	Succès
TC-08	auth.middleware retourne 401 si le token est invalide.	status 401 avec le message "Authentification Failed".	Succès
TC-09	auth.middleware retourne 401 si l'en-tête est manquant.	status 401.	Succès
TC-10	registerValidation passe avec des données valides.	next est appelé.	Succès
TC-11	registerValidation échoue avec des données invalides.	status 412 et erreur de validation.	Succès
TC-12	loginValidation passe avec des données valides.	next est appelé.	Succès
TC-13	loginValidation échoue avec des données invalides.	status 412 et erreur de validation.	Succès

```
PASS src/test/auth/auth.middleware.test.js
  auth.middleware
    ✓ TC-07: should call next if token is valid (6 ms)
    ✓ TC-08: should return 401 if token is invalid (1 ms)
    ✓ TC-09: should return 401 if authorization header is missing (1 ms)
  authvalidation.middleware
    ✓ TC-10: registerValidation passes valid data (3 ms)
    ✓ TC-11: registerValidation fails invalid data (2 ms)
    ✓ TC-12: loginValidation passes valid data
    ✓ TC-13: loginValidation fails invalid data
```

Couche de Service d'Authentification

I. Objectifs

L'objectif de ces tests est de vérifier la logique et l'interaction du service d'authentification avec la base de données, couvrant les fonctions d'enregistrement et de connexion.

II. Périmètre des Tests

Domaine	Description du Périmètre
Service d'Authentification	Fonctions registerUser et loginUser
Services Mockés	
Scénarios	Enregistrement (succès/échec), Connexion (succès/échec), Gestion des erreurs.

III. Fiches de Tests Détaillées

ID Test	Fonction Testée	Description du Test	Résultat Attendu	Erreur/Résultat
TC-14	registerUser	Échec de l'enregistrement : l'e-mail est déjà utilisé.	Une exception est levée avec le message "Email already used".	Échec/Exception

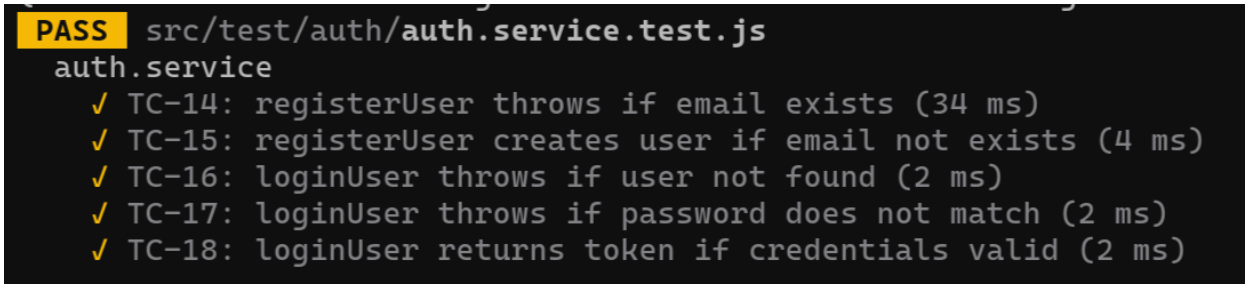
TC-15	registerUser	Succès de l'enregistrement : le nouvel utilisateur est créé.	Un nouvel utilisateur est créé.	Succès/Retour
TC-16	loginUser	Échec de la connexion : l'utilisateur n'est pas trouvé.	Une exception est levée avec le message "Authentication failed".	Échec/Exception
TC-17	loginUser	Échec de la connexion : le mot de passe ne correspond pas (après vérification par bcrypt.compare).	Une exception est levée avec le message "Authentication failed".	Échec/Exception
TC-18	loginUser	Succès de la connexion : les identifiants sont valides.	Un objet contenant un (token JWT) est retourné.	Succès/Retour

IV. Exécution des Tests

Les tests unitaires de la couche de service d'authentification ont été exécutés avec **Succès**.

ID Test	Description du Test	Résultat Obtenu (Asserts Clés)	Statut
TC-14	registerUser lève une exception si l'e-mail existe.	userModel.findOne a été appelé, et la fonction registerUser a rejeté la promesse avec le message attendu.	Succès
TC-15	registerUser crée un utilisateur si l'e-mail n'existe pas.	userModel.findOne a retourné null. bcrypt.hash a été appelé. La fonction userModel a été instanciée et save a été appelé. L'objet utilisateur retourné contient les bonnes informations.	Succès
TC-16	loginUser lève une exception si l'utilisateur n'est pas	userModel.findOne a retourné null. La fonction loginUser a rejeté la promesse avec le message	Succès

	trouvé.	"Authentication failed".	
TC-17	loginUser lève une exception si le mot de passe ne correspond pas.	userModel.findOne a retourné un utilisateur. bcrypt.compare a été appelé et a retourné false. La fonction loginUser a rejeté la promesse avec le message "Authentication failed".	Succès
TC-18	loginUser retourne un token si les identifiants sont valides.	userModel.findOne a retourné un utilisateur. bcrypt.compare a été appelé et a retourné true. jwt.sign a été appelé. L'objet retourné contient l'accessToken attendu.	Succès



Contrôleur de Réservation

I. Objectifs

L'objectif de ces tests est de vérifier que le contrôleur de réservation interagit correctement avec la couche de service transmet les données appropriées et retourne les codes de statut HTTP et les messages d'erreur pertinents pour tous les scénarios (succès, échecs clients, erreurs serveur).

II. Périmètre des Tests

Domaine	Description du Périmètre
Contrôleur de Réservation	Fonctions createBooking, deleteBooking et getMyBookings.
Services Mockés	createBooking, deleteBooking et getUserBookings de bookings.service.

Scénarios	Succès (200/201), Échecs clients (400, 404, 418), Erreurs internes (500).

III. Fiches de Tests Détaillées

ID Test	Fonction Testée	Description du Test	Résultat Attendu	Statut HTTP Attendu
TC-19	createBooking	Succès de la création d'une réservation.	La réservation est créée et retournée avec le statut 201.	201
TC-20	createBooking	Échec de création : Le service lève une erreur spécifique (ex: 400).	L'erreur du service est gérée et le statut 400 est retourné.	400
TC-21	createBooking	Gestion d'une erreur interne sans statusCode (erreur inattendue).	L'erreur est traitée par défaut avec le statut 500 et un message d'erreur.	500
TC-22	createBooking	Gestion d'une erreur avec statusCode mais sans message d'erreur.	Utilise le statusCode de l'erreur (ex: 418) avec le message par défaut "Server error".	418
TC-23	deleteBooking	Succès de la suppression d'une réservation.	Un message de confirmation de suppression est retourné.	200
TC-24	deleteBooking	Échec de suppression : Réservation non trouvée ou non autorisée (statusCode 404).	L'erreur du service est gérée avec le statut 404 et le message d'erreur.	404
TC-25	deleteBooking	Gestion d'une erreur interne sans statusCode (erreur inattendue).	L'erreur est traitée par défaut avec le statut 500 et le message d'erreur.	500

TC-26	getMyBookings	Succès de la récupération des réservations de l'utilisateur.	La liste des réservations est retournée.	200
TC-27	getMyBookings	Échec de la récupération des réservations (erreur de service).	L'erreur est traitée par défaut avec le statut 500 et le message d'erreur par défaut "Server error".	500

IV. Exécution des Tests

Les tests unitaires du contrôleur de réservation ont été exécutés avec **Succès**.

ID Test	Description du Test	Résultat Obtenu (Asserts Clés)	Statut
TC-19	createBooking retourne 201 au succès.	bookingService.createBooking appelé avec les données du corps (roomId, startTime, endTime) et le userId de req. res.status appelé avec 201.	Succès
TC-20	createBooking retourne 400 si le service échoue avec ce statut.	res.status appelé avec 400 et res.json avec le message d'erreur.	Succès
TC-21	createBooking retourne 500 sur une erreur serveur non spécifiée.	res.status appelé avec 500 et res.json avec le message de l'erreur.	Succès
TC-22	createBooking retourne le statut de l'erreur même sans message (418).	res.status appelé avec 418 et res.json avec le message "Server error".	Succès
TC-23	deleteBooking retourne 200 sur une suppression réussie.	bookingService.deleteBooking appelé avec userId et bookingId. res.status appelé avec 200.	Succès
TC-24	deleteBooking gère l'erreur 404 "Booking not	res.status appelé avec 404 et res.send avec le message	Succès

	found".	d'erreur.	
TC-25	deleteBooking retourne 500 si l'erreur n'a pas de statusCode.	res.status appelé avec 500 et res.send avec le message de l'erreur.	Succès
TC-26	getMyBookings retourne 200 et la liste des réservations.	bookingService.getUserBookings appelé avec le userId. res.json appelé avec la liste.	Succès
TC-27	getMyBookings retourne 500 si le service échoue.	res.status appelé avec 500 et res.send avec le message "Server error".	Succès

```
PASS src/test/booking/booking.controller.test.js
  Booking Controller
    createBooking
      ✓ TC-19: should return 201 and the booking if creation is successful (7 ms)
      ✓ TC-20: should return 400 if service throws a specific error (50 ms)
      ✓ TC-21: should return 500 on unexpected server error (4 ms)
      ✓ TC-22: should use default 'Server error' message if error object has no message (2 ms)
    deleteBooking
      ✓ TC-23: should return 200 on successful deletion (1 ms)
      ✓ TC-24: should handle 404 if booking not found
      ✓ TC-25: should default to 500 if error has no statusCode
    getMyBookings
      ✓ TC-26: should return 200 and list of bookings (1 ms)
      ✓ TC-27: should return 500 if service fails (2 ms)
```

Couche de Service de Réservation

I. Objectifs

L'objectif de ces tests est de vérifier la logique (contraintes horaires, vérification des conflits) et l'interaction avec la base de données (création, recherche, suppression) du service de réservation.

II. Périmètre des Tests

Domaine	Description du Périmètre
Service de Réservation	Fonctions createBooking, getUserBookings et deleteBooking.
Services Mockés	Modèle Mongoose Booking, avec mocking des fonctions findOne, find, findOneAndDelete, et de la méthode save.

Scénarios	Validation des contraintes de temps, gestion des conflits (409), consultation (tri/population), suppression (404/succès).

III. Fiches de Tests Détaillées

ID Test	Fonction Testée	Description du Test	Résultat Attendu	Erreur/Statut Attendu
TC-28	createBooking	Échec de création : L'heure de début est dans le passé.	"Cannot book a room in the past".	Échec/Exc…
TC-29	createBooking	Échec de création : L'heure de fin est avant l'heure de début.	"End time must be after start time".	Échec/Exc…
TC-30	createBooking	Échec de création : Les heures ne sont pas pleines (ex: 10:15).	Lève une exception (regex) avec le message sur les "full hours only".	Échec/Exc…
TC-31	createBooking	Échec de création : L'heure est en dehors des heures d'ouverture (avant 08:00).	"Bookings are only allowed between 08:00 and 18:00".	Échec/Exc…
TC-32	createBooking	Échec de création : L'heure est en dehors des heures d'ouverture (après 18:00).	"Bookings are only allowed between 08:00 and 18:00".	Échec/Exc…
TC-33	createBooking	Échec de création : L'heure de début est à l'heure de fermeture (18:00).	"Bookings are only allowed between 08:00 and 18:00".	Échec/Exc…

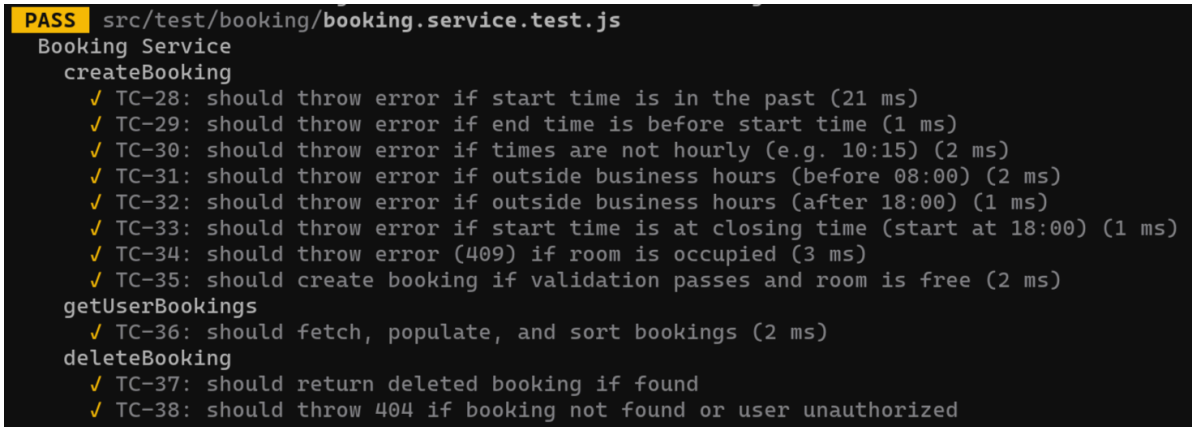
TC-34	createBooking	Échec de création (Conflit) : La salle est déjà occupée sur la plage horaire.	Lève une exception avec le message "ROOM_OCCUPIED" et un statusCode: 409.	Échec/409
TC-35	createBooking	Succès de création : Toutes les validations passent et la salle est libre.	Une nouvelle réservation est créée	Succès/R…
TC-36	getUserBooking	Succès de consultation : Récupération des réservations d'un utilisateur.	Récupère, "populate" les données de la salle, trie les résultats par `startTime` décroissant, et retourne la liste.	Succès/R…
TC-37	deleteBooking	Succès de suppression : Suppression d'une réservation existante par l'utilisateur propriétaire.	La réservation est supprimée	Succès/R…
TC-38	deleteBooking	Échec de suppression : Réservation non trouvée (ou l'utilisateur n'est pas le propriétaire).	"Booking not found" et un `statusCode: 404`.	Échec/404

IV. Exécution des Tests

Les tests unitaires de la couche de service de réservation ont été exécutés avec **Succès**.

ID Test	Description du Test	Résultat Obtenu (Asserts Clés)	Statut

TC-28	Lève une erreur si l'heure de début est dans le passé.	"Cannot book a room in the past".	Succès
TC-29	Lève une erreur si l'heure de fin est avant l'heure de début.	"End time must be after start time"	Succès
TC-30	Lève une erreur si les heures ne sont pas pleines.	L'exception avec la regex `full hours only` est levée.	Succès
TC-31	Lève une erreur si la réservation est avant 08:00.	"Bookings are only allowed between 08:00 and 18:00"	Succès
TC-32	Lève une erreur si la réservation se termine après 18:00.	"Bookings are only allowed between 08:00 and 18:00".	Succès
TC-33	Lève une erreur si l'heure de début est à 18:00.	"Bookings are only allowed between 08:00 and 18:00".	Succès
TC-34	Lève une erreur 409 (ROOM_OCCUPIED) si la salle est occupée.	L'exception avec le `statusCode: 409` et le message attendu est levée.	Succès
TC-35	Crée une réservation si la salle est libre et les données valides.	L'objet créé est retourné avec un `_id`.	Succès
TC-36	Récupère, "populate" et trie les réservations pour un utilisateur.	La liste des mocks est retournée.	Succès
TC-37	Retourne la réservation supprimée si trouvée.	La réservation est confirmée supprimée.	Succès
TC-38	Lève une erreur 404 (Booking not found) si la réservation n'existe pas ou n'appartient pas à l'utilisateur.	L'exception avec le `statusCode: 404` et le message attendu est levée.	Succès



Contrôleur de Salle

I. Objectifs

L'objectif de ces tests est de vérifier que le contrôleur de salle interagit correctement avec la couche de service, transmet les paramètres appropriés (filtre de capacité, identifiant de salle, date) et retourne les codes de statut HTTP et les messages d'erreur pertinents pour tous les scénarios (succès et échecs).

II. Périmètre des Tests

Domaine	Description du Périmètre
Contrôleur de Salle	Fonctions getRooms, getRoomAvailability et seedDatabase.
Services Mockés	getAllRooms, getRoomAvailability et seedRooms de rooms.service.
Scénarios	Consultation de la liste (sans/avec filtre), Consultation de disponibilité (succès/échec), Initialisation de la base de données (succès/échec), Gestion des erreurs internes (statut 500).

III. Fiches de Tests Détaillées

ID Test	Fonction Testée	Description du Test	Résultat Attendu	Statut HTTP Attendu

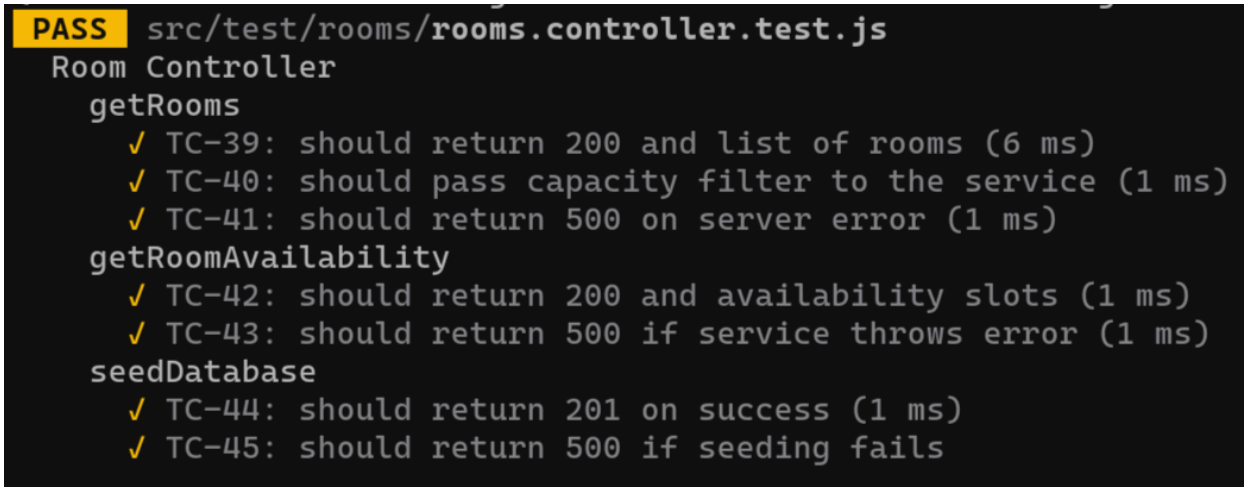
TC-39	getRooms	Succès de la récupération de la liste des salles (sans filtre).	La liste de toutes les salles est retournée.	200
TC-40	getRooms	Succès de la récupération avec un filtre de capacité (ex: capacity: "10").	Le filtre est correctement transmis au service, et la liste des salles filtrées est retournée.	200
TC-41	getRooms	Gestion d'une erreur interne (ex: échec de connexion à la DB).	L'erreur est traitée comme une erreur serveur avec un message par défaut.	500
TC-42	getRoomAvailability	Succès de la récupération des créneaux de disponibilité pour une salle et une date données.	Un objet contenant l'ID de la salle et la liste des créneaux disponibles (`availableSlots`) est retourné.	200
TC-43	getRoomAvailability	Gestion d'une erreur (ex: date invalide) lancée par le service.	L'erreur du service est retournée avec le statut 500 et le message d'erreur.	500
TC-44	getRoomAvailabilit	Succès de l'initialisation de la base de données (seeding).	Un message de succès est retourné.	201
TC-45	getRoomAvailability	Échec de l'initialisation de la base de données (ex: violation de contrainte d'unicité).	L'erreur du service est retournée avec le statut 500 et le message d'erreur.	500

IV. Exécution des Tests

Les tests unitaires du contrôleur de salle ont été exécutés avec **Succès**.

ID Test	Description du Test	Résultat Obtenu (Asserts Clés)	Statut
TC-39	getRooms retourne 200 et la liste des salles.	roomService.getAllRooms appelé avec undefined. res.status appelé avec 200 et res.json avec la liste des mocks.	Succès

TC-40	getRooms transmet le filtre de capacité au service.	roomService.getAllRooms appelé avec "10". res.status appelé avec 200.	Succès
TC-41	getRooms retourne 500 sur une erreur serveur.	roomService.getAllRooms a rejeté la promesse. res.status appelé avec 500 et res.send avec "Server error".	Succès
TC-42	getRoomAvailability retourne 200 et les créneaux.	roomService.getRoomAvailability appelé avec l'ID et la date corrects. res.json appelé avec l'objet formaté (roomId et availableSlots).	Succès
TC-43	getRoomAvailability retourne 500 si le service échoue.	roomService.getRoomAvailability a rejeté la promesse. res.status appelé avec 500 et res.json appelé avec l'objet d'erreur.	Succès
TC-44	seedDatabase retourne 201 au succès.	roomService.seedRooms appelé. res.status appelé avec 201 et res.json avec le message de succès.	Succès
TC-45	seedDatabase retourne 500 si l'initialisation échoue.	roomService.seedRooms a rejeté la promesse. res.status appelé avec 500 et res.send avec le message d'erreur du service ("Duplicate Key").	Succès



Couche de Service de Salle

I. Objectifs

L'objectif de ces tests est de vérifier la logique et l'interaction avec les modèles de données (Room et Booking) pour :

- Récupérer les salles avec et sans filtre de capacité.
- Calculer avec précision les créneaux horaires disponibles en tenant compte des réservations existantes.

II. Périmètre des Tests

Domaine	Description du Périmètre
Service de Salle	Fonctions getAllRooms, getRoomAvailability et seedRooms.
Services Mockés	Modèles Mongoose Room (find, insertMany) et Booking (find).
Scénarios	Consultation de la liste (sans/avec capacité, gestion des types), Consultation de disponibilité (libre, partiellement occupé, entièrement occupé, chevauchement), Initialisation (succès/échec).

III. Fiches de Tests Détaillées

ID Test	Fonction Testée	Description du Test	Résultat Attendu	Erreur/Statut Attendu
TC-46	getAllRooms	Succès de la récupération de toutes les salles sans spécifier de capacité.	Toutes les salles sont retournées.	Succès/R…
TC-47	getAllRooms	Succès de la récupération des salles avec un filtre de capacité numérique.	Les salles dont la capacité est supérieure ou égale au filtre sont retournées.	Succès/R…
TC-48	getAllRooms	Gestion d'une entrée de capacité fournie sous forme de chaîne de caractères.	Le filtre est appliqué correctement après conversion en nombre.	Succès/R…
TC-49	getRoomAvailability	Disponibilité complète : La salle est entièrement libre pour la journée (08:00 - 18:00).	Un seul créneau de 08:00 à 18:00 est retourné.	Succès/R…
TC-50	getRoomAvailability	Disponibilité partielle : Calcul des créneaux libres autour d'une seule réservation.	Deux créneaux distincts sont retournés : un avant et un après la réservation.	Succès/R…
TC-51	getRoomAvailability	Indisponibilité complète : La salle est entièrement réservée pour la journée.	Aucune créneau disponible n'est retourné.	Succès/R…
TC-52	getRoomAvailability	Disponibilité : Gestion de deux réservations séquentielles (calcul correct des intervalles).	Deux créneaux sont retournés, ignorant le temps occupé.	Succès/R…

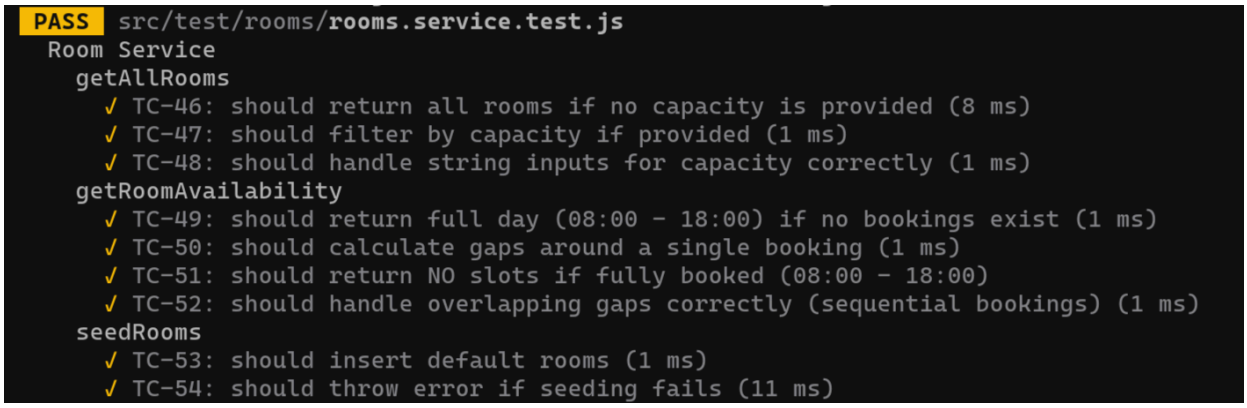
TC-53	seedRooms	Succès de l'initialisation : Insertion des salles par défaut.	Les salles par défaut sont insérées dans la base de données.	Succès/R…
TC-54	seedRooms	Échec de l'initialisation : Le processus d'insertion échoue.	Une exception est levée avec le message d'erreur de la base de données.	Échec/Exc…

IV. Exécution des Tests

Les tests unitaires de la couche de service de salle ont été exécutés avec **Succès**.

ID Test	Description du Test	Résultat Obtenu (Asserts Clés)	Statut
TC-46	Récupère toutes les salles si aucune capacité n'est fournie.	Room.find appelé avec {}. La liste des salles mockées est retournée.	Succès
TC-47	Filtre par capacité numérique.	Room.find appelé avec { capacity: { $gte: 15 } }. La liste des salles filtrées est retournée.	Succès
TC-48	Gère correctement les entrées de capacité en chaîne de caractères.	Room.find appelé avec { capacity: { $gte: 10 } }.	Succès
TC-49	Retourne la plage complète de 08:00 à 18:00 si aucune réservation n'existe.	Booking.find appelé. Un slot { startTime: 08:00, endTime: 18:00 } est retourné.	Succès
TC-50	Calcule deux créneaux autour d'une seule réservation (ex: 08:00-10:00 et 12:00-18:00).	La logique de calcul des créneaux a identifié et retourné les deux slots libres.	Succès

TC-51	Retourne aucun créneau si la salle est entièrement réservée de 08:00 à 18:00.	La liste des slots retournée est vide (length: 0).	Succès
TC-52	Gère correctement les chevauchements/réservations séquentielles pour déterminer les créneaux restants.	Le calcul des créneaux a correctement fusionné les temps occupés (09:00-11:00) et retourné les deux slots libres (08:00-09:00 et 11:00-18:00).	Succès
TC-53	Insère les salles par défaut lors de l'initialisation.	Room.insertMany appelé avec un tableau contenant les données des salles par défaut.	Succès
TC-54	Lève une erreur si l'initialisation de la base de données (insertMany) échoue.	Room.insertMany a rejeté la promesse. L'exception "DB Error" est correctement levée.	Succès



Tests d'Intégration

I. Objectifs

L'objectif de ces tests est de vérifier l'intégration complète des composants de l'application (contrôleurs, services, middlewares et base de données) et de garantir que le flux utilisateur final fonctionne comme attendu à travers les différentes routes API.

II. Périmètre des Tests

Domaine	Description du Périmètre
Authentification (/auth)	Inscription, Connexion, Gestion des erreurs (409, 412).
Gestion des Salles (/rooms)	Initialisation des données (seed) et consultation (avec authentification).
Flux de Réservation (/bookings)	Création, Vérification de conflit, Consultation par utilisateur, Annulation/Suppression.
Technologie	MongoDB/Mongoose, Express.js, JWT, Supertest.

III. Fiches de Tests Détaillées

ID Test	Fonction/Path Testée	Description du Test	Résultat Attendu	Statut HTTP Attendu
TC-55	POST /api/auth/register	Succès de l'enregistrement d'un nouvel utilisateur.	Statut 201 et succès dans le corps de la réponse.	201
TC-56	POST /api/auth/login	Succès de la connexion après enregistrement.	Statut 200 et retour d'un accessToken (JWT).	200
TC-57	POST /api/auth/register	Échec de l'enregistrement : l'e-mail est déjà utilisé.	Statut 409 et message "Email already used".	409
TC-58	POST /api/auth/register	Échec de l'enregistrement : données de validation manquantes (mot de passe manquant).	Statut 412 et message "Validation failed".	412
TC-59	POST /api/rooms/seed	Succès de l'initialisation des données de salles.	Statut 201 et confirmation de l'insertion d'au	201

			moins une salle.	
TC-60	GET /api/rooms	Succès de la récupération de la liste des salles (nécessite JWT).	Statut 200 et un tableau de salles est retourné.	200
TC-61	POST /api/bookings	Succès de la création d'une réservation avec des heures et une salle valides.	Statut 201 et la nouvelle réservation est retournée.	201
TC-62	POST /api/bookings	Échec de la création d'une réservation en conflit avec une réservation existante sur le même créneau horaire.	Statut 409 (ROOM_OCCUPIED).	409
TC-63	GET /api/bookings/my-bookings	Succès de la consultation de la liste des réservations de l'utilisateur connecté.	Statut 200 et un tableau contenant la ou les réservations de l'utilisateur est retourné.	200
TC-64	DELETE /api/bookings/:id	Succès de la suppression d'une réservation existante par l'utilisateur propriétaire.	Statut 200 (Suppression réussie).	200

IV. Exécution des Tests

Les tests d'intégration EasyBooking ont été exécutés avec **Succès**.

ID Test	Description du Test	Résultat Obtenu (Asserts Clés)	Statut
TC-55	Enregistrement réussi.	res.statusCode est 201, res.body.success est true.	Succès
TC-56	Connexion réussie.	res.statusCode est 200, res.body contient accessToken.	Succès

TC-57	Échec de l'enregistrement si l'e-mail existe déjà.	res.statusCode est 409, res.body.message est "Email already used".	Succès
TC-58	Échec de l'enregistrement si la validation échoue.	res.statusCode est 412, res.body.message est "Validation failed".	Succès
TC-59	Initialisation des salles réussie.	res.statusCode est 201, Room.countDocuments() est > 0.	Succès
TC-60	Récupération des salles réussie.	res.statusCode est 200, res.body.length est > 0.	Succès
TC-61	Création de réservation réussie.	res.statusCode est 201.	Succès
TC-62	Rejet de réservation en cas de chevauchement.	res.statusCode est 409.	Succès
TC-63	Consultation des réservations de l'utilisateur réussie.	res.statusCode est 200, res.body.length est 1.	Succès
TC-64	Suppression/Annulation de réservation réussie.	res.statusCode est 200.	Succès

```
PASS src/test/integration.test.js
  EasyBooking API Integration Tests
    ✓ TC-55: POST /api/auth/register - Should register a new user (236 ms)
    ✓ TC-56: POST /api/auth/login - Should return a JWT token (413 ms)
    ✓ TC-57: POST /api/auth/register - Should fail (409) if email exists (202 ms)
    ✓ TC-58: POST /api/auth/register - Should fail (412) if missing password (2 ms)
    ✓ TC-59: POST /api/rooms/seed - Should populate rooms (10 ms)
    ✓ TC-60: GET /api/rooms - Should retrieve rooms (8 ms)
    ✓ TC-61: POST /api/bookings - Should create a booking (132 ms)
    ✓ TC-62: POST /api/bookings - Should reject overlapping booking (154 ms)
    ✓ TC-63: GET /api/bookings/my-bookings - Should list user bookings (148 ms)
    ✓ TC-64: DELETE /api/bookings/:id - Should cancel booking (133 ms)
```

Tests de Sécurité & Pénétration

I. Objectifs

L'objectif principal est de vérifier que l'application :

1. Applique correctement les contrôles d'accès pour prévenir les actions non autorisées (ex: **Broken Access Control**).
2. Valide les entrées utilisateur pour empêcher les attaques par injection (**NoSQL Injection**, **XSS**).
3. Gère correctement les secrets et les sessions (**JWT forgery/expiration**).
4. Évite les fuites d'informations sensibles en cas d'erreur (**Information Leakage**).
5. Implémente une validation stricte des données pour prévenir les erreurs de traitement.

II. Périmètre des Tests

Domaine	Description du Périmètre
Authentification / JWT	Prévention de l'injection NoSQL (Login), rejet des tokens forgés et expirés, application d'une politique de mot de passe fort, non-retour de données sensibles dans la réponse.
Autorisation / Réservation	**Broken Access Control** (Test SEC-01) : un utilisateur ne peut pas modifier/supprimer la ressource d'un autre utilisateur.
Validation des Entrées	Rejet des charges utiles malveillantes (scripts XSS) et des types de données inattendus.
Résilience / Erreurs	Rejet des charges utiles trop volumineuses (413 Payload Too Large) et prévention de la fuite de traces de pile (stack traces) et de chemins de fichiers sur les erreurs serveur.

III. Fiches de Tests Détaillées

ID Test	Fonction Testée / Risque	Description du Test	Résultat Attendu	Statut HTTP Attendu
SEC-01	**Broken Access Control** (Réservation)	Un utilisateur (B) essaie de supprimer la réservation créée par un autre utilisateur (A).	Statut 404 ("Booking not found") ou 403 (Unauthorized) pour empêcher la fuite d'information sur l'existence de la ressource.	404

SEC-02	**NoSQL Injection** (Login)	Tentative de connexion avec un objet de requête NoSQL ({$ne: null}) dans le champ email.	Statut 412 (Validation failed) : le middleware de validation doit bloquer l'attaque.	412
SEC-03	**Falsification de Token** (JWT)	Tentative d'accès à une route protégée avec un token JWT créé mais signé avec une mauvaise clé secrète.	Statut 401 (Unauthorized) ou 403 (Forbidden) : le middleware auth doit détecter la signature invalide.	401/403
SEC-04	**Cross-Site Scripting (XSS)**	Tentative d'enregistrement d'un nom d'utilisateur contenant un script malveillant (<script>alert()</script>).	Statut 412 (Validation failed) : le middleware de validation doit rejeter les caractères spéciaux.	`!= 201`
SEC-05	**Fuite de Données Sensibles**	Vérification que les réponses de l'API (ex: register) ne contiennent pas de champs sensibles comme le mot de passe haché.	Le champ password (ou son équivalent) doit être absent ou indéfini dans le corps de la réponse.	`!= 500`
SEC-06	**Rejet de Token Expiré** (JWT)	Tentative d'accès à une route protégée avec un token JWT dont la date d'expiration est dépassée.	Statut 401 (Unauthorized) : le middleware auth doit détecter l'expiration.	401
SEC-07	**Politique de Mot de Passe Faible**	Tentative d'enregistrement d'un utilisateur avec un mot de passe non conforme aux règles de complexité ("123").	Statut 412 (Validation failed) : le validateur doit spécifier l'erreur sur le mot de passe.	412

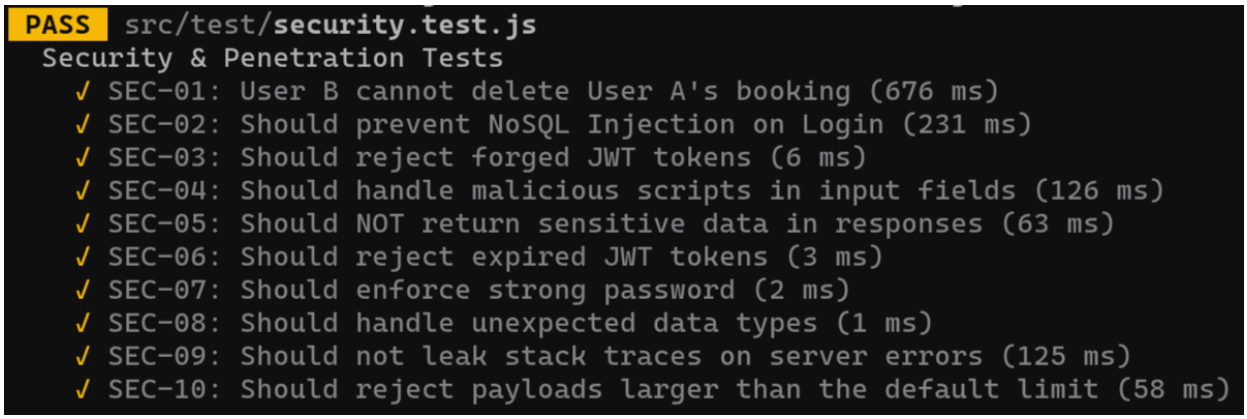
SEC-08	**Types de Données Inattendus**	Tentative d'enregistrement avec un champ de type incorrect (ex: username est un tableau au lieu d'une chaîne).	Statut 412 (Validation failed) ou autre échec : le serveur ne doit pas planter (!= 500) ni accepter la requête (!= 201).	`!= 201`
SEC-09	**Fuite de Trace de Pile** (Errors)	Tentative de déclencher une erreur serveur (ex: en utilisant un ID mal formé pour la suppression).	Le corps de la réponse (500/400) ne doit contenir aucune trace de pile, chemins de fichiers, ou dépendances (node_modules).	500/400
SEC-10	**Limitation de Taux / Charge Utile**	Tentative d'envoyer une charge utile (payload) excessivement volumineuse (10MB) au serveur.	Statut 413 (Payload Too Large) : le serveur doit rejeter la requête au niveau du middleware body-parser.	413

IV. Exécution des Tests

Les tests de sécurité et de pénétration ont été exécutés avec **Succès**.

ID Test	Description du Test	Résultat Obtenu (Asserts Clés)	Statut
SEC-01	Le contrôle d'accès empêche un utilisateur non propriétaire de supprimer la réservation.	attackRes.statusCode est 404 et le message "Booking not found" est retourné.	Succès
SEC-02	Le validateur rejette la tentative d'injection NoSQL.	res.statusCode est 412.	Succès
SEC-03	Le token forgé est rejeté.	res.statusCode est 401 ou 403.	Succès
SEC-04	La validation bloque les scripts malveillants.	res.statusCode est != 500.	Succès

SEC-05	Les données sensibles (mot de passe) ne sont pas exposées dans la réponse register.	Le champ password est undefined dans res.body.	Succès
SEC-06	Le token expiré est rejeté.	res.statusCode est 401.	Succès
SEC-07	La validation de mot de passe fort est appliquée.	res.statusCode est 412. L'erreur de validation (res.body.data.errors.password) est définie.	Succès
SEC-08	Les types de données inattendus sont gérés sans planter le serveur.	res.statusCode est != 201 et res.statusCode est géré (412).	Succès
SEC-09	Aucune fuite d'informations sensibles (traces de pile, chemins de fichiers, etc.) n'est détectée en cas d'erreur.	Le corps de la réponse ne correspond à aucune des expressions régulières de fuite d'information.	Succès
SEC-10	La charge utile trop volumineuse est rejetée.	res.statusCode est 413.	Succès



Flux Utilisateur E2E (Cypress)

I. Objectifs

L'objectif de ces tests est de valider le bon fonctionnement de l'application client (UI) et son intégration correcte avec les routes de l'API en couvrant :

1. Le flux utilisateur complet (enregistrement, connexion, recherche, réservation,

annulation).
2. L'affichage correct des données et les mises à jour de l'interface utilisateur.
3. La gestion des erreurs d'authentification.

II. Périmètre des Tests

Domaine	Description du Périmètre
Technologie	Test End-to-End avec Cypress.
Composants Testés	Pages : /register, /login, / (Dashboard), /my-bookings.
Intégration (Mockée)	Mocks des appels API (Auth, Rooms, Bookings) pour s'assurer des statuts de réponse et des données cohérentes pour l'UI.
Scénarios Couverts	Enregistrement/Connexion, Affichage de la liste/Filtrage, Création/Annulation de réservation, Gestion d'une erreur 401 (Login).

III. Fiches de Tests Détaillées

Fonction Testée / Flux	Description du Test	Résultat Attendu	Statut HTTP Attendu
Enregistrement	L'utilisateur s'enregistre avec des données valides.	L'appel @registerUser est effectué (201), une alerte "Account created" s'affiche, et la redirection vers /login a lieu.	201
Connexion	L'utilisateur se connecte avec les identifiants valides.	L'appel @loginUser est effectué (200), et la redirection vers / (Dashboard) a lieu.	200
Affichage Dashboard	Après la connexion, la page principale charge et affiche les salles disponibles.	L'appel @getRooms est effectué (200), et les détails des salles mockées (Salle A, Salle B) sont visibles.	200
Filtre de Recherche	L'utilisateur filtre la liste des salles par capacité (ex:	L'appel @getFilteredRooms est effectué, et seule la salle correspondant au filtre (Salle	200

	capacity=15).	A) reste visible sur l'UI.	
Ouverture Réservation	L'utilisateur clique sur le bouton pour réserver une salle.	L'appel @getAvailability est effectué (200) et le modal de réservation s'ouvre, affichant les créneaux disponibles.	N/A
Création Réservation	L'utilisateur sélectionne un créneau et confirme la réservation.	L'appel @createBooking est effectué (201) avec l'ID de la salle correcte, et une alerte "Room booked" s'affiche.	201
Mes Réservations	L'utilisateur navigue vers la page de ses réservations.	L'appel @getMyBookings est effectué (200), l'URL est /my-bookings, et la réservation confirmée est visible.	200
Annulation Réservation	L'utilisateur clique sur le bouton d'annulation.	L'appel @deleteBooking est effectué (200) avec l'ID de la réservation. La carte de réservation disparaît de l'UI.	200
Gestion Erreur Login	Tentative de connexion avec des identifiants invalides (l'interception est configurée pour retourner une erreur 401).	L'appel @loginFail est effectué (401), le message d'erreur (Invalid credentials) est affiché, et la redirection est bloquée.	401

IV. Exécution des Tests

Le script de tests E2E a été exécuté avec **Succès**.

Description du Test	Résultat Obtenu (Asserts Clés)	Statut
Enregistrement	cy.wait("@registerUser") et redirection vers /login.	Succès
Connexion	cy.wait("@loginUser") et redirection vers /.	Succès
Affichage Dashboard	cy.wait("@getRooms") et Salle A / Salle B visibles.	Succès

Filtre de Recherche	cy.wait("@getFilteredRooms") et Salle B n'est plus visible.	Succès
Ouverture Réservation	Le modal "Booking: Salle A" est visible.	Succès
Création Réservation	cy.wait("@createBooking") et le modal se ferme.	Succès
Mes Réservations	cy.wait("@getMyBookings") et la carte de la réservation est visible.	Succès
Annulation Réservation	cy.wait("@deleteBooking") et "No upcoming bookings found" est visible.	Succès
Gestion Erreur Login	cy.wait("@loginFail") (401) et le message d'erreur (Invalid credentials) est visible.	Succès

Rapport de Synthèse Qualité

I. Statut Général et Objectif

Statut Global : SUCCÈS

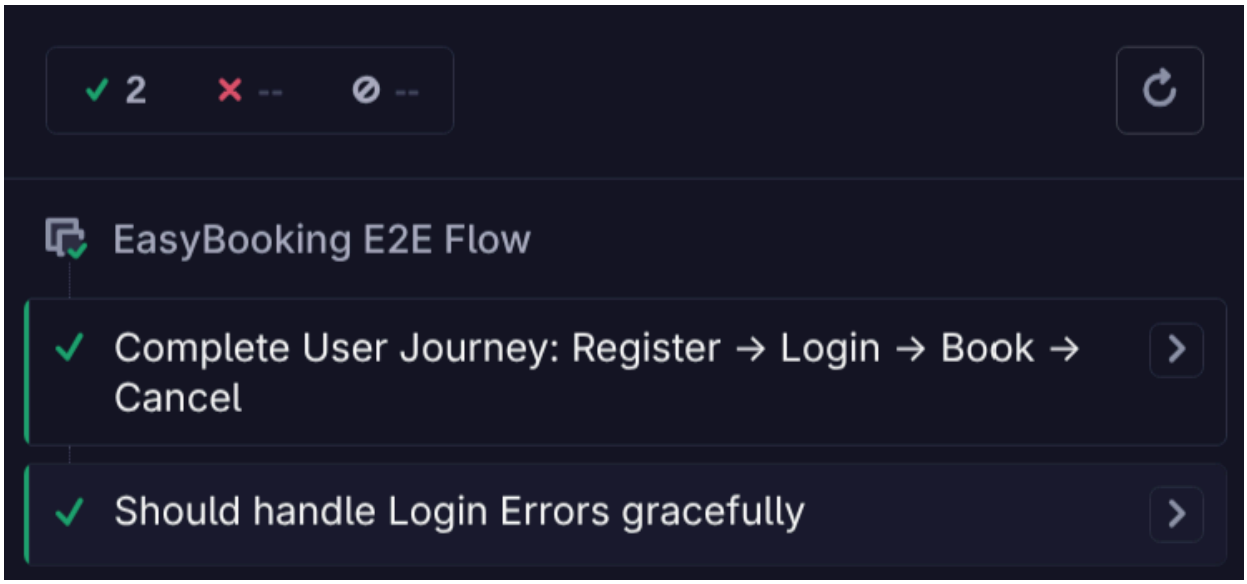
L'ensemble des **74 cas de test** critiques, tels que définis dans le Plan de Test Global, ont été exécutés avec succès. Cela valide la fonctionnalité, la robustesse et la sécurité des composants clés de l'application.

II. Synthèse de la Couverture de Test

Catégorie de Test	Nombre Total de Cas	Objectif	Statut
Unitaires	54	Validation isolée de la logique métier (Services, Contrôleurs, Middlewares).	**Succès**
Intégration	10	Validation de l'intégration API - Base de Données	**Succès**

		(Flux Register/Login, Réservation).	
Sécurité	10	Couverture des vulnérabilités critiques de l'API (Injection, Contrôle d'Accès, JWT).	**Succès**
E2E (Cypress)		Validation du parcours utilisateur complet et de la cohérence de l'interface client.	**Succès**
Total Global	**74 cas**		**Succès**

```
-----------------------|---------|----------|---------|---------|-------------------
File                   | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------------------|---------|----------|---------|---------|-------------------
All files              |   96.56 |       90 |   84.61 |   96.89 |
 server                |   78.94 |       25 |       0 |   83.33 |
  server.js            |   78.94 |       25 |       0 |   83.33 | 21-23
 server/src/config     |   57.14 |      100 |       0 |   57.14 |
  db.config.js         |   57.14 |      100 |       0 |   57.14 | 6-9
 server/src/controller |     100 |      100 |     100 |     100 |
  auth.controller.js   |     100 |      100 |     100 |     100 |
  ...ings.controller.js |     100 |      100 |     100 |     100 |
  rooms.controller.js  |     100 |      100 |     100 |     100 |
 server/src/middleware |     100 |      100 |     100 |     100 |
  auth.middleware.js   |     100 |      100 |     100 |     100 |
  ...tion.middleware.js |     100 |      100 |     100 |     100 |
 server/src/models     |     100 |      100 |     100 |     100 |
  booking.model.js     |     100 |      100 |     100 |     100 |
  room.model.js        |     100 |      100 |     100 |     100 |
  user.model.js        |     100 |      100 |     100 |     100 |
 server/src/routes     |     100 |       50 |     100 |     100 |
  auth.routes.js       |     100 |      100 |     100 |     100 |
  bookings.routes.js   |     100 |      100 |     100 |     100 |
  rooms.routes.js      |     100 |       50 |     100 |     100 | 8
 server/src/services   |   98.27 |    94.73 |     100 |   98.23 |
  auth.service.js      |     100 |      100 |     100 |     100 |
  bookings.service.js  |     100 |      100 |     100 |     100 |
  rooms.service.js     |   94.28 |       80 |     100 |   94.28 | 16,66
-----------------------|---------|----------|---------|---------|-------------------
Test Suites: 9 passed, 9 total
Tests:       74 passed, 74 total
```


✓ 2
✕ --
⊘ --
EasyBooking E2E Flow
✓ Complete User Journey: Register → Login → Book → Cancel
✓ Should handle Login Errors gracefully