## 🎯 Objectifs des Tests – Projet Propelize

Définir les objectifs de test est une étape essentielle du processus de test logiciel, car cela oriente l’ensemble de l’effort de validation et assure la cohérence entre les exigences du projet et les résultats attendus. Dans le cadre du développement de l’API Propelize, les tests visent à garantir que le produit est fiable, sécurisé, fonctionnel et conforme aux besoins métiers. Les objectifs spécifiques sont les suivants :

### ✅ 1. **Assurer la conformité avec les exigences métiers**

Les tests doivent vérifier que l’API Propelize répond fidèlement aux besoins définis dans les spécifications fonctionnelles : gestion des utilisateurs, gestion des véhicules, authentification et autorisation, etc.

### 🐞 2. **Détecter les défauts (bugs)**

Un objectif fondamental du processus de test est de détecter les défauts, qu’ils soient fonctionnels (ex. : mauvaise réponse à une requête POST /login) ou non fonctionnels (ex. : lenteur d’exécution, mauvaise gestion des erreurs).

### ⚙️ 3. **Vérifier le bon fonctionnement des fonctionnalités**

Chaque fonctionnalité de l’API (création d’un utilisateur, connexion, ajout de véhicule, etc.) doit être testée pour s’assurer qu’elle fonctionne correctement et retourne les résultats attendus, y compris en cas d’entrée incorrecte.

### 🌟 4. **Garantir la qualité globale du logiciel**

Les tests doivent vérifier que le code respecte les standards de qualité définis (structure modulaire, cohérence des réponses API, gestion des erreurs, respect des formats JSON, etc.).

### 🛡️ 5. **Réduire le risque de défaillance**

Grâce à une couverture de test adéquate, l’équipe pourra détecter les régressions ou comportements imprévus avant la mise en production, diminuant ainsi le risque de défaillances majeures.

### 🔐 6. **Renforcer la sécurité de l’application**

Les tests doivent vérifier que les mécanismes d’authentification et d’autorisation fonctionnent correctement (accès par token JWT, rafraîchissement de token, accès interdit sans credentials, etc.) et que les données utilisateurs sont protégées contre les accès non autorisés.

### 😀 7. **Améliorer la satisfaction utilisateur**

Enfin, le processus de test doit s’assurer que l’API permet une expérience utilisateur fluide, cohérente et conforme aux attentes, tant pour les développeurs frontend que pour les utilisateurs finaux via les interfaces.

## 🗂️ Portée des Tests – Projet Propelize

La portée des tests définit ce qui sera testé dans le cadre du projet Propelize, ainsi que les limites du processus de test. Elle permet de clarifier les responsabilités de l’équipe de test, d’éviter les zones d’ombre, et de concentrer les efforts sur les fonctionnalités critiques, tout en prenant en compte les contraintes de temps, de budget et de ressources humaines.

### ✅ En portée (**In Scope**)

Les éléments suivants seront testés dans ce projet :

#### 🔒 API Sécurité

* Authentification via POST /login
* Rafraîchissement de token via POST /refresh
* Middleware de protection (authenticate, authorize)
* Vérification de l'accès interdit sans token

#### 👤 Module Utilisateur (/users)

* Création d’un utilisateur (POST /users)
* Modification de profil (PUT /users/:id)
* Validation des champs (email, mot de passe, etc.)

#### 🚗 Module Véhicules (/vehicles)

* Récupération des véhicules (GET /vehicles)
* Création d’un véhicule (POST /vehicles)
* Vérification des droits d'accès aux données véhicules

#### ⚙️ Intégration avec la base de données

* Connexion Sequelize avec SQLite
* Test de persistance des données (création, suppression)
* Vérification des contraintes (types, valeurs nulles, etc.)

#### 🧪 Tests automatisés (Jest)

* Tests unitaires des fonctions critiques
* Tests d’intégration des routes principales
* Mocking des dépendances (ex. : base de données, JWT)

#### 🌐 Tests front-end (Playwright)

* Connexion via interface /login
* Affichage des véhicules après authentification
* Création d’un utilisateur via interface /users

#### 🔁 CI/CD

* Exécution automatique des tests via GitHub Actions à chaque pull request

### 🚫 Hors portée (**Out of Scope**)

Les éléments suivants ne seront pas testés pour le moment :

* Interface graphique complète (non prévue dans ce projet)
* Compatibilité mobile et tests multi-navigateurs
* Performances (load testing, stress testing)
* Tests de pénétration ou audit de sécurité avancé
* Modules futurs ou non définis dans les TPs actuels

### 🧩 Justification de la portée

Cette portée a été définie en fonction :

* des objectifs pédagogiques des TPs (INF352 ),
* de la complexité fonctionnelle actuelle de l'application,
* du temps disponible avant la deadline (04 juin 2025),
* et des ressources techniques et humaines accessibles.

Les tests prioritaires se concentrent sur les fonctionnalités critiques : sécurité, gestion des utilisateurs, intégrité des données véhicules, et validation des workflows API.

## 🧭 Approche de Test – Projet Propelize

Le choix d’une ou plusieurs approches de test est crucial pour assurer une couverture efficace des exigences tout en respectant les contraintes de temps, de ressources et de complexité du projet. Pour le projet Propelize, plusieurs approches de test seront combinées afin d’assurer à la fois la fiabilité fonctionnelle, la sécurité, et la qualité technique du code.

### ✅ 1. **Black-box Testing** (Test fonctionnel)

* Pourquoi ?  
  Permet de vérifier que l’API renvoie les bons résultats en fonction des entrées, sans se soucier de la manière dont ces résultats sont produits.
* Exemples concrets :
  + POST /login retourne un token pour des identifiants valides
  + GET /vehicles retourne la liste des véhicules autorisés

### ✅ 2. **White-box Testing** (Test structurel)

* Pourquoi ?  
  Utile pour tester la logique interne de certaines fonctions (par exemple : la validation d’entrée, le traitement du token, les contrôles de rôle).
* Où ?
  + Fichiers middleware/, utils/, controllers/

### ✅ 3. **Gray-box Testing**

* Pourquoi ?  
  Combine la vision externe et interne, pratique pour les membres de l’équipe ayant accès au code et aux spécifications.
* Exemples :
  + Tester le bon fonctionnement d’une route tout en analysant les conditions if, les appels à la base, etc.

### ✅ 4. **Automated Testing** (avec Jest & Playwright)

* Pourquoi ?
  + Gain de temps
  + Exécution continue via GitHub Actions
  + Documentation vivante des comportements attendus
* Outils utilisés :
  + Jest + Supertest pour l’API
  + Playwright pour le front-end (/login, /users, /vehicles)

### ✅ 5. **Manual Testing** (temporaire, complémentaire)

* Pourquoi ?
  + Utile dans les premières phases de test ou pour explorer rapidement une nouvelle fonctionnalité.
* Outils :
  + Postman, Insomnia, navigateur

### ✅ 6. **Regression Testing**

* Pourquoi ?
  + S’assurer qu’une nouvelle fonctionnalité (ex. ajout de la gestion des tokens) n’a pas cassé les anciennes (ex. GET /vehicles).
* Comment ?
  + Réexécuter les anciens tests automatiquement après chaque ajout de code

### 🚫 7. **Exploratory Testing** (Hors portée principale)

* Cette approche peut être utilisée ponctuellement, mais ne sera pas la base principale du plan de test, car le projet suit un cahier des charges bien défini avec des endpoints fixes.

### 🎯 Justification du choix des approches

Les approches ont été sélectionnées pour assurer :

* Une haute couverture fonctionnelle (Black-box)
* Une maîtrise du comportement interne (White-box)
* Une intégration dans un pipeline CI/CD (Automated testing)
* Une sécurité logicielle accrue (Gray-box sur les middleware d’authentification)

## 🧰 Ressources de Test – Projet Propelize

L'identification des ressources de test est une étape cruciale du processus de test logiciel. Elle permet de planifier de manière réaliste les activités de test en tenant compte des compétences disponibles, des outils nécessaires, de l’environnement technique, des données utiles et des moyens financiers. Pour le projet Propelize, les ressources ont été réparties comme suit :

### 👥 1. **Ressources humaines (Personnel)**

Le projet est réalisé par un groupe de sept étudiants, répartis selon les rôles suivants :

|  |  |  |
| --- | --- | --- |
| Membres | Rôle principal | Détail des responsabilités |
| 4 membres | Développement des APIs | Conception, implémentation, sécurisation des endpoints |
| 3 membres | Tests & validation | Écriture des cas de test, exécution, CI/CD, Playwright |

L’organisation du travail permet une spécialisation tout en assurant la collaboration entre développeurs et testeurs. Les testeurs participent également à la revue de code et à la documentation des tests.

### 🧪 2. **Outils de test (Testing Tools)**

Les outils utilisés pour l'exécution et l'automatisation des tests sont :

|  |  |
| --- | --- |
| Outil | Usage prévu |
| Jest | Tests unitaires et d’intégration côté backend |
| Supertest | Simulation des requêtes HTTP pour tester l’API |
| Playwright | Tests automatisés des interfaces front-end |
| GitHub Actions | Intégration continue : exécution des tests à chaque push/pull request |
| Postman | Tests manuels, exploration des endpoints |

### 🖥️ 3. **Infrastructure technique**

Les éléments suivants composent l’environnement de test :

|  |  |
| --- | --- |
| Infrastructure | Détail |
| Serveur local | Exécution de l’API localement avec Node.js |
| Base de données | SQLite (locale) – utilisable directement sans serveur |
| Docker (à venir) | Conteneurisation de l’API pour simulation de déploiement |
| GitHub | Gestion de code, collaboration, CI/CD |

### 📊 4. **Jeux de données (Test Data)**

Des données fictives sont utilisées pour simuler les cas réels d’utilisation :

* Utilisateurs de test : emails, mots de passe
* Véhicules de test : marques, modèles, années
* Jetons JWT de test pour simuler les scénarios sécurisés

Les données sont générées manuellement ou via des seeders selon les besoins et sont contenus dans le fichier **data.txt**.

### 💰 5. **Budget**

En tant que projet académique, aucun budget monétaire n’est nécessaire. Les ressources utilisées sont :

* Outils open source ou gratuits
* Ordinateurs personnels des membres
* GitHub gratuit pour projets publics
* Temps de travail réparti selon les disponibilités de chacun

### 🔷 Tests unitaires – Module User

### 🧪 CT-UU-01 – Vérification du hachage du mot de passe (createUser)

1. Objectif : S'assurer que bcrypt.hash() est bien utilisé pour hacher le mot de passe avant stockage.
2. Entrée : password = "azerty123"
3. Sortie attendue : Le mot de passe enregistré doit être un hash différent de celui d'origine, vérifiable par bcrypt.compare().
4. Étapes : Mock de User.create, appel de createUser(), vérification du hash généré.
5. Révision : Sécurité essentielle.
6. Groupe : createUser
7. Priorité : 🔴 Critique
8. Couverture : Fonction de hachage

### 🧪 CT-UU-02 – Gestion de l’utilisateur inexistant (loginUser)

1. Objectif : Vérifier que User.findOne retourne une erreur 404 si l'utilisateur n'existe pas.
2. Entrée : name = "unknown"
3. Sortie attendue : Statut 404 Not Found
4. Étapes : Mock de User.findOne → null, appel de loginUser, vérification du code retour.
5. Révision : Test d’accès sécurisé.
6. Groupe : loginUser
7. Priorité : 🔴 Critique
8. Couverture : Recherche d’utilisateur

### 🧪 CT-UU-03 – Suppression du refreshToken (logoutUser)

1. Objectif : Vérifier que refreshToken est bien remis à null et sauvegardé.
2. Entrée : refreshToken présent dans le cookie
3. Sortie attendue : user.refreshToken === null
4. Étapes : Mock de User.findOne, appel de logoutUser, vérification du champ et de l’appel à save().
5. Révision : Sécurité des sessions
6. Groupe : logoutUser
7. Priorité : 🟡 Moyenne
8. Couverture : Logique de déconnexion

### 🧪 CT-UU-04 – Token invalide dans refreshExpiredToken

1. Objectif : S’assurer que le serveur rejette un token corrompu ou non associé.
2. Entrée : Token faux ou utilisateur non trouvé
3. Sortie attendue : Statut 403 Forbidden
4. Étapes : Mock de User.findOne, falsification du token, appel de refreshExpiredToken
5. Révision : Robuste contre manipulation JWT
6. Groupe : refreshExpiredToken
7. Priorité : 🔴 Critique
8. Couverture : JWT.verify, contrôle d’intégrité

### 🧪 CT-UU-05 – Mise à jour du nom d’utilisateur (updateUserName)

1. Objectif : Vérifier que le nom d’utilisateur peut être modifié et sauvegardé
2. Entrée : id=1, name="nouveauNom"
3. Sortie attendue : Objet utilisateur avec name="nouveauNom"
4. Étapes : Mock de User.findByPk, appel de updateUserName, vérification des changements
5. Révision : Fonction de mise à jour validée
6. Groupe : updateUserName
7. Priorité : 🟡 Moyenne
8. Couverture : Modification des champs de profil

### 🧪 CT-UU-06 – Refus d’un rôle invalide (updateUserRole)

1. Objectif : S'assurer qu’un rôle autre que user ou admin est refusé
2. Entrée : role = "superadmin"
3. Sortie attendue : Statut 400 Bad Request
4. Étapes : Mock de User.findByPk, appel updateUserRole, vérification de la validation du rôle
5. Révision : Contrôle des droits
6. Groupe : updateUserRole
7. Priorité : 🟡 Moyenne
8. Couverture : Validation métier des rôles

### 🔷 Tests d’intégration – Module User

### 🧪 CT-IU-01 – Création complète d’un utilisateur

1. Objectif : Vérifier que la route /users crée un utilisateur, génère les tokens, les envoie dans les cookies et retourne le bon code.
2. Entrée :

{

"name": "ideme",

"password": "azerty123",

"role": "admin"

}

1. Sortie attendue :

* HTTP 201 Created
* Objet utilisateur en réponse
* Cookies accessToken, refreshToken définis

1. Étapes :

* Simuler une requête POST /users
* Vérifier le contenu JSON de la réponse
* Vérifier la présence des cookies

1. Révision : Fonction de base de l’API
2. Groupe : createUser
3. Priorité : 🔴 Critique
4. Couverture : Route /users, middleware, sécurité

### 🧪 CT-IU-02 – Connexion réussie d’un utilisateur

1. Objectif : Vérifier que la route /login fonctionne avec de bons identifiants.
2. Entrée :

{

"name": "ideme",

"password": "azerty123"

}

1. Sortie attendue : HTTP 200, tokens en JSON et cookies
2. Étapes :

* Créer un utilisateur avec le mot de passe haché
* Requête POST /login
* Vérifier les tokens dans la réponse + cookies

1. Révision : Authentification de base
2. Groupe : loginUser
3. Priorité : 🔴 Critique
4. Couverture : Route /login, tokens

### 🧪 CT-IU-03 – Déconnexion d’un utilisateur

1. Objectif : Tester la route /logout pour qu’elle efface les tokens (cookie + BD)
2. Entrée : Cookie refreshToken valide
3. Sortie attendue : Message de confirmation + cookies supprimés
4. Étapes :

* Se connecter → Obtenir les cookies
* Requête POST /logout avec cookie
* Vérifier suppression token en BD + cookies vides

1. Révision : Fin de session sécurisée
2. Groupe : logoutUser
3. Priorité : 🟡 Moyenne
4. Couverture : Auth, cookies, BD

### 🧪 CT-IU-04 – Rafraîchissement d’un accessToken

1. Objectif : Vérifier que le serveur renouvelle l’accessToken à partir d’un refreshToken valide
2. Entrée : Cookie refreshToken
3. Sortie attendue : Nouveau accessToken dans la réponse + cookie mis à jour
4. Étapes :

* Se connecter → extraire cookie refreshToken
* Requête POST /refresh
* Vérifier réponse et contenu du nouveau cookie

1. Révision : Sécurité du jeton maintenue
2. Groupe : refreshExpiredToken
3. Priorité : 🔴 Critique
4. Couverture : JWT, sécurité

### 🧪 CT-IU-05 – Mise à jour du nom

1. Objectif : Tester la route PUT /users/:id/name pour modifier le nom
2. Entrée :

{

"name": "Ornella"

}

1. Sortie attendue : HTTP 200 + nouvel utilisateur avec nom mis à jour
2. Étapes :

* Créer un utilisateur
* Requête PUT avec ID et nouveau nom
* Vérifier réponse et valeur modifiée

1. Révision : Fonction de mise à jour
2. Groupe : updateUserName
3. Priorité : 🟡 Moyenne
4. Couverture : Modification utilisateur

### 🧪 CT-IU-06 – Mise à jour du rôle d’un utilisateur

1. Objectif : Tester la mise à jour du champ role via /users/:id/role
2. Entrée :

{

"role": "admin"

}

1. Sortie attendue : HTTP 200, utilisateur avec role: admin
2. Étapes :

* Créer un utilisateur basique
* Envoyer une requête PUT avec rôle admin
* Vérifier que la réponse retourne le nouveau rôle

1. Révision : Gestion des privilèges
2. Groupe : updateUserRole
3. Priorité : 🟡 Moyenne
4. Couverture : Contrôle rôle utilisateur

### 🔷 Tests unitaires – Module Vehicle

### 🧪 CT-UV-01 – Création de véhicule valide

1. Objectif : Vérifier que la fonction createVehicule crée un véhicule avec les données valides.
2. Entrée : Données complètes : registrationNumber, make, model, year, rentalPrice
3. Sortie attendue : Appel à Vehicle.create() avec les bons paramètres
4. Étapes :
   * Mock de Vehicle.create
   * Appel de createVehicule
   * Vérification des appels avec données
5. Révision : Bon fonctionnement du point de création
6. Groupe : createVehicule
7. Priorité : 🔴 Critique
8. Couverture : Création objet BD

### 🧪 CT-UV-02 – Suppression d’un véhicule par ID

1. Objectif : Vérifier que la fonction deleVehiculeById supprime bien un véhicule existant.
2. Entrée : id = 1
3. Sortie attendue : Appel à destroy() sur une instance de véhicule
4. Étapes :
   * Mock de Vehicle.findByPk
   * Appel de deleVehiculeById
   * Vérification de destroy() appelé
5. Révision : Test logique de suppression
6. Groupe : deleVehiculeById
7. Priorité : 🔴 Critique
8. Couverture : Suppression conditionnelle

### 🧪 CT-UV-03 – Recherche de véhicule par prix valide

1. Objectif : Vérifier que les véhicules sont recherchés selon un prix ≤ donné.
2. Entrée : price = 50000
3. Sortie attendue : Appel de findAll avec condition Op.lte
4. Étapes :
   * Mock ValuidNumber, Vehicle.findAll
   * Appel searchByPrice
   * Vérification du filtre et des résultats
5. Révision : Test de filtrage valide
6. Groupe : searchByPrice
7. Priorité : 🟡 Moyenne
8. Couverture : Validation + recherche

### 🧪 CT-UV-04 – Recherche par numéro d’immatriculation

1. Objectif : Vérifier que la recherche de véhicule par numéro fonctionne
2. Entrée : registrationNumber = "ABC123"
3. Sortie attendue : Appel findOne avec condition sur registrationNumber
4. Étapes :
   * Mock de ValuidNumber et Vehicle.findOne
   * Appel de searchByRegistrationNumber
   * Vérification de l’appel
5. Révision : Test de recherche exacte
6. Groupe : searchByRegistrationNumber
7. Priorité : 🟡 Moyenne
8. Couverture : Validation + filtre exact

### 🧪 CT-UV-05 – Mise à jour complète d’un véhicule

1. Objectif : Vérifier que les champs modifiables sont mis à jour conditionnellement
2. Entrée : Tous les champs présents dans le corps (ex: make, model, ...)
3. Sortie attendue : Appel de save() après modification conditionnelle
4. Étapes :
   * Mock de Vehicle.findByPk
   * Appel updateVehiculeById
   * Vérification des champs modifiés + save
5. Révision : Test complet de mise à jour
6. Groupe : updateVehiculeById
7. Priorité : 🔴 Critique
8. Couverture : Mise à jour conditionnelle

## 🔷 Tests d’intégration – Module Vehicle

### 🧪 CT-IV-01 – Création d’un véhicule

1. Objectif  
   Vérifier que la route POST /vehicles permet la création d’un véhicule à partir des données fournies.
2. Entrée

{

"registrationNumber": "AB123CD",

"make": "Toyota",

"model": "Corolla",

"year": 2022,

"rentalPrice": 50

}

1. Sortie attendue

* Statut 201 Created
* JSON contenant les données du véhicule créé

1. Étapes à exécuter

* Simuler une requête POST /vehicles avec les données
* Vérifier que la réponse contient les données du véhicule
* Vérifier le statut de réponse

1. Document du test  
   Objectif : Vérifier la création avec insertion BD  
   Entrée : véhicule complet  
   Sortie : véhicule créé  
   Étapes : POST → Vérif contenu + code
2. Révision et correction  
   Valider les erreurs possibles (incomplétude, duplicata...)
3. Groupe  
   Route /vehicles → API de création
4. Priorité  
   🔴 Critique
5. Couverture  
   Insertion BD, réponse API, validation

### 🧪 CT-IV-02 – Suppression d’un véhicule existant

1. Objectif  
   Tester la route DELETE /vehicles/:id pour supprimer un véhicule existant
2. Entrée

* ID d’un véhicule valide (ex: 1)

1. Sortie attendue

* Statut 200 OK
* Message de confirmation

1. Étapes à exécuter

* Créer un véhicule
* Appeler DELETE /vehicles/1
* Vérifier le message de retour
* Vérifier que l’objet n’existe plus en BD

1. Document du test  
   Objectif : Test suppression API  
   Entrée : ID valide  
   Sortie : statut + message  
   Étapes : POST → DELETE → GET pour vérifier
2. Révision  
   Gérer les erreurs de suppression d’un ID inexistant
3. Groupe  
   Route /vehicles/:id
4. Priorité  
   🔴 Critique
5. Couverture  
   Suppression logique + BD

### 🧪 CT-IV-03 – Recherche de véhicules par prix

1. Objectif  
   Tester la route GET /vehicles/search/:price pour filtrer les véhicules selon leur prix
2. Entrée

* URL : /vehicles/search/50000

1. Sortie attendue

* Statut 200 OK
* Liste de véhicules avec rentalPrice <= 50000

1. Étapes à exécuter

* Créer plusieurs véhicules à divers prix
* Appeler la route
* Vérifier que seuls ceux < 50000 sont retournés

1. Document du test  
   Objectif : Test de filtre API  
   Entrée : paramètre URL  
   Sortie : liste filtrée  
   Étapes : seed → GET → analyse JSON
2. Révision  
   Tester absence de résultat ou valeur invalide
3. Groupe  
   Route /vehicles/search/:price
4. Priorité  
   🟡 Moyenne
5. Couverture  
   Filtrage côté base + validation

### 🧪 CT-IV-04 – Recherche par numéro d’immatriculation

1. Objectif  
   Tester GET /vehicles/registration/:number
2. Entrée

* URL : /vehicles/registration/AB123CD

1. Sortie attendue

* Statut 200 OK
* JSON avec le véhicule correspondant

1. Étapes à exécuter

* Créer un véhicule avec registrationNumber = "AB123CD"
* Appeler la route
* Vérifier que le bon objet est retourné

1. Document du test  
   Objectif : Recherche exacte  
   Entrée : paramètre  
   Sortie : JSON ou 404  
   Étapes : POST → GET → comparaison
2. Révision  
   Ajouter des cas pour immatriculation invalide
3. Groupe  
   Route /vehicles/registration/:number
4. Priorité  
   🟡 Moyenne
5. Couverture  
   Recherche exacte

### 🧪 CT-IV-05 – Mise à jour d’un véhicule

1. Objectif  
   Tester la route PUT /vehicles/:id pour modifier un ou plusieurs champs
2. Entrée

{

"make": "Nissan",

"model": "Sentra",

"rentalPrice": 39000

}

1. Sortie attendue

* Statut 200 OK
* Données modifiées du véhicule

1. Étapes à exécuter

* Créer un véhicule
* Modifier certains champs avec PUT
* Vérifier les champs modifiés en base

1. Document du test  
   Objectif : modification API  
   Entrée : JSON partiel  
   Sortie : nouveau véhicule  
   Étapes : POST → PUT → GET
2. Révision  
   Vérifier que les champs non envoyés ne sont pas modifiés
3. Groupe  
   Route /vehicles/:id
4. Priorité  
   🔴 Critique
5. Couverture  
   Modification conditionnelle, mise à jour BD

### 🔷 Tests unitaires – Module Middleware

### 🧪 CT-UM-01 – Jeton manquant (authenticate)

1. Objectif : Vérifier que l’accès est refusé si aucun accessToken n’est présent.
2. Entrée : Requête sans cookie accessToken
3. Sortie attendue : HTTP 401 + message "No token provided"
4. Étapes :
   * Mock de req.cookies vide
   * Appel authenticate
   * Vérification de la réponse
5. Révision : Cas de sécurité critique
6. Groupe : authenticate
7. Priorité : 🔴 Critique
8. Couverture : Présence de token

### 🧪 CT-UM-02 – Jeton invalide (authenticate)

1. Objectif : Rejeter les tokens JWT non valides ou expirés.
2. Entrée : Cookie contenant un token invalide
3. Sortie attendue : HTTP 401 + message "Invalid token"
4. Étapes :
   * Mock de jwt.verify → erreur
   * Appel middleware
   * Vérification réponse
5. Révision : Sécurité JWT
6. Groupe : authenticate
7. Priorité : 🔴 Critique
8. Couverture : Vérification JWT

### 🧪 CT-UM-03 – Jeton valide (authenticate)

1. Objectif : Permettre l’accès quand un token JWT est correct.
2. Entrée : JWT valide dans cookie
3. Sortie attendue : Appel à next(), req.userId défini
4. Étapes :
   * Mock de jwt.verify → ID simulé
   * Vérification de l’appel à next() et des champs dans req
5. Révision : Accès sécurisé
6. Groupe : authenticate
7. Priorité : 🔴 Critique
8. Couverture : Passage de contrôle

### 🧪 CT-UM-04 – Accès refusé si rôle non autorisé (authorize)

1. Objectif : S’assurer que l’accès est bloqué si le rôle n’est pas autorisé.
2. Entrée : req.role = "user", authorize("admin")
3. Sortie attendue : HTTP 403 "Forbidden"
4. Étapes :
   * Middleware avec req.role = "user"
   * Vérifier que la réponse est bloquée
5. Révision : Contrôle de privilèges
6. Groupe : authorize
7. Priorité : 🔴 Critique
8. Couverture : Sélection de rôle

### 🧪 CT-UM-05 – Accès autorisé si rôle permis (authorize)

1. Objectif : Permettre l’accès quand le rôle correspond à un rôle autorisé.
2. Entrée : req.role = "admin", authorize("admin")
3. Sortie attendue : Appel next()
4. Étapes :
   * Injecter req.role = "admin"
   * Appeler le middleware authorize("admin")
   * Vérifier que next() est bien appelé
5. Révision : Cas positif
6. Groupe : authorize
7. Priorité : 🔴 Critique
8. Couverture : Filtrage autorisé

### 🔷 Tests d’intégration – Module Middleware

#### 🧪 CT-IM-01 – Requête sans accessToken

1. Objectif : Vérifier qu'une requête sans accessToken est rejetée avec un code 401.
2. Entrée : Requête HTTP sans cookie
3. Sortie attendue : Statut 401 Unauthorized
4. Étapes :
   * Envoyer une requête protégée sans token
   * Vérifier que la réponse est 401
5. Groupe : authenticate
6. Priorité : 🔴 Critique
7. Couverture : Vérification d’accès sans authentification

#### 🧪 CT-IM-02 – Jeton JWT invalide

1. Objectif : Vérifier qu’une requête avec token invalide retourne une erreur.
2. Entrée : Token invalide dans le cookie accessToken
3. Sortie attendue : Statut 401 Unauthorized
4. Étapes :
   * Envoyer une requête avec un token corrompu
   * Vérifier la réponse 401
5. Groupe : authenticate
6. Priorité : 🔴 Critique
7. Couverture : JWT invalide

#### 🧪 CT-IM-03 – Jeton JWT valide

1. Objectif : Vérifier qu’une requête avec token valide passe le middleware.
2. Entrée : Cookie contenant un JWT valide
3. Sortie attendue : Accès autorisé (statut 200)
4. Étapes :
   * Authentifier un utilisateur
   * Utiliser le cookie dans une requête sécurisée
   * Vérifier l’accès
5. Groupe : authenticate
6. Priorité : 🔴 Critique
7. Couverture : Cas d’usage réussi

#### 🧪 CT-IM-04 – Refus d’accès pour rôle interdit

1. Objectif : Vérifier que l’accès est bloqué si le rôle n’est pas autorisé.
2. Entrée : req.role = "user", middleware authorize("admin")
3. Sortie attendue : Statut 403 Forbidden
4. Étapes :
   * Authentifier avec rôle "user"
   * Accéder à une route protégée pour "admin"
   * Vérifier le refus d’accès
5. Groupe : authorize
6. Priorité : 🔴 Critique
7. Couverture : Contrôle des autorisations

#### 🧪 CT-IM-05 – Autorisation d’un rôle valide

1. Objectif : Vérifier que l’accès est accordé si le rôle est autorisé.
2. Entrée : req.role = "admin", middleware authorize("admin")
3. Sortie attendue : Statut 200 OK
4. Étapes :
   * Authentifier un admin
   * Accéder à une route pour "admin"
   * Vérifier que l’accès est autorisé
5. Groupe : authorize
6. Priorité : 🔴 Critique
7. Couverture : Passage autorisé

## 🔻 Stop Testing Criteria – Projet Propelize

Les tests du projet Propelize seront arrêtés lorsque les conditions suivantes seront réunies. Ces critères ont été définis en accord avec les objectifs du projet, les ressources disponibles et la stratégie de couverture définie dans le plan de test.

### ✅ 1. **Taux de réussite des cas de test**

* 95 % des tests unitaires et tests d’intégration doivent être réussis.
* Tous les tests critiques (priorité ) doivent être 100 % réussis.

### ✅ 2. **Taux de couverture des fonctionnalités**

* 100 % des fonctionnalités principales (authentification, gestion utilisateurs, gestion véhicules, middleware) doivent avoir au moins un test unitaire ou d’intégration validé.
* Toutes les routes protégées doivent être testées avec des rôles valides/invalides.

### ✅ 3. **Absence de défauts bloquants ou critiques**

* Aucun bug de niveau critique ou bloquant ne doit rester ouvert.
* Tous les défauts de priorité moyenne doivent avoir une solution ou un contournement documenté.

### ✅ 4. **Performance et stabilité**

* Le système doit répondre en moins de 2 secondes sur toutes les requêtes REST testées.
* Aucun crash ou fuite mémoire ne doit être détecté lors des tests prolongés.

### ✅ 5. **Achèvement du plan de test**

* Tous les cas de test documentés dans le plan (CT-UU-01 à CT-IV-05, etc.) ont été exécutés au moins une fois.
* Les résultats sont documentés et revus par les testeurs responsables.

### ✅ 6. **Acceptation par les parties prenantes**

* Le responsable qualité ou enseignant valide que le logiciel peut être livré ou présenté.
* La checklist d’acceptation est signée.

## 🤖 Test Automation – Projet Propelize

### 🎯 Objectif

L’objectif de l’automatisation est de garantir une exécution rapide, reproductible et fiable des cas de test critiques et récurrents, notamment les tests unitaires, d’intégration et de sécurité liés à l’API REST du projet.

### 🛠️ Outils utilisés

|  |  |
| --- | --- |
| Outil | Usage principal |
| Jest | Framework principal de test unitaire/integration |
| Supertest | Simulation des requêtes HTTP sur Express.js |
| nodemon | Rechargement automatique pendant le dev |
| dotenv | Gestion des variables d’environnement |

### 🧪 Types de tests automatisés

|  |  |
| --- | --- |
| Type de test | Détails |
| ✅ Tests unitaires | Couvrent les fonctions critiques des modules user, vehicle, middleware |
| ✅ Tests d’intégration | Testent les routes REST de bout en bout avec données réelles simulées |
| 🚫 Tests end-to-end | // a venir |
| 🚫 Tests UI | // a venir |

### 📜 Structure des scripts automatisés

Le dossier /tests est structuré de la manière suivante :

/tests

├── user/

│ ├── createUser.test.js

│ ├── loginUser.test.js

│ └── ...

├── vehicle/

│ ├── createVehicle.test.js

│ └── ...

├── middleware/

│ ├── authenticate.test.js

│ └── authorization.test.js

Chaque fichier correspond à un module testé et contient des cas de test documentés avec leur identifiant (ex : CT-IU-01, CT-UV-03…).

### ⚙️ Stratégie d’automatisation

* Tous les tests sont exécutables via un seul script :

bash

CopyEdit

npm test

* Les tests sont automatiquement lancés avant toute livraison ou validation finale.
* Les tests critiques sont prioritairement automatisés (auth, sécurité, accès rôle…).

### 📅 Fréquence d’exécution

|  |  |
| --- | --- |
| Moment | Action |
| 🧪 En développement | À chaque changement majeur (via jest) |
| 🔁 En pré-livraison | Tous les tests unitaires et d’intégration |
| ✅ En validation finale | Check global : couverture + statut |