***Plan de test***

***PROPELIZE***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Référence du document :** | INF3521 |  | **Validé par :** | Groupe 11 |
| **Version du document :** | 1.05 |  | **Validé le :** | 13/06/2025 |
| **Date du document :** | **14/06/2025** |  | **Soumis le :** | 12/06/2025 |
| **Auteur(s) :** | Groupe 11 |  | **Type de diffusion :** | Document  électronique (.PDF) |
|  |  |  | **Confidentialité :** |  |

**Mots clés** : mock, CRUD, …

## Sommaire

[Sommaire 2](#_Toc200705723)

[1.Introduction (ou préambule) 3](#_Toc200705724)

[1.1. Objectifs et méthodes 3](#_Toc200705725)

[1.2. Documents de référence 4](#_Toc200705726)

[2.Tests Unitaires 4](#_Toc200705727)

[2.1. Pour chaque scenario : 4](#_Toc200705728)

[2.1.1. Identification 4](#_Toc200705729)

[2.1.2. Description 4](#_Toc200705730)

[2.1. 3. Dépendances 7](#_Toc200705731)

[3.Tests d’intégration 8](#_Toc200705732)

[3.1. Pour chaque test d’intégration : 8](#_Toc200705733)

[3.1.1. Identification 8](#_Toc200705734)

[3.1.2. Description 8](#_Toc200705735)

[3.1.3. Dépendances 8](#_Toc200705736)

[3.1.5. Procédure de test 9](#_Toc200705737)

[4.Tests d’interface 17](#_Toc200705738)

[4.1. Pour le test de chaque interface : 17](#_Toc200705739)

[4.1.1. Identification 17](#_Toc200705740)

[4.1.2. Procédure de Test 17](#_Toc200705741)

[19](#_Toc200705742)

[4.1.5. Description 19](#_Toc200705743)

[5.Vérification de la documentation 19](#_Toc200705744)

# 1.Introduction (ou préambule)

Ce plan de test couvre les aspects Backend (API) et Frontend (Interface) de l’application Propelize. Propelize est une application de location de véhicule en ligne qui permet à un utilisateur de créer un compte (admin ou visiteur), s’y connecter, ajouter un véhicule, supprimer un véhicule, rechercher un véhicule selon des paramètres bien définis.

### 1.1. Objectifs et méthodes

Notre logiciel a été conçu suivant les méthodes Agile basé sur le Scrum et ce document vise à atteindre les objectifs de tests fonctionnels suivants :

* API
  + Créer un compte
  + Se connecter
  + Créer un véhicule
  + Modifier un véhicule
  + Afficher tous les véhicules
  + Rechercher les véhicules par ID
  + Rechercher un véhicule par le numéro d’enregistrement
  + Récupérer tous les véhicules loués par prix
  + Supprimer un véhicule
  + Vérifier le statut d’un véhicule
* Test d’intégration
* Interface
  + Login
  + Dashboard
  + Gestion des utilisateurs

### 1.2. Documents de référence

Pour ce projet les documents que nous prendrons comme référence sont :

* Les fichiers index.html montrant le rapport de la couverture des tests unitaires de l’API et des interfaces.

# 2.Tests Unitaires

Nous décrirons les différents scénarios utilisés pour nos tests pour chaque module mis en place

### 2.1. Pour chaque scenario :

#### 2.1.1. Identification

Il se présente selon une répartition bien définie

* Test sur authentification
* Test de création d’un utilisateur
* Test de connexion d’un utilisateur

#### 2.1.2. Description

* Test sur l ’authentification :

Il est question ici de vérifier qu’un utilisateur peut être crée sans encombre. Notre test vérifie que les données entrées sont conformes. Il simule la création d’un utilisateur et vérifie si le nom d’utilisateur, l’id, l’email et le rôle sont bien retranscris. S’ils sont bien retranscrits selon la norme du programme, alors la requête est validée. Puis il simule la connexion de cet utilisateur.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Description | Entrée | Sortie Attendue | Méthode de Test | Remarque |
| 01 | Vérification du fonctionnement lorsqu’on veut créer un utilisateur vide en créant un utilisateur | Aucune | Message d’erreur du serveur « Données d'entrée invalides » | Nous créons une instance vide d’un utilisateur et on vérifie son fonctionnement. | Aucune |
| 02 | Vérification du fonctionnement lorsqu’un champ du formulaire est oublié en créant un utilisateur | Données valide avec omission d’un champ obligatoire | Message d’erreur du serveur « Données d'entrée invalides » | Nous créons une instance d’un utilisateur en omettant de remplir un champ obligatoire comme l’email | Aucune |
| 03 | Vérification du fonctionnement lorsqu’un champ est rempli avec des données incorrectes en créant un utilisateur | Données invalides | Message d’erreur du serveur | Nous créons un utilisateur mais en entrant des entrées invalides sur ‘User’ | Aucune |
| 04 | Vérification du fonctionnement lorsque nous entrons des données valides pour la création d’utilisateur | Données valides (username, email, role) | Message de validation du serveur « *Utilisateur enregistré avec succès* » | Nous insérons des données respectant les règles de création d’un utilisateur | Aucune |
| 05 | Vérification du fonctionnement si on crée un utilisateur qui existe déjà | Données d’un utilisateur déjà créé | Message d’erreur du serveur « L’email ou le nom d'utilisateur existe déjà » | Nous essayons de créer un utilisateur qui existe déjà comme nouvel utilisateur | Aucune |
| 06 | Vérification du fonctionnement lorsque nous voulons nous connecter sans remplir le formulaire | Formulaire vide | Message d’erreur du serveur « Email ou mot de passe incorrect » | Nous cherchons à nous connecter avec un utilisateur vide | Aucune |
| 07 | Vérification du fonctionnement lorsque nous voulons nous connecter avec des données d’un utilisateur qui n’existe pas | Données invalides | Message d’erreur du serveur « Email ou mot de passe incorrect » | Nous entrons des données invalides sur la page login | Aucune |
| 08 | Vérification du fonctionnement en se connectant avec un utilisateur existant | Donnees valides | Message de validation du serveur « Connexion Réussie » | Nous nous connectons en ajoutant les donnees d’un utilisateur existant dans la base de données | Aucune |

#### 2.1. 3. Dépendances

Framework : Jest

Librairies : Express, Supertest, Mongoose, jsonwebtoken

Base de données : MongoDB

# 3.Tests d’intégration

Pour ce projet, nous avons fait en sorte que pour quiconque souhaitant avoir accès à ses fonctionnalités soit tout d’abord en disposition d’un token crée après création d’un compte. Nos tests se chargeront de vérifier le bon fonctionnement des ends points CRUD pour les véhicules avec authentification obligatoire et des routes

### 3.1. Pour chaque test d’intégration :

#### 3.1.1. Identification

* Tests des routes
* Test du Contrôleur des véhicules

#### 3.1.2. Description

* Tests des routes

Il s’agit ici de mettre en exergue les différentes voies de communication des modules pour tester leur fonctionnement :

* Tests du contrôleur des routes

Nous testerons les modules de suivant ayant déjà simulé une authentification :

* + Liste des véhicules
  + Création d’un véhicule
  + Recherche d’un véhicule par ID
  + Recherche d’un véhicule par numéro d’enregistrement
  + Recherche d’un véhicule par catégorie de prix

#### 3.1.3. Dépendances

Framework : Jest

Librairies : Express, Supertest, Mongoose, jsonwebtoken

Base de données : MongoDB

#### 3.1.5. Procédure de test

* Tests des routes
  + Accès sans Token

Ici nous vérifions que l’accès est bien refusé au module lorsqu’aucun Token n’est fourni. Par l’intermédiaire de la requête Get vers la route /Protected et le résultat que nous attendions est évidemment une erreur. Si c’est le cas alors notre test passera.

* + Accès avec Token invalide

Nous vérifions que le passage est bloqué lorsque le token généré lors de l’authentification n’est pas valide. Si la route /Protected se rend compte que le token reçu n’est pas valide alors le serveur doit envoyer une erreur. Si nous recevons une erreur alors le test passera.

* + Accès avec un Utilisateur introuvable

Ici notre test portera sur le fait qu’un Token généré est valide mais l’utilisateur a été supprimé. Une Requête GET avec un token valide, on aura un Mock de jwt. verify pour retourner un payload valide et un Mock d’User.findById pour retourner null et ainsi le serveur nous renverra un message d’erreur. Si le résultat obtenu est un message d’erreur du serveur alors le test passera.

* + Accès avec un Token valide et un utilisateur existant

Vérifier que les modules sont bien disponibles lors que nous avons un Token valide avec un utilisateur existant. Une Requête GET avec un token valide en utilisant un Mock de jwt. verify pour retourner un payload valide et un Mock d’User.findById pour retourner un utilisateur fictif, le résultat attendu est un message d’accès autorisé par le serveur. Si le résultat obtenu est le même alors le test passera

* Test du contrôleur de Véhicules

Les tests que nous effectuerons à ce niveau sont avec un Token déjà valide issus de l’authentification

* + Creation des véhicules

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Description | Entrée | Sortie Attendue | Méthode de Test | Remarque |
| 01 | Vérifie si on peut créer un véhicule sans caractéristique (pas de marque, …) | Aucune entrée (vide) | Message d’erreur du serveur | Nous créons une instance de véhicule vide | Aucune |
| 02 | Vérifie si l’on peut créer avec certaines caractéristiques manquantes | Nom, marque, pas de modèle | Message d’erreur du serveur | Nous créons une instance de véhicule en omettant d’ajouter un paramètre | Aucune |
| 03 | Vérifie si l’on peut créer un véhicule avec des données valides | Nom, marque, ID | Message de validation du serveur et notification « Véhicule crée » | Nous créons une instance de véhicule avec tous les paramètres requis | Aucune |

* + Recherche de Véhicule par ID

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Description | Entrée | Sortie Attendue | Méthode de Test | Remarque |
| 01 | Vérifions le fonctionnement lorsqu’on n’ajoute pas d’ID | Aucune entrée | Message d’erreur « Véhicule non trouvé » | Nous simulons une recherche sans avoir au prealable ajouter l’iD à rechercher | Aucune |
| 02 | Vérifions le fonctionnement en ajoutant un Id non valide | Id invalide | Message d’erreur serveur « Véhicule non trouvé » | Nous simulons une recherche en y ajoutant un Id non valide | Aucune |
| 03 | Vérifions le comportement avec une ID valide | Id existant ou enregistré lors de la création du véhicule | Message valide du serveur « Véhicule trouvé » et affichage du véhicule | Nous simulons une recherche avec l’ID d’un véhicule existant | Aucune |

* + Mise à jour d’un véhicule

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Description | Entrée | Sortie Attendue | Méthode de Test | Remarque |
| 01 | Vérifions le fonctionnement lorsqu’on met à jour un véhicule inexistant | Données invalides | Message d’erreur « Véhicule non trouvé » | Nous simulons une mise a jour pour un vehicule dont l’ID est inexistant | Aucune |
| 02 | Vérifions le fonctionnement en voulant mettre à jour dont l’Id est non valide | Id invalide | Message d’erreur serveur « Véhicule non trouvé » | Nous simulons une mise a jour d’un vehicule dont l’id est inexistant | Aucune |
| 03 | Vérifions le comportement avec une ID valide | Id existant ou enregistré lors de la création du véhicule | Message valide du serveur « Vehicule mis à jour » et affichage du véhicule | Nous simulons une recherche avec l’ID d’un véhicule avant de le mettre à jour | Aucune |

* + Recherche par Immatriculation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Description | Entrée | Sortie Attendue | Méthode de Test | Remarque |
| 01 | Vérifions le fonctionnement lorsqu’on n’ajoute pas de numéro d’immatriculation | Aucune entrée | Message d’erreur | Nous simulons une recherche sans avoir au prealable ajouter le numéro d’immatriculation à rechercher | Non couvert |
| 02 | Vérifions le fonctionnement en ajoutant un numéro d’immatriculation non valide | numéro d’immatriculation invalide | Message d’erreur serveur « Aucun véhicule trouvé avec ce numéro d’enregistrement » | Nous simulons une recherche en y ajoutant un numéro d’immatriculation non valide | Non couvert |
| 03 | Vérifions le comportement avec un numéro d’immatriculation valide | numéro d’immatriculation existant ou enregistré lors de la création du véhicule | Message valide du serveur « Véhicule trouvé » et affichage du véhicule | Nous simulons une recherche avec un numéro d’immatriculation d’un véhicule existant | Non couvert |

* + Liste des véhicules filtres par prix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Description | Entrée | Sortie Attendue | Méthode de Test | Remarque |
| 01 | Vérifions le fonctionnement lorsqu’on veut rechercher des véhicules en entrant des valeurs négatives | Valeurs < 0 | Message d’erreur « Prix maximum invalide (doit être un nombre positif) » | Nous simulons une recherche de véhicule en inscrivant un prix négatif | Non couvert |
| 02 | Vérifions le fonctionnement lorsqu’on veut rechercher des véhicules en entrant des valeurs supérieures au prixMAx ou inférieur au prixMin | Valeur > PrixMax ou  Valeur < PrixMin | Message d’erreur serveur « Aucun véhicule trouvé dans la fourchette de prix » | Nous simulons une recherche de véhicule avec un prix supérieure au montant Max ou inférieur au montant Min | Non couvert |
| 03 | Vérifions le fonctionnement lorsqu’on veut rechercher des véhicules en entrant des elements differents des chiffres | Valeurs invalides | Message valide du serveur | Nous simulons une recherche avec des valeurs qui ne sont pas des chiffres | Non couvert |

* + Suppression d’un véhicule

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Description | Entrée | Sortie Attendue | Méthode de Test | Remarque |
| 01 | Vérifions le fonctionnement lorsqu’on veut supprimer un véhicule inexistant | Données invalides | Message d’erreur « Véhicule non trouvé » | Nous simulons une recherche d’un véhicule absent de notre base de données dans le but de le supprimer | Aucune |
| 02 | Vérifions le fonctionnement en voulant supprimer un véhicule dont l’Id est non valide | Id invalide | Message d’erreur serveur « Véhicule non trouvé » | Nous simulons une recherche de véhicule avec un ID invalide dans l4optique de le supprimer | Aucune |
| 03 | Vérifions le comportement avec une ID valide | Id existant ou enregistré lors de la création du véhicule | Message valide du serveur « Véhicule supprimee » | Nous simulons une recherche avec l’ID d’un véhicule avant de le mettre a jour | Aucune |

# 4.Tests d’interface

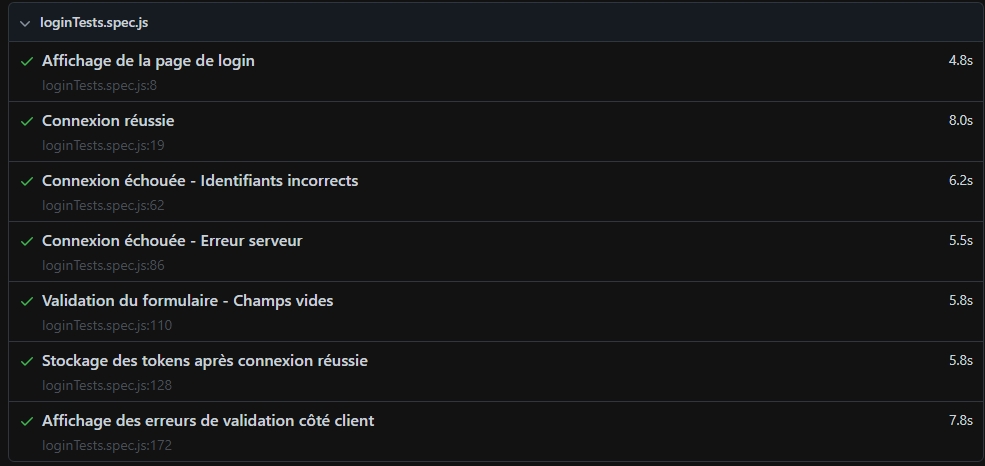
### 4.1. Pour le test de chaque interface :

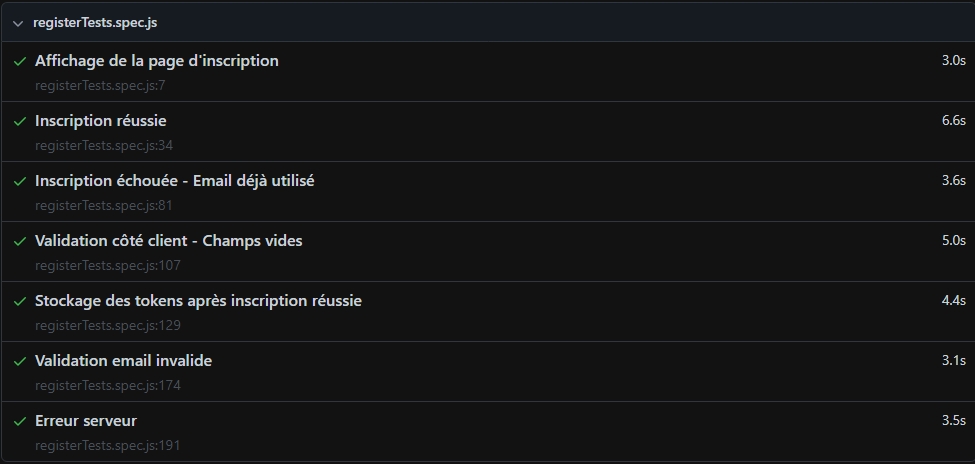
#### 4.1.1. Identification

Ces tests sont sur la partie FrontEnd de notre application, cela englobe la gestion et le fonctionnement de nos interfaces a l’instar de :

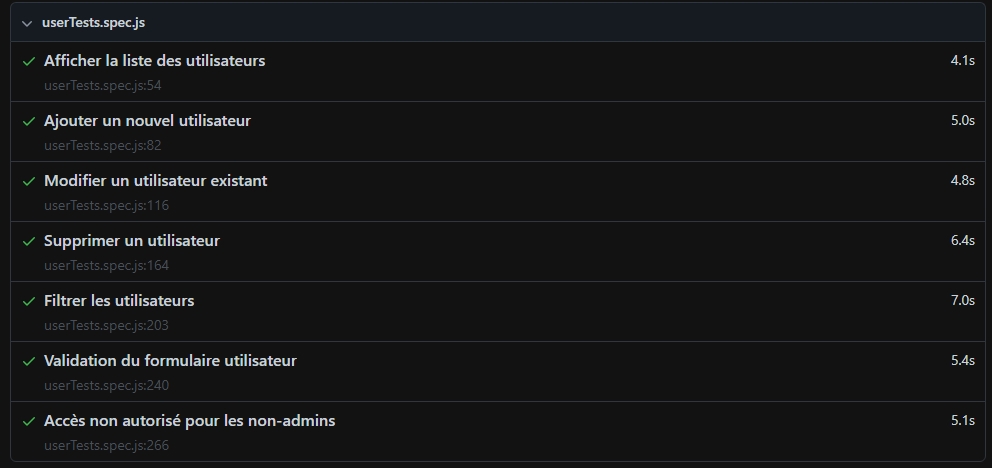
* Login
* Register
* User manipulation
* Vehicle Controller

#### 4.1.2. Procédure de Test

* Login
* Register



* User

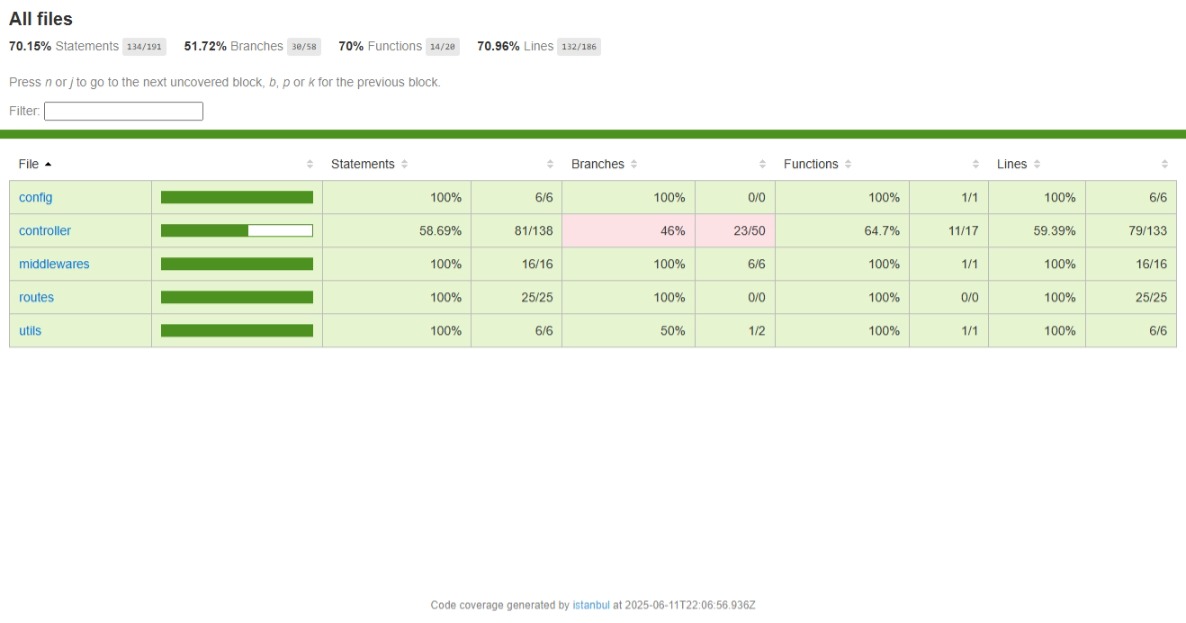


* Vehicle

#### 

#### 4.1.5. Description

# 5.Vérification de la documentation

A ce niveau nous présentons les différentes couvertures que nos tests englobent

