
Qualité logicielle

“Stock Car”



5A MAJ - SWE Groupe 2

Membres du groupe :

- MARRICHA Zineb
- ZIACH Doha

Encadré par:

M. PEREZ Emmanuel

Année universitaire : 2023 - 2024

Table des matières

Livrables du projet	3
Descriptif du contexte d'utilisation	4
Introduction	4
Diagramme de cas d'utilisation	4
Objectifs et scénarios	4
Dictionnaire de données	5
Environnement Technique	5
Dépendances	6
Interfaces Utilisateur	6
Plan de test	9
Tests réalisés	11
Tests unitaires	11
Tests d'intégration	12
Tests frontaux	14
Anomalies	17
Bilan de test	18
Résumé des tests réalisés	18
Lancement des tests	18
Résultats des tests	20
Couverture des tests	20
Anomalies et problèmes identifiés	20
Évaluation de la qualité	21
Recommandations	21
Conclusion	21

Livrables du projet

Livrable	État
Descriptif du contexte d'utilisation	Terminé ▾
Plan de test	Terminé ▾
Tests unitaires	Terminé ▾
Tests d'intégration	Terminé ▾
Cas des test Système	Terminé ▾
Anomalies	Terminé ▾
Rapports d'exécution	Terminé ▾
Bilan de test	Terminé ▾
Scripts de tests automatisés	Terminé ▾

Descriptif du contexte d'utilisation

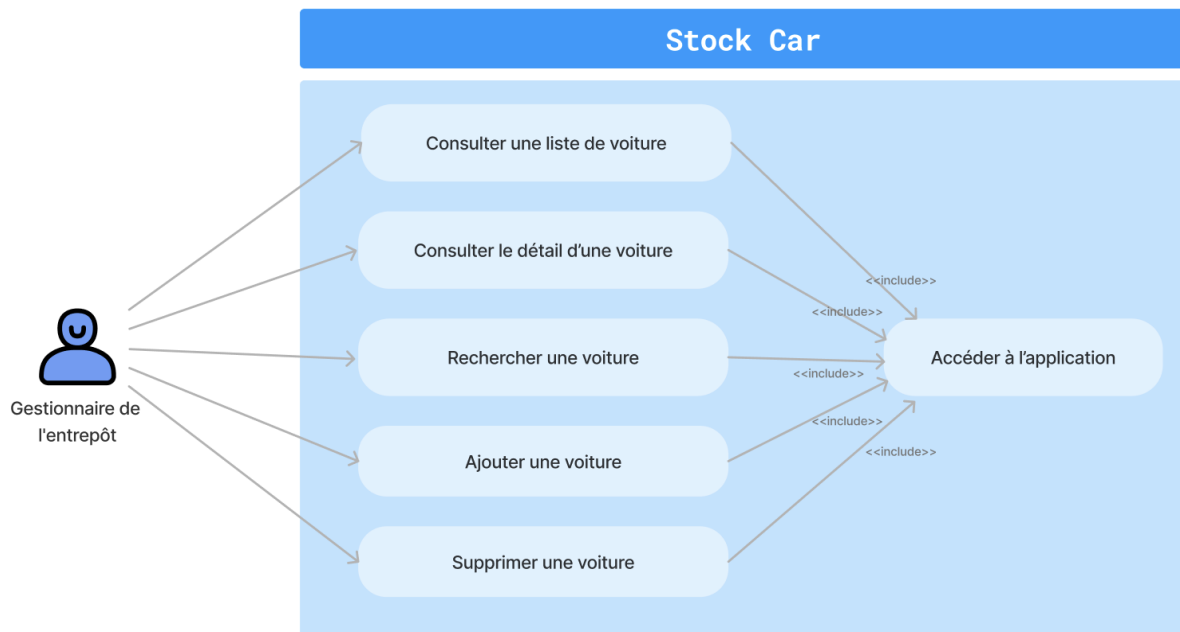
Introduction

Stock Car est une application web qui offre une variété de fonctionnalités permettant de gérer un entrepôt de voitures.

L'objectif de cette partie est d'avoir une compréhension approfondie de la manière dont l'application est utilisée.

Diagramme de cas d'utilisation

Ce diagramme de cas d'utilisation est une représentation visuelle du comportement fonctionnel de l'application :



Objectifs et scénarios

L'objectif de l'application est de faciliter la gestion des véhicules stockés dans l'entrepôt en offrant différentes fonctionnalités à l'utilisateur. Dans ce contexte, notre utilisateur cible est le gestionnaire de l'entrepôt. Une fois accéder à l'application, il peut :

- consulter la liste de voitures stockées dans l'entrepôt.
- consulter les caractéristiques d'une voiture spécifique en affichant les informations suivantes : ID, marque, modèle, finition, carburant, kilométrage, année et prix.
- rechercher une voiture en utilisant des mots-clefs ou des parties de mots.
- ajouter une nouvelle voiture en saisissant ses informations.
- et supprimer une voiture de la liste si besoin.

Dictionnaire de données

Les données stockées dans notre base de données contiennent des informations sur les voitures :

Nom de la table	Nom du champs	Type de données	Contraintes	Définition du champs
voiture	id	Integer	Clé primaire auto-incrémentée	L'identifiant unique pour chaque enregistrement de voiture
	marque	Varchar(30)	Non nulle	La marque du véhicule
	modele	Varchar(30)	Non nulle	Le modèle du véhicule
	finition	Varchar(30)	Non nulle	La finition du véhicule
	carburant	Char		Le type de carburant utilisé par le véhicule
	km	Integer		Le kilométrage du véhicule
	annee	Integer		L'année de fabrication du véhicule
	prix	Integer		Le prix de vente du véhicule

Environnement Technique

Le développement et le fonctionnement de l'application web "Stock Car" reposent sur les technologies suivantes :

Front-end :

- HTML : utilisé pour structurer la page web et son contenu
- CSS : utilisé pour mettre en forme les différents contenus définis par le HTML
- JS : utilisé pour ajouter des fonctionnalités interactives et dynamiques

Back-end :

- JAVA : un langage de programmation utilisé pour créer la logique métier de l'application web et pour interagir avec la base de données.
- MySql : un système de gestion de bases de données relationnelles qui permet de stocker et gérer les informations de notre stock de voitures.

Tests unitaires :

- Junit : un framework utilisé pour effectuer nos tests unitaires en langage de programmation Java.

Serveur d'application :

- Tomcat : un serveur d'applications open source utilisé pour déployer l'application.

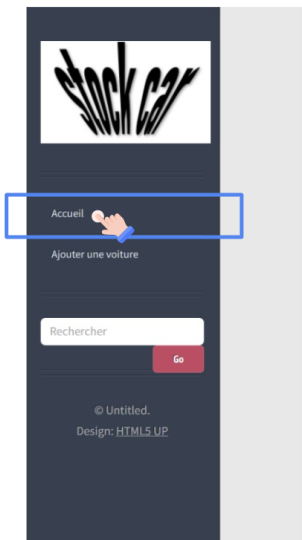
Dépendances

L'application Stock Car fonctionne de manière autonome, c'est-à-dire qu'elle peut s'exécuter sans dépendre d'autres systèmes ou applications externes pour son fonctionnement quotidien.

Interfaces Utilisateur

L'interface utilisateur de l'application Stock Car présente une barre de navigation composée de deux pages principales et une barre de recherche :

- La page "Accueil" qui permet aux utilisateurs de consulter la liste des voitures stockées dans l'entrepôt :




Liste des voitures en stock

Cette page recense les voitures disponibles dans la base

Marque	Modèle	Finition	Carburant	Kilométrage	Année	Prix	
Citroën	C4 Picasso	Feel	DIESEL	78000	2017	15500 €	Détails
Peugeot	3008	Allure	DIESEL	4	2020	38000 €	Détails
Renault	Mégane	Dynamique	ESSENCE	133000	2007	3100 €	Détails
Opel	Corsa	Elegance	ESSENCE	140	0	0 €	Détails
Audi	R8	Black edition	ESSENCE	68000	2009	50000 €	Détails

Previous Page 1 2 Next Page

- La page "Ajouter une voiture" qui permet aux utilisateurs d'ajouter de nouvelles voitures au stock :



Accueil

Ajouter une voiture

Rechercher

Go

© Untitled.
Design: HTML5 UP

Liste des voitures en stock

Cette page recense les voitures disponibles dans la base

Marque :

Modèle :

Finition :

Carburant :

--Veuillez choisir un carburant--

Kilométrage :

Année :

Prix :

Ajouter

- La barre de recherche permet aux utilisateurs d'effectuer des recherches ciblées et rapides, ce qui simplifie la localisation de voitures dans l'entrepôt :



Accueil

Ajouter une voiture

peugeot

Go

© Untitled.
Design: HTML5 UP


Liste des voitures en stock

Cette page recense les voitures disponibles dans la base

Marque	Modèle	Finition	Carburant	Kilométrage	Année	Prix	
Peugeot	3008	Allure	DIESEL	4	2020	38000 €	Détails

Previous Page 1 Next Page

- Le lien "Détails" permet à l'utilisateur d'accéder aux informations spécifiques de la voiture et de la supprimer si nécessaire :



[Accueil](#)

[Ajouter une voiture](#)

[Go](#)

© Untitled.
Design: HTML5 UP

Liste des voitures en stock

Cette page recense les voitures disponibles dans la base

Marque	Modèle	Finition	Carburant	Kilométrage	Année	Prix	
Peugeot	3008	Allure	DIESEL	4	2020	38000 €	Détails

Marque : Peugeot
Modèle : 3008
Finition : Allure
Carburant : Autre
Kilométrage : 4
Année : 2020
Prix : 38000

[Supprimer](#)

Plan de test

Ce plan de test couvrira une série de tests fonctionnels, visant à évaluer les fonctionnalités principales de l'application Stock Car.

- Dans la portée :

ID du test	Fonctionnalité	Description	Étapes du test	Résultat attendu
1	Consultation de la liste des voitures	Vérifier si la liste des voitures est affichée correctement	1. Ouvrir l'application; 2. Une fois sur la page d'accueil, vérifier que la liste contient des véhicules.	- Le tableau contenant les voitures est correctement affichée - Au moins un véhicule est affiché
2	Consultation des détails d'une voiture	Vérifier que les caractéristiques et informations de la voiture spécifique sont affichés correctement	1. Ouvrir l'application; 2. Une fois sur la page d'accueil choisissez une voiture et cliquer sur le lien "Détails"; 3. Vérifiez que les informations ID, marque, modèle, finition, carburant, kilométrage, année et prix sont correctement affichées.	- Les caractéristiques de la voiture sont affichées. - Les informations sont correctes et adéquates avec le nom de la colonne.
3	Recherche d'une voiture	Vérifier que la barre de recherche fonctionne correctement	1. Ouvrir l'application; 2. Utilisez la barre de recherche en saisissant des mots-clés ou des parties de mots (par exemple, "Renaut") dans la barre de recherche; 3. Vérifiez que les voitures correspondantes sont correctement affichées.	- Les voitures correspondantes sont affichées.
4	Ajout d'une voiture	Ajouter une nouvelle voiture en saisissant ses informations	1. Ouvrir l'application; 2. Accéder à la page "Ajouter une voiture"; 3. Remplir le formulaire avec des informations valides; 4. Cliquez sur le bouton "Ajouter".	- La nouvelle voiture est ajoutée à la liste des voitures avec succès.
5	Suppression d'une voiture	Supprimer une voiture de la liste	1. Ouvrir l'application; 2. Une fois sur la page d'accueil choisissez une voiture et cliquer sur le lien "Détails"; 3. Un bouton "Supprimer" s'affiche; 4. Cliquez sur le bouton "Supprimer".	- Le bouton "Supprimer" s'affiche avec succès; - En cliquant sur le bouton, la voiture est supprimée et n'apparaît donc plus dans la liste.

6	Gestion des erreurs	Vérifier comment l'application gère les erreurs	1. Ouvrir l'application; 2. Essayez d'ajouter une voiture sans remplir certains champs obligatoires.	<ul style="list-style-type: none"> - Un message d'erreur s'affiche - La voiture ne s'ajoute pas à la liste
			1. Ouvrir l'application; 2. Essayez d'ajouter une voiture en envoyant un formulaire vide.	
			1. Ouvrir l'application; 2. Essayez d'ajouter une voiture en envoyant un formulaire avec des informations invalides (par exemple en saisissant des caractères au lieu de valeurs numériques dans le champ de l'année)	
7	Contrôle de pagination	Tester la pagination du tableau	1. Ouvrir l'application; 2. Vérifiez si la pagination est activée lorsque la liste des voitures est longue; 3. Essayez de passer à la page suivante en cliquant sur le bouton "Next Page"; 4. Essayez de revenir en arrière en cliquant sur le bouton "Previous Page".	<ul style="list-style-type: none"> - Les boutons "Next Page" et "Previous Page" sont visibles pour la navigation; - En cliquant sur le bouton "Next Page", les éléments de la page suivante sont affichés; - En cliquant sur le bouton "Previous Page", les éléments de la page précédente sont affichés à nouveau;
8	Contrôle de charge	Vérifier la réactivité de l'application face à une charge élevée	1. Ouvrir une application de test d'API comme postman; 2. Effectuer une requête POST en utilisant l'endpoint /voiture/add 3. Simuler une charge élevée en effectuant un grand nombre de requêtes pour ajouter des voitures à la liste.	Les temps de réponse devraient rester dans une plage acceptable

- Hors de portée :

Ces fonctionnalités ne sont pas testées car elles ne sont pas demandées dans le contexte de ce projet :

- Sécurité du site Web

Tests réalisés

L'application Stock Car est couverte par différents niveaux de test :

- Les tests unitaires pour vérifier la validité d'unités individuelles de l'application, indépendamment les unes des autres.
- Les tests d'intégration pour garantir que les différentes unités de code peuvent collaborer ensemble.
- Les tests de système pour valider le fonctionnement global de l'application.

Tests unitaires

ID	Fonction	Description	Type de test	Cas de test	Résultat attendu	Statut
1	StringUtils. estEntier()	Vérifier que la fonction renvoie vrai pour les entiers	unitaire	1. Vérifier la fonction avec différentes valeurs d'entrée : Chaînes de test = "4", "40", "-50"	La fonction renvoie "true"	Réussi
				2. Vérifier la fonction avec une seule valeur d'entrée de type entier : Chaîne de test : "10"	La fonction renvoie "true"	Réussi
				3. Vérifier la fonction avec une valeur d'entrée de type caractère : Chaîne de test : "-"	La fonction renvoie "false"	Réussi
2	StringUtils.nbOccurrences()	Vérifier que la fonction renvoie le nombre d'occurrences d'un caractère dans une chaîne	unitaire	1. Vérifier la fonction avec la chaîne de test : "aabbdkjnfreljne", et compter le nombre d'occurrence du caractère "e"	La fonction renvoie 2	Réussi
				2. Vérifier la fonction avec la chaîne de test : "aabbdkjnfreljne", et compter le nombre d'occurrence du caractère "z"	La fonction renvoie 0	Réussi
3	Voiture.check()	Vérifier les champs d'une voiture pour déterminer si la voiture est cohérente, pour être insérée en base	unitaire	1. Données de test correctes	La fonction renvoie "true"	Réussi
				2. Prix négatif	La fonction renvoie "false"	Réussi
4	Voiture.getTypeDonnee()	Vérifier le type de données renvoyés	unitaire	1. Tester les propriétés : "marque", "modele", "finition"	La fonction renvoie "string" pour chaque propriété	Réussi
				2. tester les propriétés : "id", "annee", "km", "prix"	La fonction renvoie "entier" pour chaque propriété	Réussi
				3. tester avec une entrée nulle	La fonction renvoie la chaîne vide ""	Réussi
				4. tester avec une autre entrée	La fonction renvoie la chaîne vide ""	Réussi

5	voitureAPI.voitureFromJson()	Tester la conversion d'un JSON en objet Voiture	unitaire	<ul style="list-style-type: none"> - Créer un JSON représentant une voiture - Convertir le JSON en un objet Voiture 	Obtenir un objet Voiture valide avec les détails correspondants aux données du JSON	Réussi
6	voitureDAO.construireRequeteMasque()	Vérifier si la méthode construireRequeteMasque produit la requête SQL attendue en fonction de la saisie fournie	unitaire	Appeler la méthode construireRequeteMasque avec une saisie spécifique de type String pour construire une requête.	La requête construite contient des clauses LIKE pour les colonnes "marque", "modele", " finition" et "carburant", séparées par des OR.	Réussi

Tests d'intégration

ID	Fonctionnalité	Scénario de test	Conditions de test	Type de test	Étapes du test	Résultat attendu	Statut
Voiture DAO Integration Test							
7	Ajout d'une voiture	Ajouter une nouvelle voiture à la base de donnée	<ul style="list-style-type: none"> - Base de données accessible - Paramètres de connexion valides 	d'intégration	<ul style="list-style-type: none"> - Créer une voiture avec les différents attributs - Appeler la méthode "ajouterVoiture" avec la voiture créée - Faire un appel vers la base de données pour récupérer l'objet créé 	Le résultat de l'appel n'est pas vide : la voiture créée existe bien dans la base de données	Réussi
8	Modification d'une voiture	Vérifier la modification des détails d'une voiture dans la base de données	<ul style="list-style-type: none"> - Base de données accessible - Paramètres de connexion valides - Mise à jour des détails d'une voiture existante 	d'intégration	<ul style="list-style-type: none"> - Choisir l'ID de voiture à modifier - Modifier les détails de la voiture - Vérifier la mise à jour dans la base de données 	Les valeurs modifiées correspondent à celles attendues	Réussi
9	Suppression d'une voiture	Supprimer une voiture de la base de données	<ul style="list-style-type: none"> - Base de données accessible - Paramètres de connexion valides - ID valide passé en paramètre pour la suppression 	d'intégration	<ul style="list-style-type: none"> - Créer une voiture avec les différents attributs - L'ajouter dans la base de données en utilisant la méthode "ajouterVoiture" - La supprimer en utilisant la méthode "supprimerVoiture" - Faire un appel vers la base de données pour récupérer l'objet supprimé 	Le résultat de l'appel est vide : la voiture supprimée n'est plus présente dans la base de données	Réussi

10	Récupération des voitures	Récupérer les voitures avec des critères spécifiques	<ul style="list-style-type: none"> - Base de données accessible - Paramètres de connexion valides - Voitures ajoutées à la base de données avec des critères spécifiques 	d'intégration -	<ul style="list-style-type: none"> - Création de deux objets Voiture avec des attributs spécifiques. - Ajout de ces voitures à la base de données via voitureDAO. - Création d'une HashMap de critères de recherche - Appel de getVoitures avec ces critères. - Vérification des résultats. 	La méthode doit retourner les voitures correspondant aux critères spécifiés et respecter le nombre spécifié de voitures attendues.	Réussi -
Voiture API/JSON Integration Test							
11	Récupération de toutes les voitures via l'API	Récupérer toutes les voitures	<ul style="list-style-type: none"> - Serveur REST disponible - Endpoint "/voiture/get/all" accessible 	d'intégration -	Effectuer une requête GET pour récupérer toutes les voitures	Réponse HTTP 200 et une liste de voitures	Réussi -
12	Récupérer toutes les voitures au format JSON	Récupérer la liste de toutes les voitures au format JSON	<ul style="list-style-type: none"> - Serveur opérationnel - Endpoint "/voiture/get/all" disponible 	d'intégration -	Appeler l'endpoint /voiture/get/all pour récupérer les voitures au format JSON	Une liste JSON contenant toutes les voitures	En cours -
13	Récupération d'une voiture par ID via l'API	Récupérer une voiture spécifique par ID via l'API	<ul style="list-style-type: none"> - Serveur REST disponible - Endpoint "/voiture/get/{id}" accessible - ID existant 	d'intégration -	Effectuer une requête GET pour récupérer une voiture spécifique par ID	<ul style="list-style-type: none"> - Réponse HTTP 200 - Les détails de la voiture dont l'ID a été spécifié sont affichés 	Réussi -
14	Récupération d'une voiture par ID au format JSON	Vérifier la récupération d'une voiture spécifique	<ul style="list-style-type: none"> - Serveur opérationnel - Endpoint "/voiture/get/{id}" disponible - ID existant 	d'intégration -	Appeler l'endpoint /voiture/get/{id}	Les détails de la voiture spécifique sont affichés au format JSON	En cours -
15	Ajout d'une voiture via l'API	Ajouter une nouvelle voiture via l'API	<ul style="list-style-type: none"> - Serveur REST disponible, - Endpoint "/voiture/add" accessible 	d'intégration -	Effectuer une requête POST avec les détails d'une nouvelle voiture	<ul style="list-style-type: none"> - Réponse HTTP 200 et confirmation de succès (true) - La voiture est ajoutée dans la base de données 	Réussi -
16	Ajout d'une voiture en format JSON	Tester l'ajout d'une nouvelle voiture en format JSON	<ul style="list-style-type: none"> - Paramètres valides pour créer une voiture - Endpoint "/voiture/add" accessible 	d'intégration -	<ul style="list-style-type: none"> - Créer un objet JSON représentant une nouvelle voiture, - Appeler l'endpoint /voiture/add avec les données JSON de la voiture 	Obtenir une réponse 200 confirmant l'ajout réussi de la voiture	Réussi -

17	Suppression d'une voiture via l'API	Supprimer une voiture existante via l'API	- Serveur REST disponible, - Endpoint "/voiture/delete" accessible - ID valide pour la voiture à supprimer	d'intégration	Effectuer une requête POST avec l'ID d'une voiture à supprimer	- Réponse HTTP 200 et confirmation de succès (true) - La voiture est supprimée de la base de données	Réussi
18	Suppression d'une voiture au format JSON	Supprimer une voiture existante au format JSON	- Endpoint "/voiture/delete" accessible - ID valide pour la voiture à supprimer	d'intégration	- Créer un JSON pour représenter la voiture à supprimer - Appeler l'endpoint /voiture/delete avec les données JSON de la voiture à supprimer	- Obtenir une réponse 200 confirmant la suppression réussie de la voiture au format JSON - La voiture est supprimée de la base de données	En cours

Tests frontaux

Pour évaluer la fonctionnalité et la fiabilité de l'interface utilisateur, des tests frontaux ont été menés à l'aide de l'outil Cypress (tests automatisés).

Cypress fournit une gamme complète d'outils pour automatiser les tests frontaux, permettant une vérification approfondie de la navigation, des interactions utilisateur et de la cohérence visuelle de l'application.

Nous avons réalisés trois scénarios de tests :

1- Test de Chargement Initial des Voitures (sur une voiture spécifique):

Ce test vise à s'assurer que la page chargée affiche correctement les informations d'une voiture spécifique dans une table et que les interactions utilisateur (comme le clic sur un lien pour plus de détails) se déroulent comme prévu.

Les étapes du test sont les suivants :

- Mise en place d'interceptions pour les requêtes GET et POST vers l'API.
- Visite de la page web de l'application.
- Vérification de la présence d'un élément dans le DOM identifié par l'ID "listeVoitureTable". Cela garantit que la structure HTML contient cet élément attendu.
- Attente de la réponse de l'API pour la liste des voitures.
- Vérification de la table : vérifier si elle contient une seule ligne avec les détails de la voiture, tels que la marque, le modèle, le carburant, etc.
- Interaction avec un lien dans la première ligne de la table (le lien "Détails")

The screenshot shows the Cypress test runner on the left and a web browser on the right. The browser displays the page "Liste des voitures en stock" with a single car entry: Toyota Camry, Sedan, ESSENCE, 50000 km, 2020, 25000 €. The test runner shows a spec file "details-test cy.js" with a test "should display the list of cars on load". The test body includes a visit to the URL, a get for the table, an assertion to check the table exists, a fetch to the API, a wait for the API call, and a get for the first tbody row.

2- Test de Chargement Initial des Voitures (sur plusieurs voitures) :

Ce test vise à vérifier si la liste des voitures est correctement affichée lors du chargement initial de l'application "Stock Car App"

Les étapes du test sont les suivants :

- Interception de la requête de type GET vers l'URL qui renvoie les détails des voitures.
- Visite de la page web de l'application.
- Vérification de l'existence de l'élément avec l'ID "listeVoitureTable" pour garantir que la table est présente dans le DOM, garantissant ainsi que les détails des voitures sont bien affichés à l'utilisateur.
- Le test attend la réponse de l'API qui fournit les détails des voitures avant de terminer.

Aucune assertion spécifique sur les données des voitures n'est effectuée dans ce test.

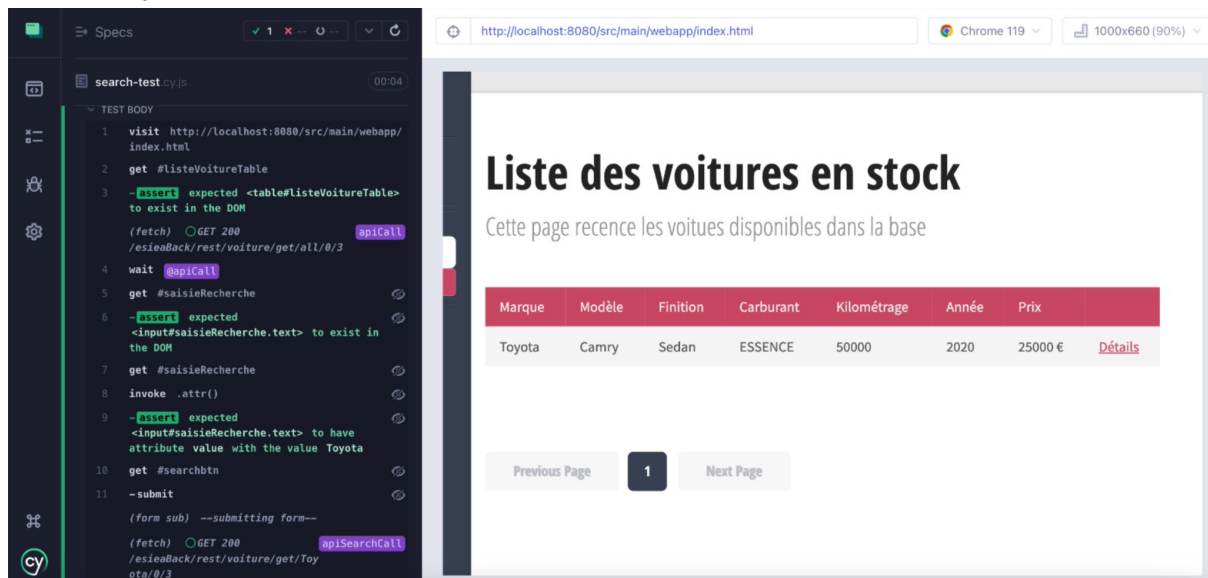
The screenshot shows the Cypress test runner on the left and a web browser on the right. The browser displays the page "Liste des voitures en stock" with a list of three cars: Toyota Camry, Honda Civic, and Ford Fusion. The test runner shows a spec file "get-all-test cy.js" with a test "should display the list of cars on load". The test body includes a visit to the URL, a get for the table, an assertion to check the table exists, a fetch to the API, a wait for the API call, and a get for the first tbody row.

3- Recherche d'une voiture spécifique :

Ce test valide la fonctionnalité de recherche de voiture en utilisant un critère spécifique (ici la marque "Toyota").

Les étapes du test sont les suivantes :

- Mise en place d'une interception pour la requête GET vers l'API de recherche de voitures par marque.
- Visite de la page web de l'application.
- Vérification de l'existence de l'élément avec l'ID "saisieRecherche".
- Définition de la valeur de recherche comme "Toyota" dans l'élément de saisie.
- Soumission de la recherche.
- Attente de la réponse de l'API pour la recherche basée sur la marque "Toyota".
- Vérification que la table affiche les détails de la voiture correspondant à la marque "Toyota".



The screenshot displays a Cypress test runner on the left and a web application on the right. The test runner shows a script for searching for Toyota cars. The web application shows the 'Liste des voitures en stock' page with a table of cars.

Test Script (Cypress):

```

1 visit http://localhost:8080/src/main/webapp/index.html
2 get '#listeVoitureTable'
3 -assert expected <table#listeVoitureTable> to exist in the DOM
(fetch) GET 200 /esieaBack/rest/voiture/get/all/0/3 apiCall
4 wait apiCall
5 get '#saisieRecherche'
6 -assert expected <input#saisieRecherche.text> to exist in the DOM
7 get '#saisieRecherche'
8 invoke .attr()
9 -assert expected <input#saisieRecherche.text> to have attribute value with the value Toyota
10 get '#searchbtn'
11 -submit
(form sub) --submitting form--
(fetch) GET 200 /esieaBack/rest/voiture/get/Toyota/0/3 apiSearchCall

```

Web Application (Liste des voitures en stock):

Cette page recense les voitures disponibles dans la base

Marque	Modèle	Finition	Carburant	Kilométrage	Année	Prix	
Toyota	Camry	Sedan	ESSENCE	50000	2020	25000 €	Détails

Previous Page 1 Next Page

Les résultats des tests que nous avons effectués n'ont pas révélé d'anomalies. Cependant, cela ne signifie pas nécessairement qu'il n'y en a pas. En procédant à une simulation de l'expérience utilisateur, nous avons néanmoins rencontré plusieurs dysfonctionnements que nous avons consignés dans le chapitre suivant de notre rapport. Malheureusement, faute de temps, nous n'avons pas pu réaliser tous les tests prévus pour ce rapport.

Anomalies

Ce rapport résume les anomalies rencontrées dans l'application, leur impact potentiel et leur priorité pour la correction. Pour faciliter la résolution, chaque anomalie est décrite avec autant de détails que possible.

Rapport d'anomalies :

Anomalie	Description	Impact	Priorité	Statut
Bouton Supprimer	Le bouton de suppression de voiture reste affiché même après un changement de page.	Réduction de l'expérience utilisateur en raison d'une fonctionnalité erronée	Elevée ▾	En attente de correction
Fonction onClick non opérationnelle	Certaines fonctionnalités ou actions associées aux événements onClick ne se déclenchent pas correctement, affectant potentiellement l'interactivité de l'application.	Réduction de l'interactivité utilisateur	Moyenne ▾	En attente de correction
Barre de recherche limitée à la page d'accueil	La fonctionnalité de recherche semble être restreinte à la page d'accueil uniquement, limitant son utilisation sur d'autres pages de l'application.	Limite l'utilité de la fonction de recherche pour les autres pages Réduction de l'efficacité de la recherche	Moyenne ▾	En attente de clarification ou de correction
Bouton "Next Page" désactivé mais toujours cliquable	Le bouton "Next Page" semble désactivé mais reste cliquable, même s'il ne change pas de page.	Confusion pour l'utilisateur, comportement inattendu.	Basse ▾	En attente de correction
Ajout d'une Voiture avec Inputs Vides	Lors de l'ajout d'une voiture avec des champs vides, un message d'erreur devrait être affiché pour informer l'utilisateur que les champs obligatoires doivent être renseignés et que l'ajout de la voiture dans la base de données n'a pas été effectué.	Confusion de l'utilisateur car aucun message d'erreur ou de confirmation n'est affiché, l'informant sur l'état de l'opération d'ajout dans la base de données.	Elevée ▾	En attente de correction

Bilan de test

Résumé des tests réalisés

Nous avons effectué une série de tests couvrant différents niveaux :

- tests unitaires
- tests d'intégration
- tests frontaux

Les cas de test inclus étaient principalement axés sur la consultation, l'ajout, la suppression et la recherche de voitures dans l'entrepôt de l'application "Stock Car", ainsi que la validation des données saisies.

Lancement des tests

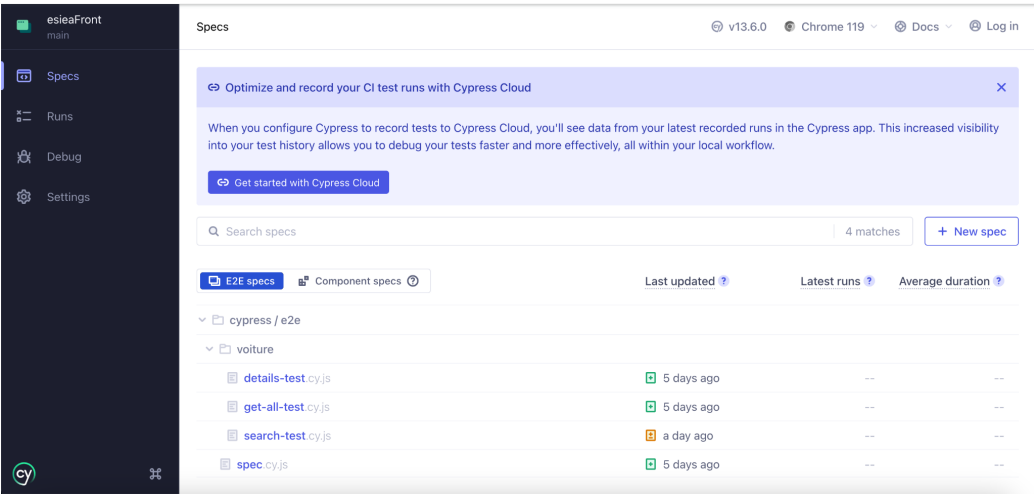
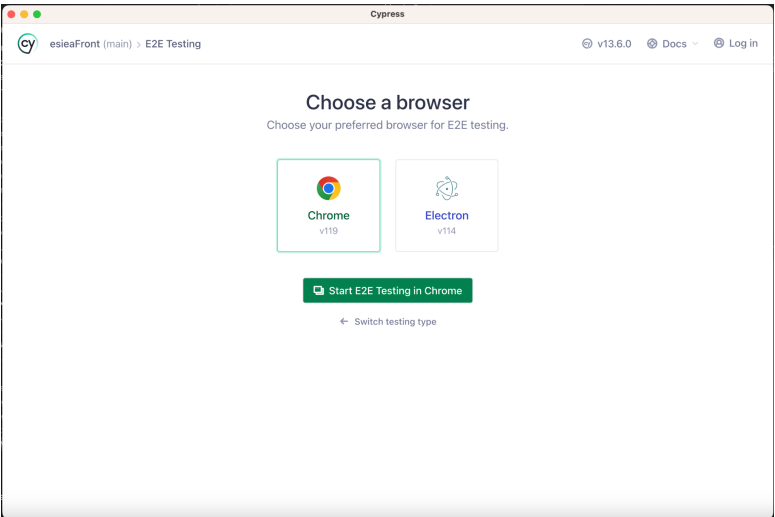
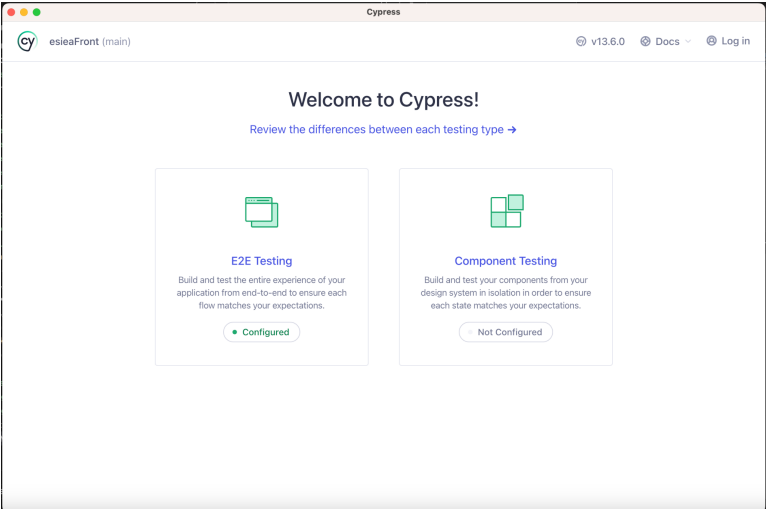
- Test front-end automatisés avec Cypress:
 - Télécharger/installer nodejs
 - **npm init -y** dans le dossier esieaFront
 - **npm install cypress --save-dev**
 - dans un terminal : **http-server**

```
(base) marrichazineb@MacBook-Pro-de-MARRICHA esieaFront % http-server
Starting up http-server, serving ./
http-server version: 14.1.1

http-server settings:
CORS: disabled
Cache: 3600 seconds
Connection Timeout: 120 seconds
Directory Listings: visible
AutoIndex: visible
Serve GZIP Files: false
Serve Brotli Files: false
Default File Extension: none

Available on:
http://127.0.0.1:8081
http://192.168.1.19:8081
Hit CTRL-C to stop the server
```

- Une fois l'installation terminée et le serveur lancé (par http-server sans passer par tomcat (on utilise nodejs)), exécuter la commande suivante (dans un autre terminal) pour ouvrir l'interface graphique de Cypress : **npm cypress open**
Cypress recherche les fichiers de test dans le dossier cypress/integration par défaut.



Il suffit de faire un double clic pour lancer l'un des trois tests.

- Test back-end :
 - Pas de procédure particulière pour lancer les tests du back-end, nous avons utilisé une base de donnée en mémoire H2 et un serveur d'application tomcat embarqué.

Résultats des tests

Tests unitaires :

- EstEntier() : Réussi pour différents scénarios.
- NbOccurrence() : Réussi pour les scénarios définis.
- Voiture.check() : Réussi pour les scénarios attendus.
- Voiture.getTypeDonnee() : Réussi pour les propriétés définies.
- VoitureAPI.voitureFromJson() : Réussi.
- voitureDAO.construireRequeteMasque() : Réussi.

Tests d'intégration :

- Voiture DAO Integration Test : Réussite de l'ajout, modification, suppression et récupération de voitures.
- Voiture API Integration Test : Réussite de la récupération, de l'ajout et de la suppression de voitures via l'API.
- Voiture JSON Integration Test : Réussite de la récupération, de l'ajout et de la suppression de voitures en format JSON.

Tests frontaux :

- Test de Chargement Initial des Voitures sur une voiture spécifique : Réussi.
- Test de Chargement Initial des Voitures sur plusieurs voitures : Réussi.
- Recherche d'une voiture spécifique : Réussi.

Couverture des tests

Bien que la couverture des tests ne soit pas complète à 100%, nous avons fait de notre mieux pour couvrir un maximum de scénarios, en accordant une attention particulière aux tests d'intégration et aux tests unitaires.

Anomalies et problèmes identifiés

Nous avons identifié plusieurs anomalies lors de nos tests :

- Bouton Supprimer persistant malgré la navigation
- L'événement "onClick" ne se déclenche pas correctement.
- Barre de recherche limitée à la page d'accueil.
- Problèmes avec le bouton "Next Page".
- Ajout de voiture avec des champs vides ne renvoie pas de message d'erreur.

Évaluation de la qualité

En général, l'application semble stable et fonctionnelle. Cependant, les anomalies identifiées ont un impact sur l'expérience utilisateur et nécessitent une intervention immédiate pour améliorer la qualité globale de l'application.

Recommandations

Nous recommandons de :

- Corriger les anomalies identifiées en priorité pour améliorer l'expérience utilisateur.
- Finaliser les tests pour une couverture complète.
- Effectuer une révision approfondie pour détecter d'autres éventuels problèmes.

Conclusion

Le bilan des tests actuels démontre que l'application est fonctionnelle et que des problèmes doivent être résolus. L'application "Stock Car" devrait offrir une meilleure expérience utilisateur et une meilleure qualité globale en résolvant ces problèmes.

