Rapport de testing

I. Recommandations

A. Tests à automatiser

Justification du choix des tests critiques

1. La connexion

Pourquoi ce test est-il critique ?

- **Point d'entrée obligatoire** : La connexion est le passage obligé pour accéder à toutes les fonctionnalités réservées aux utilisateurs (panier, commandes, historique, etc.).
- **Sécurité** : Elle protège l'accès aux données personnelles et aux actions sensibles (achats, gestion de compte).
- **Expérience utilisateur** : Un problème de connexion bloque l'utilisation du site, ce qui entraîne une perte immédiate de clients potentiels.
- **Fiabilité globale** : Si la connexion échoue ou est vulnérable, tout le reste du parcours utilisateur s'effondre.

En résumé : Tester la connexion garantit que seuls les utilisateurs autorisés peuvent accéder à leur espace, et que le site reste sécurisé et fonctionnel dès la première étape du parcours client.

---

2. Le panier

Pourquoi ce test est-il critique ?

- **Cœur du métier e-commerce** : Le panier est la fonctionnalité centrale d'une boutique en ligne. C'est là que l'utilisateur ajoute, modifie ou supprime des produits avant de passer commande.
- **Impact direct sur le chiffre d'affaires** : Un bug sur le panier (ajout impossible, quantités erronées, suppression non fonctionnelle…) empêche ou décourage l'achat, ce qui a un impact immédiat sur les ventes.
- **Expérience utilisateur** : Un panier fiable rassure l'utilisateur, permet de corriger ses choix, et fluidifie le passage à l'achat.
- **Effet domino** : Si le panier ne fonctionne pas, il est inutile de tester la commande, le paiement ou la livraison : tout le processus d'achat est bloqué.

En résumé : Tester le panier, c'est s'assurer que l'utilisateur peut réellement acheter sur le site, ce qui est la finalité de toute boutique en ligne.

Pourquoi ne pas avoir choisi les fiches produits ?

- **Les fiches produits sont importantes**, mais elles n'empêchent pas à elles seules l'utilisation du site : un affichage imparfait ou un détail manquant ne bloque pas forcément l'achat.
- **La connexion et le panier sont des prérequis absolus** : sans eux, aucun parcours client n'est possible, même si les fiches produits sont parfaites.

B. Préconisations pour la suite

1. Poursuivre l'automatisation sur les fonctionnalités restantes

Tests à envisager :

- **Fiches produits** : Vérifier l'affichage dynamique, la cohérence des informations (prix, stock, description), la gestion des images, etc.
- **Les parcours critiques pour le business** : Commande, paiement, gestion des stocks, notifications e-mail, etc.
- **Vérifications purement esthétiques** : Elles évoluent souvent et sont mieux testées manuellement ou via des outils de visual testing spécialisés.
- **Tests de régression** : Pour s'assurer qu'aucune fonctionnalité existante n'est cassée lors de l'ajout de nouvelles features ou de correction de bug.

---

2. Valeur ajoutée de l'automatisation

- **Gain de temps** : Les tests automatisés s'exécutent rapidement et peuvent être rejoués à chaque livraison, ce qui accélère les cycles de développement.
- **Détection précoce des bugs** : Les anomalies sont détectées dès l'intégration, limitant leur coût de correction.
- **Fiabilité** : L'automatisation permet de tester de nombreux scénarios, y compris des cas limites difficiles à couvrir manuellement.
- **Documentation vivante** : Les scripts de tests servent de référence sur le comportement attendu de l'application.

---

3. Conclusion

Prioriser les tests selon leur criticité métier et leur fréquence d'utilisation.
L'automatisation des tests restants est un investissement rentable sur le moyen et long terme. Elle permettra à l'équipe QA de se concentrer sur les tests exploratoires et les cas complexes, tout en assurant une couverture optimale des fonctionnalités clés.
Il est donc fortement recommandé d'automatiser progressivement tous les parcours utilisateurs majeurs, en adaptant la stratégie selon l'évolution du budget et des priorités métier.

II.Bilan de campagne de validation : tests automatisés

Document revu par :

Nom	Fonction	Version	Signature
JC Lefèvre	*Testeur*	*1.0.0*	

Contexte :

Eco Bliss Bath est une start-up de 20 collaborateurs, spécialisée dans la vente de produits de beauté écoresponsables. Son produit phare est un savon solide, conçu dans une démarche respectueuse de l'environnement.
Dans le cadre de son développement, l'entreprise prépare la mise en ligne de sa boutique e-commerce. Une première version du site a été réalisée par l'équipe de développement, puis testée manuellement par Marie, Testeuse QA.
À la demande du CTO, j'interviens en tant que **QA Engineer** afin de définir et de mettre en œuvre l'automatisation des tests. Cette campagne s'appuie sur le bilan des tests manuels déjà effectués, tout en tenant compte des contraintes budgétaires exposées par le Product Owner.
Les tests automatisés ont été réalisés sur une période d'**une semaine**, dans l'environnement suivant :
Système d'exploitation : Windows 11 Pro, version 23H2
Navigateurs :

- Firefox, version 131
- Chrome, version 130.0.6723.60

Outils de support aux tests :

- Visual Studio Code, version 1.93.0
- Docker, version 4.39.0
- Cypress, version 14.02.1

Objectifs de la campagne de tests

Garantir que la boutique en ligne respecte les spécifications fonctionnelles, fonctionne correctement et répond aux besoins des utilisateurs finaux. Cette campagne vise à valider la qualité de la première version du site avant sa mise en production, afin de prévenir les défauts, détecter et corriger les anomalies, et ainsi réduire les coûts tout en assurant la satisfaction client.

- **Évaluer la robustesse et la conformité de la première version du produit** par rapport aux exigences définies.
- **Prévenir l'apparition de défauts en production** en identifiant et corrigeant les anomalies en amont.
- **Réduire les coûts de maintenance** en anticipant les corrections nécessaires avant le lancement.
- **Assurer la stabilité des fonctionnalités de base** lors de l'ajout de nouvelles évolutions.

Scénarios de test retenus

- **Tests API**
  Les API seront entièrement automatisées afin de valider le travail de l'équipe de développement et garantir le bon fonctionnement des échanges entre le front-end et le back-end. L'objectif est de détecter d'éventuels bugs au niveau des endpoints, des données retournées et du traitement des requêtes.
- **Tests des fonctionnalités principales**
  Parmi les principales fonctionnalités (connexion, affichage des produits, gestion du panier), les deux plus critiques — la connexion et le panier — ont été sélectionnées pour des tests automatisés.

1. **Connexion** : S'assurer que l'authentification fonctionne, condition indispensable pour accéder à l'espace client et réaliser des achats.
2. **Panier** : Vérifier que l'ajout, la modification et la suppression de produits fonctionnent correctement, garantissant ainsi la possibilité de finaliser une commande valide.

- **Smoke tests**
  Des tests automatisés rapides ("smoke tests") sont réalisés pour valider les fonctionnalités essentielles du site et garantir leur stabilité lors de futures évolutions.

Cette campagne de tests s'inscrit dans une démarche qualité visant à fiabiliser la première version du site Eco Bliss Bath, à limiter les risques en production et à offrir la meilleure expérience possible aux utilisateurs dès le lancement.

Tests Effectués

Tests API – Non connecté

Connexion

Se connecter via l'API : http://localhost:8081/login
Méthode POST

1. Vérification de la connexion avec identifiant et mot de passe INCORRECTS :

- email : fakeuser@test.fr
- mot de passe : wrongpassword
  → **doit renvoyer un code 401 ou 403**

1. Vérification de la connexion avec identifiant et mot de passe CORRECTS :

- email : test2@test.fr
- mot de passe : testtest
  → **doit renvoyer un code 200**
  Récupérer le token pour s'authentifier dans l'API.

Profil utilisateur

- Requête sur les informations du profil utilisateur **avant connexion** :
  http://localhost:8081/me

Méthode GET
→ **doit renvoyer un code 401 ou 403** (accès à la ressource non autorisé)

Panier

- Requête sur les données confidentielles d'un utilisateur **avant connexion** :
  http://localhost:8081/orders

Méthode GET
→ **doit renvoyer un code 401 ou 403** (accès à la ressource non autorisé)

Tests API – Connecté

Pré-requis :
Se connecter via l'API : http://localhost:8081/login
Méthode POST

- email : test2@test.fr
- mot de passe : testtest
  → **doit renvoyer un code 200**
  Récupérer le token pour s'authentifier dans l'API.

Panier

1) Récupération des informations produit du panier

http://localhost:8081/orders
Méthode GET
→ **doit renvoyer un code 200**
doit retourner la liste des produits mis au panier.

---

2) Ajouter un produit disponible au panier

http://localhost:8081/orders/add

Méthode PUT

Indiquer dans le corps de la requête :

- l'id du produit (stock > 0) à ajouter
- la quantité à ajouter : 1

→ **doit renvoyer un code 200**

Méthode POST

- l'id du produit (stock > 0) à ajouter
- la quantité à ajouter : 1

→ **doit renvoyer un code 200**

---

3) Ajouter un produit en rupture de stock au panier

http://localhost:8081/orders/add

Méthode PUT

Indiquer dans le corps de la requête :

- l'id du produit (stock = 0) à ajouter
- la quantité à ajouter : 1

→ **doit renvoyer un code 400 ou 409**

Méthode POST

Indiquer dans le corps de la requête :

- l'id du produit (stock = 0) à ajouter
- la quantité à ajouter : 1

→ **doit renvoyer un code 400 ou 409**

---

4) Ajouter un produit avec une quantité négative

http://localhost:8081/orders/add

Méthode PUT

Indiquer dans le corps de la requête :

- l'id du produit à ajouter
- la quantité à ajouter : -2

→ **doit renvoyer un code 400 ou 422**

Produits

1) Récupérer la liste de tous les produits et leurs propriétés.

http://localhost:8081/products

Méthode GET

→ **doit renvoyer un code 200**

doit retourner la liste complète des produits disponibles ainsi que les champs « id, name, availableStock, skin, aromas, ingredients, description, price, picture, varieties » associés à chaque produit.

---

2) Récupérer une sélection aléatoire de produits

http://localhost:8081/products/random

Méthode GET

→ **doit renvoyer un code 200**

doit retourner exactement 3 produits sélectionnés aléatoirement.

Avis

1) Ajouter un avis

http://localhost:8081/reviews
Méthode POST
Indiquer dans le corps de la requête :
- title : "Test : Produit validé!"
- comment : "Test : odeur très agréable"
- rating : 5
  → **doit renvoyer un code 200**
  Le corps de la réponse doit contenir un id pour l'avis posté.

---

2) Ajouter un avis sans titre

http://localhost:8081/reviews
Méthode POST
Indiquer dans le corps de la requête :
- comment : "Avis sans titre."
- rating : 4
  → **doit renvoyer un code 400 ou 422**

---

3) Ajouter un avis avec une note invalide

http://localhost:8081/reviews
Méthode POST
Indiquer dans le corps de la requête :
- title : "Test avis note invalide"
- comment : "Note trop élevée."
- rating : 10 (note invalide, supposée hors limites, ex : max 5)
  → **doit renvoyer un code 400 ou 422**

---

4) Ajouter un avis avec un commentaire dangereux (XSS)

http://localhost:8081/reviews
Méthode POST
Indiquer dans le corps de la requête :
- title : "Test XSS"
- comment : "<script>alert('xss')</script>"
- rating : 3
  → **doit renvoyer un code 400 ou 422**
  Le commentaire ne doit pas être interprété ni stocké tel quel dans la base de données.

Tests Fonctionnels

Connexion

Se connecter avec des identifiants corrects

1. Aller sur la page d'accueil
2. Dans la barre de navigation, cliquer sur le lien « Connexion »
   → La page de connexion avec le formulaire s'affiche.
3. Dans le champ Email, entrer : test2@test.fr
4. Dans le champ Mot de passe, entrer : testtest

5. Cliquer sur le bouton « Se connecter »
   → En tant qu'utilisateur connecté, je suis redirigé vers la page d'accueil
   → Le lien vers le panier doit être visible dans la barre de navigation.

Se connecter avec des identifiants incorrects

1. Aller sur la page d'accueil
2. Dans la barre de navigation, cliquer sur le lien « Connexion »
   → La page de connexion avec le formulaire s'affiche.
3. Dans le champ Email, entrer : wrong@test.fr
4. Dans le champ Mot de passe, entrer : wrongpassword
5. Cliquer sur le bouton « Se connecter »
   → Un message doit alerter : « Identifiants incorrects ».
   → L'utilisateur reste sur la page de connexion.

Panier

Remarque préalable :
Avant chaque série de tests, la base de données (stocks, panier) est remise à zéro afin de garantir un état initial cohérent.

Pré-requis : Connexion

Avant chaque test, l'utilisateur se connecte avec :

- Email : test2@test.fr
- Mot de passe : testtest
  → Le lien vers le panier doit être visible dans la barre de navigation.

Ajouter un produit au panier

1. Aller sur la page « Produits »
2. Sélectionner un produit et cliquer sur « Consulter »

→ La page produit s'affiche

1. Cliquer sur le bouton « Ajouter au panier »

→ Le produit est ajouté au panier

1. Cliquer sur le lien « Mon panier »

→ Le produit ajouté est visible dans le panier

Modifier la quantité d'un produit dans le panier

1. Ajouter un produit au panier
2. Aller dans le panier
3. Modifier la quantité du produit en saisissant une nouvelle valeur (ex : 2)
   → La quantité affichée est mise à jour dans le champ

Refuser une quantité négative dans le panier

1. Ajouter un produit au panier
2. Aller dans le panier
3. Modifier la quantité du produit en saisissant une valeur négative (ex : -2)
   → La quantité reste à 1 (la valeur négative n'est pas acceptée)

Supprimer un produit du panier

1. Ajouter un produit au panier
2. Aller dans le panier

3. Cliquer sur l'icône poubelle pour supprimer le produit
   → Le produit n'est plus présent dans le panier (le panier peut devenir vide)
   → Le message « Votre panier est vide. Consultez nos produits. » s'affiche s'il n'y a plus aucun produit dans le panier.

Accepter une très grande quantité dans le panier

1. Ajouter un produit au panier
2. Aller dans le panier
3. Modifier la quantité du produit en saisissant une valeur très élevée (ex : 99999)
   → La quantité affichée ne doit pas dépasser 20.

Smoke Tests

Vérifier la présence des champs et boutons de connexion

1. Aller sur la page de connexion
   → Le champ « Email » doit être présent
   → Le champ « Mot de passe » doit être présent
   → Le bouton « Se connecter » doit être présent
   → Le lien vers la page d'inscription doit être présent

Vérifier la présence des champs et boutons de la page inscription

1. Aller sur la page d'inscription
   → Les champs « Nom », « Prénom », « Email », « Mot de passe », « Confirmation » doivent être présents
   → Le bouton « S'inscrire » doit être présent
   → Le lien vers la page de connexion doit être présent

Vérifier l'affichage des produits sur la page d'accueil

1. Aller sur la page d'accueil
   → Trois produits doivent être affichés
   → Chaque produit doit être visible

Vérifier la présence des boutons d'ajout au panier pour un utilisateur connecté

1. Se connecter avec un compte valide
2. Aller sur la page « Produits »
3. Cliquer sur un produit pour afficher la fiche
   → Le bouton « Ajouter au panier » doit être présent

Vérifier la présence du champ de disponibilité du produit

1. Aller sur la page « Produits »
2. Cliquer sur un produit pour afficher la fiche
   → Un champ indiquant la quantité en stock doit apparaître

Résultats de tests

Résultats des tests API

Connexion

1) Connexion avec identifiants incorrects (POST /login)

- **Résultat attendu :** Le serveur doit répondre avec un code 401 ou 403, sans délivrer de token.
- **Résultat obtenu :** **Conforme.** L'API retourne bien un code 401 ou 403 lorsqu'on tente de se connecter avec un email ou un mot de passe incorrect.

---

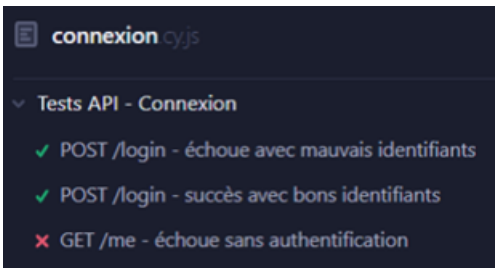
2) Connexion avec identifiants corrects (POST /login)

- **Résultat attendu :** Le serveur doit répondre avec un code 200 et retourner un token d'authentification valide dans la réponse.
- **Résultat obtenu :** **Conforme.** L'API retourne bien un code 200 et un objet contenant la propriété token lors d'une connexion avec des identifiants valides.

Profil Utilisateur

Accès à /me sans authentification (GET /me)

- **Résultat attendu :** Le serveur doit répondre avec un code 401 ou 403 (accès non autorisé).
- **Résultat obtenu :** **Non Conforme.** Le serveur retourne une erreur 500 (Internal Server Error) au lieu d'un code 401 ou 403.



Analyse du défaut :

- Ce comportement indique un bug côté back-end : la gestion des accès non authentifiés à l'endpoint /me n'est pas correctement implémentée.
- Pour reproduire le défaut :

1. Envoyer une requête GET à l'URL http://localhost:8081/me sans inclure de token d'authentification dans les headers.
2. Observer que le serveur répond avec un code 500 au lieu de 401/403.

Panier

1) GET /orders – échoue sans authentification

- **Résultat attendu :** 401 ou 403 (accès non autorisé)
- **Résultat obtenu :** **Conforme**, retourne bien 401 ou 403.

---

2) GET /orders – récupère le panier (après authentification)

- **Résultat attendu :** 200 et propriété orderLines (tableau de produits du panier) ; ou 404 si aucune commande en cours.
- **Résultat obtenu :** **Conforme**, retourne bien 200 avec la structure attendue si le panier contient des produits, ou 404 si aucune commande en cours.

3) → PUT /orders/add – ajouter un produit disponible (stock > 0)

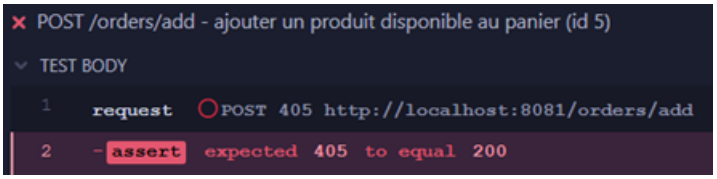
- **Résultat attendu :** 200 et panier mis à jour (orderLines présent).
- **Résultat obtenu :** **Conforme**, retourne bien 200 et la nouvelle liste du panier.

→ POST /orders/add – ajouter un produit disponible (stock > 0)

- **Résultat attendu :** 200 et panier mis à jour (orderLines présent).
- **Résultat obtenu :** **Non Conforme**, retourne un code 405.

Analyse du défaut :

- **POST donne une 405 (Method Not Allowed)** car le backend n'a pas défini de handler pour POST sur /orders/add, ou il a explicitement désactivé POST pour cette route.



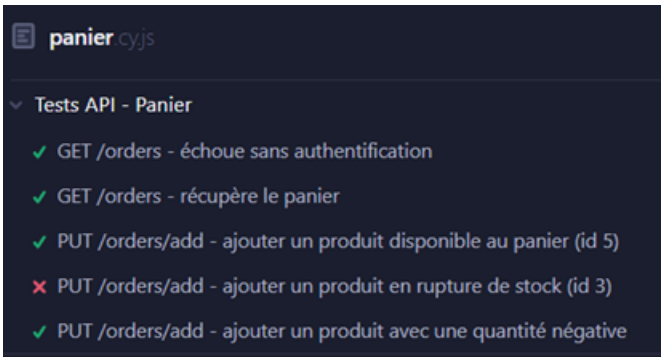
---

4) →PUT /orders/add – ajouter un produit en rupture de stock (id 3)

- **Résultat attendu :** 400 ou 409 (erreur métier)
- **Résultat obtenu :** **Non conforme** – l'API retourne 200 et ajoute le produit, alors que le stock est à 0.

Analyse du défaut :

- Le contrôle métier sur le stock n'est pas fait côté serveur.
- Pour reproduire le défaut : requête PUT à /orders/add avec un produit dont le stock est à 0, observer le code 200 au lieu de 400/409.



→POST /orders/add – ajouter un produit en rupture de stock (id 3)

- **Résultat attendu :** 400 ou 409 (erreur métier)
- **Résultat obtenu :** **Non conforme** – l'API retourne un code 405.

Analyse du défaut :

- **POST donne une 405 (Method Not Allowed)** car le backend n'a pas défini de handler pour POST sur /orders/add, ou il a explicitement désactivé POST pour cette route.

---

5) PUT /orders/add – ajouter un produit avec une quantité négative

- **Résultat attendu :** 400 ou 422 (erreur de validation)
- **Résultat obtenu :** **Conforme**, l'API refuse l'ajout et retourne bien une erreur (400 ou 422).

Produits

1) GET /products – récupération de tous les produits

- **Résultat attendu :** 200 et un tableau contenant la liste complète des produits, chaque produit comportant les champs attendus.
  (id, name, availableStock, skin, aromas, ingredients, description, price, picture, varieties)
- **Résultat obtenu :** **Conforme.** L'API retourne bien un code 200 et un tableau d'objets produits. Chaque produit du tableau contient l'ensemble des champs attendus.

---

2) GET /products/random – récupération de 3 produits aléatoires

- **Résultat attendu :** 200 et un tableau contenant exactement 3 produits.
- **Résultat obtenu :** **Conforme.** L'API retourne bien un code 200 et un tableau de 3 produits.

Avis

1) Ajouter un avis valide (POST /reviews)

Résultat attendu : Le serveur doit répondre avec un code 200. Le corps de la réponse doit contenir un id pour l'avis posté, ainsi que les champs « title », « comment », « rating ».
Résultat obtenu : **Conforme.** L'API retourne bien un code 200, un id d'avis, et les champs attendus dans la réponse.

---

2) Ajouter un avis sans titre (POST /reviews)

Résultat attendu : Le serveur doit répondre avec un code 400 ou 422 (erreur de validation).
Résultat obtenu : **Conforme.** L'API retourne bien une erreur (400 ou 422) lorsque le champ « title » est manquant.

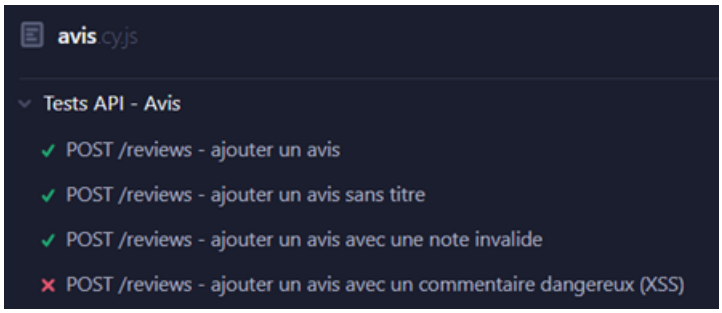
---

3) Ajouter un avis avec une note invalide (POST /reviews)

Résultat attendu : Le serveur doit répondre avec un code 400 ou 422 (erreur de validation).
Résultat obtenu : **Conforme.** L'API retourne bien une erreur (400 ou 422) lorsque la note (rating) est hors limites.

---

4) Ajouter un avis avec un commentaire dangereux (XSS) (POST /reviews)



Résultat attendu : Le serveur doit répondre avec un code 400 ou 422 (erreur de validation) ou bien neutraliser le script côté serveur.
Résultat obtenu : **Non conforme.** Le test échoue : l'API retourne un code 200 et accepte le commentaire dangereux sans le filtrer.
Analyse du défaut : Il s'agit d'une faille de sécurité potentielle (absence de filtre XSS côté backend).
Pour reproduire le défaut :

- Envoyer une requête POST à /reviews avec un commentaire contenant du code <script>.
- Observer que l'API accepte la requête (status 200) au lieu de la rejeter ou de neutraliser le contenu.

Résultat des tests fonctionnels

Connexion (Front-end)

1) Connexion avec identifiants corrects

Résultat attendu :
Après avoir saisi un email et un mot de passe valides, l'utilisateur doit être connecté :

- Redirection vers la page d'accueil
- Le lien vers le panier doit être visible dans la barre de navigation

Résultat obtenu : **Conforme.** Après connexion avec test2@test.fr / testtest, l'utilisateur est bien redirigé vers la page d'accueil et le lien vers le panier apparaît dans la barre de navigation.

---

2) Connexion avec identifiants incorrects

Résultat attendu : Après avoir saisi un email ou un mot de passe incorrect, la connexion doit échouer :

- Un message d'erreur « Identifiants incorrects » doit s'afficher
- L'utilisateur reste sur la page de connexion

Résultat obtenu : **Conforme.** Après tentative de connexion avec des identifiants erronés, un message d'erreur s'affiche (« Identifiants incorrects ») et l'utilisateur reste sur la page de connexion. Le lien vers le panier n'apparaît pas.

Panier (Front-end)

1) Ajouter un produit au panier

Résultat attendu : Après avoir ajouté un produit depuis la fiche produit, celui-ci doit apparaître dans le panier (le panier contient au moins un produit).
Résultat obtenu : **Conforme.** Après l'ajout d'un produit, il est bien visible dans le panier.

---

2) Modifier la quantité d'un produit dans le panier

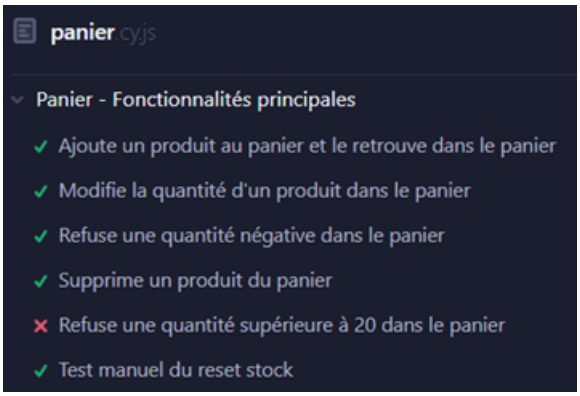
Résultat attendu : Après modification de la quantité (ex : passer de 1 à 2), la nouvelle valeur doit être affichée dans le champ de quantité.
Résultat obtenu : **Conforme.** La quantité du produit dans le panier est bien mise à jour selon la valeur saisie.

3) Refuser une quantité négative dans le panier

Résultat attendu : Si l'utilisateur saisit une quantité négative (ex : -2), la modification doit être refusée ou la valeur doit être automatiquement corrigée (ex : remise à 1).
Résultat obtenu : **Conforme.** Si une quantité négative est saisie, le champ revient à 1 (la valeur négative n'est pas acceptée).

4) Supprimer un produit du panier

Résultat attendu : Après suppression d'un produit via l'icône poubelle, le produit doit disparaître du panier (le panier peut devenir vide).
Résultat obtenu : **Conforme.** Après suppression, le produit n'est plus présent dans le panier.



5) Accepter une très grande quantité dans le panier

Résultat attendu : Si l'utilisateur saisit une quantité très élevée (ex : 99999), un message d'erreur doit apparaître et la commande ne doit pas pouvoir être validée (la limite attendue est de 20 articles maximum).
Résultat obtenu : **Non conforme.** La commande est validée avec 99999 articles, l'utilisateur est redirigé vers la page de confirmation alors qu'un message d'erreur aurait dû s'afficher.
Analyse du défaut : Ce comportement révèle l'absence de contrôle côté front-end (et/ou back-end) sur la quantité maximale autorisée par commande.
Pour reproduire le défaut :

- Ajouter un produit au panier, modifier la quantité à une valeur supérieure à 20 (ex : 99999)
- Observer qu'il n'y a pas de limite définie, toutes les valeurs sont prises en compte.

Smoke tests (Front-end)

1) Vérifie les champs et boutons de connexion

Résultat attendu : Sur la page de connexion, les champs « email » et « mot de passe », le bouton de validation, et le lien vers l'inscription doivent être présents.
Résultat obtenu : **Conforme.** Tous les champs et boutons attendus sont présents sur la page de connexion.

---

2) Vérifie les champs et boutons de la page inscription

Résultat attendu : Sur la page d'inscription, les champs « nom », « prénom », « email », « mot de passe », « confirmation », le bouton d'inscription et le lien vers la connexion doivent être présents.
Résultat obtenu : **Conforme.** Tous les champs et boutons attendus sont présents sur la page d'inscription.

---

3) Vérifie que 3 produits sont affichés sur la page d'accueil

Résultat attendu : La page d'accueil doit afficher exactement 3 produits, tous visibles.
Résultat obtenu : **Conforme.** Trois produits sont bien affichés et visibles sur la page d'accueil.

4) Vérifie la présence du bouton « Ajouter au panier » et des champs de disponibilité/stock sur la page produit (après connexion)

Résultat attendu : Sur la page d'un produit, le bouton « Ajouter au panier », le champ de disponibilité/stock et le champ de quantité doivent être présents.
Résultat obtenu : **Conforme.** Tous ces éléments sont présents sur la page détail produit après connexion.

[rapport d'incident]

Rapport d'incident – Bug majeur API

1. Anomalie : Ajout d'un produit en rupture de stock (PUT /orders/add)

Description de l'anomalie :
Lorsqu'un utilisateur tente d'ajouter au panier un produit dont le stock est à zéro (rupture de stock), l'API accepte la requête et retourne un code 200, comme si l'opération était valide.
Exemple du test :

```
cy.request({
  method: "PUT",
  url: "http://localhost:8081/orders/add",
  headers: { Authorization: `Bearer ${authToken}` },
  body: { product: 3, quantity: 1 }, // produit en rupture de stock
  failOnStatusCode: false,
}).then((response) => {
  expect([400, 409]).to.include(response.status); // attendu : erreur
});
```

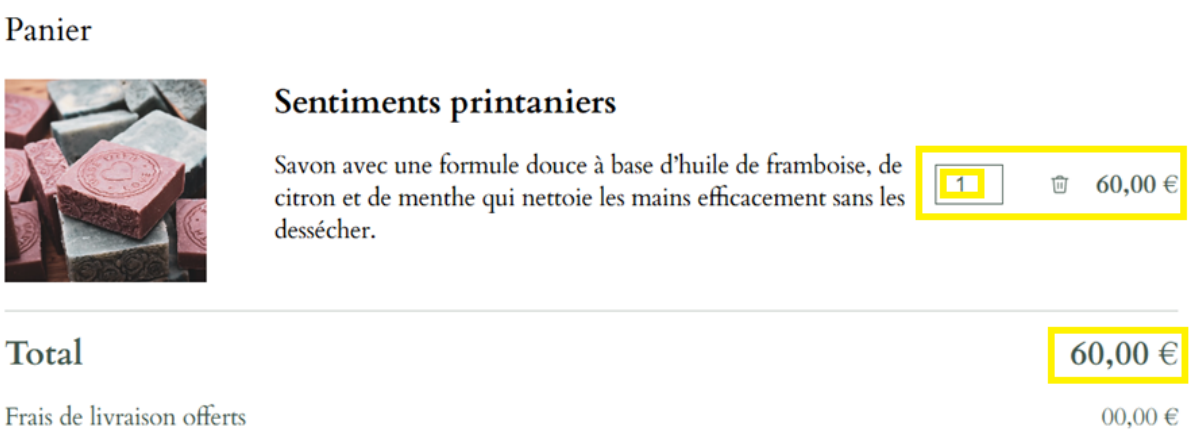
Résultat observé :
L'API retourne 200 et ajoute le produit au panier, alors que le stock est insuffisant.

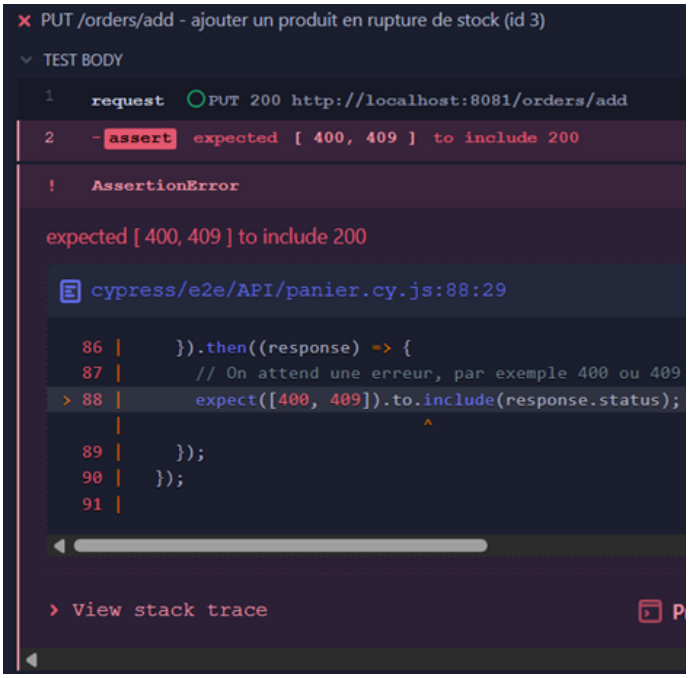
Pourquoi c'est un bug critique ?

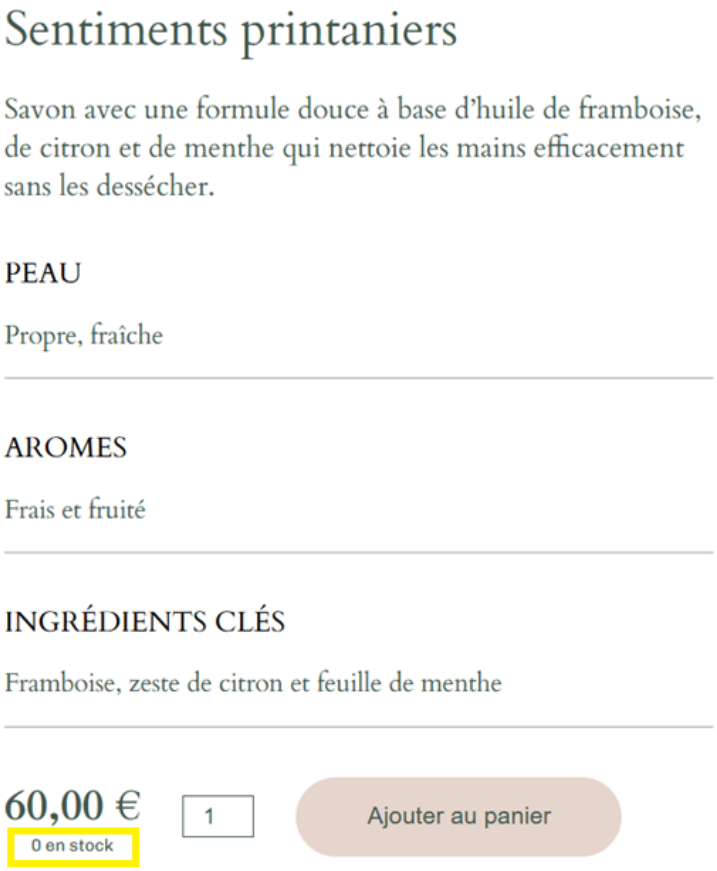
- Cela permet aux clients de commander des produits qui ne sont plus disponibles, ce qui entraîne des commandes impossibles à honorer.
- Cela dégrade fortement l'expérience utilisateur (annulation de commande, frustration).
- Cela peut causer des problèmes logistiques et de gestion de stock côté marchand.
- Ce type de faille peut impacter la crédibilité du site et générer des litiges.

Recommandations pour corriger ce bug :

1. **Contrôle métier côté serveur** :
   Ajouter une vérification stricte du stock disponible dans la logique de l'API avant d'accepter l'ajout d'un produit au panier.
   Si le stock est insuffisant (availableStock < quantité demandée), retourner une erreur HTTP :
   400 (Bad Request) ou **409 (Conflict)**, avec un message explicite : "Produit en rupture de stock".
2. **Validation côté front-end (optionnel mais recommandé)** :
   Empêcher la sélection de produits en rupture de stock dans l'interface utilisateur, pour éviter la frustration côté client.
3. **Tests automatisés** :
   Ajouter ou maintenir des tests automatisés pour s'assurer que ce contrôle reste effectif à chaque évolution du code.







2. Anomalie : Acceptation incohérente d'un commentaire dangereux (XSS) lors de l'ajout d'un avis (POST /reviews)

Description de l'anomalie :

L'API accepte la requête POST pour un avis contenant du code potentiellement dangereux (exemple : <script>alert("xss")</script>) et retourne un code 200, indiquant que l'opération a réussi. Cependant, le commentaire n'est ni affiché dans la liste des avis (ni en texte, ni exécuté), ce qui montre que le contenu n'est pas enregistré ou est filtré/supprimé après validation.

Exemple du test :

```
cy.request({
  method: "POST",
  url: "http://localhost:8081/reviews",
  headers: { Authorization: `Bearer ${authToken}` },
  body: {
    title: "Test XSS",
    comment: '<script>alert("xss")</script>',
    rating: 3
  },
  failOnStatusCode: false,
}).then((response) => {
  expect([400, 422]).to.include(response.status); // attendu : rejet ou neutralisation
});
```

Résultat observé :

L'API retourne 200 (succès), mais le commentaire n'est pas visible dans la liste des avis.

Pourquoi c'est un bug critique ?

- Il existe une incohérence entre la réponse de l'API (succès) et le traitement réel (avis non enregistré ou supprimé).
- Cela peut induire l'utilisateur en erreur, qui pense que son avis a bien été publié alors qu'il est en réalité ignoré.
- Ce comportement complique le débogage, la maintenance et la confiance dans le système.
- Même si le XSS n'est pas exploitable ici, la gestion des erreurs et la validation côté API sont insuffisantes.

Recommandations pour corriger ce bug :

1. **Validation explicite côté serveur** :

- Rejeter toute tentative d'enregistrement d'un commentaire contenant du code HTML/JS dangereux.
- Retourner un code d'erreur approprié (400 ou 422) avec un message explicite ("Le commentaire contient des caractères interdits").

1. **Alignement de la réponse API sur le traitement réel** :

- L'API ne doit jamais retourner 200 si l'avis n'est pas enregistré.
- Documenter clairement les règles de validation pour l'utilisateur.

1. **Tests automatisés** :

- Adapter les tests pour vérifier que l'API refuse explicitement les commentaires dangereux, et que l'utilisateur est informé de la raison du refus.

1. **Audit de sécurité** :

- Vérifier que ce comportement est cohérent sur tous les champs utilisateurs et sur l'ensemble de l'application.

Rapport d'incident – Bug mineur API

3. Anomalie : Refus de la méthode POST sur l'ajout au panier (405 Method Not Allowed) – (POST /orders/add)

Description de l'anomalie :

L'API refuse systématiquement les requêtes POST envoyées à l'endpoint /orders/add et retourne une erreur 405 (Method Not Allowed).

Ce comportement est observé alors que, selon les conventions REST et la logique métier, l'ajout d'un produit au panier devrait pouvoir être effectué via une requête POST.

Exemple du test :

```
it("POST /orders/add - ajouter un produit disponible au panier (id 5)
  cy.request({
    method: "POST",
    url: "http://localhost:8081/orders/add",
    headers: {
      Authorization: `Bearer ${authToken}`,
      "Content-Type": "application/json"
    },
    body: { product: 5, quantity: 1 },
    failOnStatusCode: false,
  }).then((response) => {
    expect(response.status).to.eq(200);
    expect(response.body).to.have.property("orderLines");
  });
});
```

Résultat observé :
L'API retourne une erreur 405 (Method Not Allowed) pour toute requête POST sur /orders/add, alors que la même opération avec la méthode PUT fonctionne correctement.

Pourquoi c'est un bug ?

- Il existe une incohérence entre la conception attendue de l'API (acceptation de POST pour ajouter un élément) et son implémentation réelle (POST non autorisé).
- Cela limite la compatibilité avec les clients ou outils qui s'attendent à pouvoir utiliser POST pour ce type d'opération.
- Ce comportement peut perturber les tests automatisés et la compréhension des développeurs intégrant l'API.
- L'absence de prise en charge de POST n'est pas documentée, ce qui peut entraîner une perte de temps en investigation.

Recommandations pour corriger ce bug :

- **Alignement sur les conventions REST :**
  Ajouter la prise en charge de la méthode POST sur l'endpoint /orders/add pour permettre l'ajout d'un produit au panier.
- **Retourner un code approprié :**
  Retourner 200 (succès) ou 201 (création) si l'ajout est effectué, ou un code d'erreur explicite en cas de problème métier (stock insuffisant, etc.).
- **Documentation :**
  Documenter clairement les méthodes HTTP acceptées pour chaque endpoint dans la documentation de l'API.
- **Tests automatisés :**
  Adapter les tests pour vérifier que POST fonctionne comme attendu, en plus de PUT si celui-ci reste supporté.
- **Gestion des erreurs :**
  Si POST n'est pas souhaité pour des raisons métier, retourner une erreur 405 avec un header Allow listant explicitement les méthodes supportées, et documenter ce choix.

4. Anomalie : Accès à /me sans authentification retourne une erreur 500 (GET /me)

Description de l'anomalie :

Lorsqu'un utilisateur non authentifié tente d'accéder à l'endpoint /me, l'API retourne une erreur 500 (Internal Server Error) au lieu d'un code d'erreur d'authentification approprié (401 ou 403).

Exemple du test :

```
cy.request({
  method: "GET",
  url: "http://localhost:8081/me",
  failOnStatusCode: false,
}).then((response) => {
  expect([401, 403]).to.include(response.status); // attendu : 401 ou 403
});
```

Résultat observé :

L'API retourne un code 500 au lieu de 401/403.

Pourquoi c'est un bug mineur ?

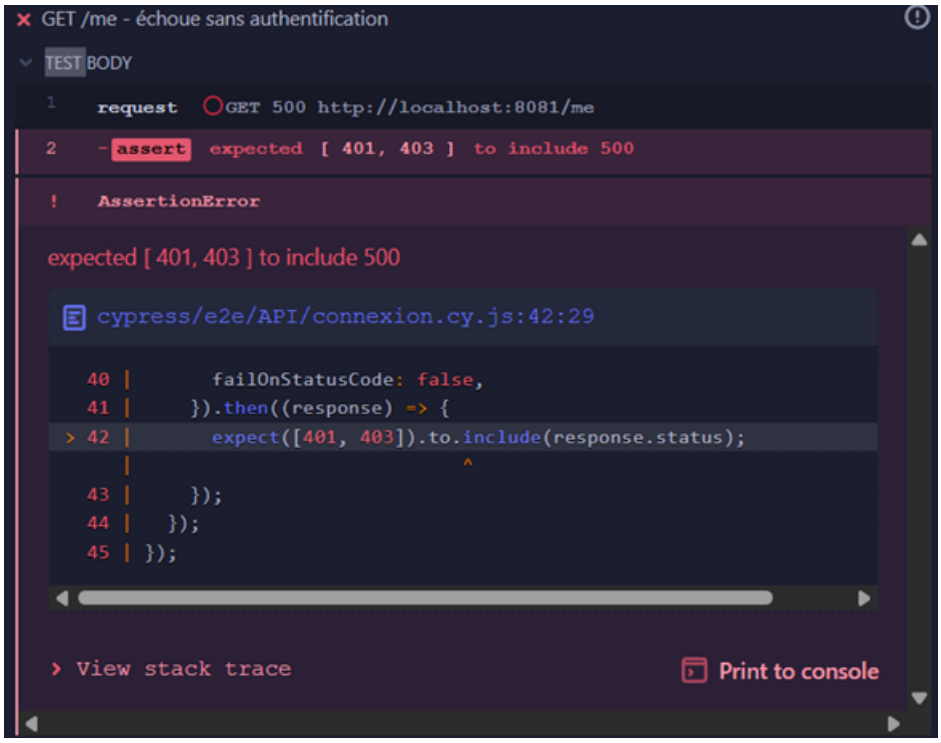
- Une erreur 500 indique un problème interne du serveur, alors qu'il s'agit ici d'un simple cas d'accès non autorisé.
- Cela ne compromet pas la sécurité, mais nuit à la clarté et à la robustesse de l'API.
- Un client API ou une application front-end ne peut pas distinguer une erreur d'authentification d'une vraie panne serveur, ce qui complique la gestion des erreurs côté client.

Recommandations pour corriger ce bug :

1. **Gestion correcte des erreurs d'authentification** :
- Modifier la logique de l'endpoint /me pour qu'il retourne systématiquement un code 401 (Unauthorized) ou 403 (Forbidden) en cas d'absence ou d'invalidité du token.
- S'assurer que toute tentative d'accès non authentifié à une ressource protégée ne déclenche jamais une erreur 500.

1. **Logging et monitoring** :
- Ajouter des logs explicites pour différencier les erreurs d'authentification des erreurs système réelles.

1. **Tests automatisés** :
- Maintenir des tests pour vérifier que tous les endpoints protégés réagissent correctement aux accès non autorisés.



Rapport d'incident – Bug majeur Front-end

5. Anomalie : Absence de limitation de la quantité à 20 articles dans le panier

Description de l'anomalie :
L'interface front-end du panier permet à l'utilisateur de saisir une quantité supérieure à la limite autorisée (20 articles) pour un produit, alors que la règle métier devrait empêcher toute commande au-delà de ce seuil.
Le champ de quantité accepte des valeurs très élevées (ex : 99999), et aucune alerte ou blocage n'est mis en place côté front.
Exemple du test :

```
it('Refuse une quantité supérieure à 20 dans le panier', () => {
  cy.visit('/products');
  cy.get('[data-cy="product-link"]').should('be.visible').eq(1).click();
  cy.get('[data-cy="detail-product-add"]').should('be.visible').click();
  cy.get('[data-cy="nav-link-cart"]').should('be.visible').click();
  // Tente de saisir une quantité supérieure à 20
  cy.get('input[data-cy="cart-line-quantity"]').should('be.visible').type('{selectall}99999');
  // Le champ doit être limité à 20
  cy.get('input[data-cy="cart-line-quantity"]').should('have.value', '20');
});
```

Résultat observé :
Le champ de quantité accepte la valeur saisie, même si elle dépasse la limite de 20. La commande peut être validée avec un nombre d'articles excessif.

Pourquoi c'est un bug critique ?

- Cela permet à un client de commander bien plus que la quantité maximale autorisée, ce qui peut entraîner des ruptures de stock, des erreurs de préparation de commande, ou des abus.
- Cela va à l'encontre des règles métier définies pour la boutique.
- Cela nuit à l'expérience utilisateur (absence de retour d'erreur, risque de commande annulée par la suite).
- Cela peut générer des problèmes logistiques et financiers pour le commerçant.

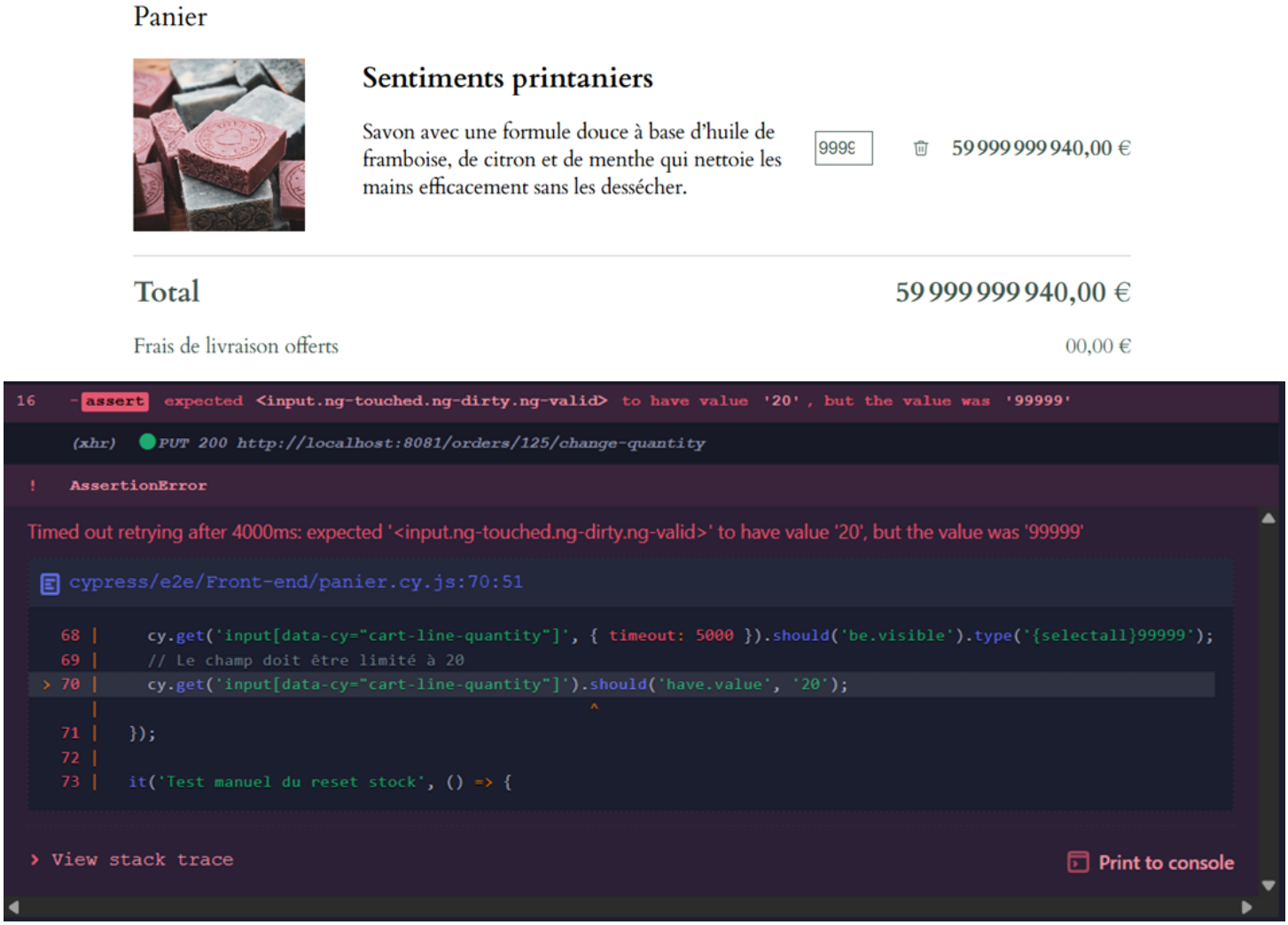
Recommandations pour corriger ce bug :

1. **Validation stricte côté front-end** :
- Limiter la valeur du champ de quantité à 20 via l'attribut max sur l'input, ou via une logique JavaScript qui bloque toute saisie supérieure à 20.
- Afficher un message d'erreur ou une alerte si l'utilisateur tente de dépasser la limite.

1. **Désactivation du bouton de validation** :
- Empêcher la validation de la commande (data-cy="cart-submit") si la quantité dépasse la limite autorisée.

1. **Validation côté back-end (sécurité)** :
- Toujours vérifier la quantité côté serveur pour éviter tout contournement via des requêtes manuelles.

1. **Tests automatisés** :
- Maintenir des tests pour s'assurer que la règle de limitation est respectée à chaque évolution du code.



Rapport de confiance et recommandations

1. Confiance dans la version à livrer

Le niveau de confiance dans la version actuelle est modéré à faible.
La majorité des fonctionnalités principales (authentification, gestion du panier, affichage des produits, parcours d'achat classique) fonctionnent correctement et passent les tests automatisés.
Cependant, plusieurs anomalies critiques ont été identifiées, qui remettent en cause la robustesse métier et la sécurité de l'application.

---

2. Criticité des anomalies trouvées

a) Bugs critiques à corriger en priorité

- **API – Ajout d'un produit en rupture de stock**
  Permet de commander des produits non disponibles, ce qui est bloquant pour l'exploitation et la satisfaction client.
- **API – Acceptation incohérente d'un commentaire XSS**
  Même si le XSS n'est pas exploitable, l'API retourne 200 alors que l'avis n'est pas enregistré. Cela crée une incohérence et peut masquer d'autres failles de validation.
- **Front-end – Limitation de la quantité dans le panier**
  L'utilisateur peut saisir une quantité supérieure à la limite autorisée (20), ce qui va à l'encontre des règles métier et peut provoquer des erreurs de stock, de logistique et d'expérience utilisateur.

b) Bug mineur

- **API – Accès à /me sans authentification retourne 500**

Ce n'est pas bloquant pour l'utilisateur final, mais cela nuit à la qualité technique de l'API et complique la gestion des erreurs côté client.

- **API – Refus de la méthode POST sur /orders/add (erreur 405)**

Ce n'est pas bloquant pour l'utilisateur final, mais cela nuit à la cohérence de l'API et complique l'intégration côté client.

---

3. Avis et recommandations

Les anomalies critiques doivent être corrigées AVANT la mise en production.

- Elles concernent des règles métier fondamentales (gestion du stock, validation des quantités) et la cohérence de l'API.
- Elles peuvent entraîner des litiges, de la frustration client, des pertes financières ou des failles de sécurité potentielles.

Les bugs mineurs peuvent être corrigés dans un second temps, mais il est recommandé de les traiter rapidement pour garantir la robustesse technique de l'API.

---

4. Conclusion

Je ne recommande pas de livrer la version actuelle en production tant que les bugs critiques ne sont pas corrigés.

La qualité globale est bonne sur les parcours standards, mais les défauts identifiés peuvent avoir un impact fort sur la satisfaction client, la sécurité et la crédibilité du service.

Priorité de correction :

1. Contrôle du stock à l'ajout au panier (API)
2. Validation et cohérence des retours API sur les avis (XSS/commentaires)
3. Limitation stricte des quantités côté front (et back)
4. Ajouter la prise en charge de la méthode POST sur l'endpoint /orders/add
5. Correction du code d'erreur sur /me sans authentification