**CHAPITRE 3 : RÉALISATION**

**3.1 Introduction**

Dans ce chapitre, nous présentons la conception et l’implémentation d’un système de tests automatisés avec Cypress. L’objectif principal est d’assurer la qualité des applications en validant les fonctionnalités critiques et en identifiant les anomalies potentielles. Le projet portera sur les fonctionnalités de la **Landing Page Bootstrap** en vue d’adapter ultérieurement cette solution à d’autres projets.

**Objectifs du projet**

* Valider les fonctionnalités critiques de l’application.
* Identifier les anomalies présentes et garantir la qualité logicielle.
* Mettre en place une architecture flexible pour les tests automatisés adaptable à divers projets.

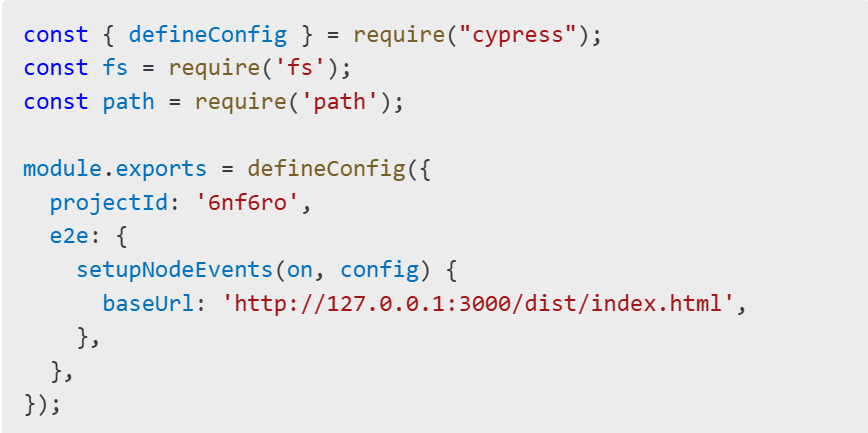
**Outil utilisé : Cypress (version 13.15.0)**

**Cypress** est un outil puissant pour les tests end-to-end (E2E) qui simplifie la validation des applications web.

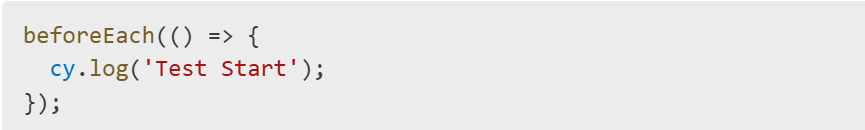
**3.2 Mise en place du système de tests automatisés**

**3.2.1 Configuration initiale**

La configuration du fichier **cypress.config.js** est une étape essentielle. Ce fichier contient les paramètres globaux, tels que l’URL de base de l’application et les configurations liées aux différents environnements.



**Ajout d’un message de démarrage des tests :** Dans le fichier cypress/support/e2e.js :



Ce message apparaît au début de chaque exécution de test.

**3.2.2 Rédaction des cas de test**

Les cas de test à implémenter ont été définis dans le tableau ci-dessous :

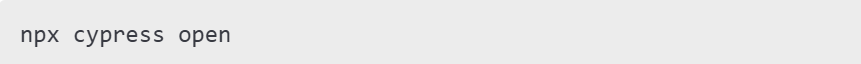
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **ID** | **Nom du Test** | **Description** | **Type de Test** | **Priorité** | | TC1 | Accès et chargement de Landing Page | Accéder à l’URL et vérifier que tous les éléments se chargent correctement | Interface utilisateur | Haute | | TC2 | Vérification de la barre de navigation | Vérifier l’affichage et l’état de la barre de navigation | Interface utilisateur | Moyenne | | TC3 | Validation de l’email | Vérifier les contraintes du champ d’entrée de l’email et la soumission | Unitaire, interface utilisateur | Haute | | TC4 | Affichage des fonctionnalités | Vérifier l’affichage correct des icones et leur contenu | Interface utilisateur | Faible | | TC5 | Vérification de la section de témoignages | Vérifier l’authenticité et l’affichage des témoignages | Interface utilisateur | Faible | | TC6 | Vérifications du footer | Vérifier les éléments et liens du footer | Interface utilisateur, fonctionnel | Moyenne | |

**3.2.3 Premier test : Accès à la Landing Page**

Voici un exemple de test automatisé :

 **3.2.4 Exécution des tests**

Pour exécuter les tests, utilisez la commande suivante :



L’interface graphique de Cypress permet de sélectionner le navigateur et le fichier de test à exécuter.

**3.3 Résultats des tests**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Nom du Test | Statut | Observation |
| TC1 | Accès et chargement de Landing Page | OK | RAS |
| TC2 | Vérification de la barre de navigation | OK | RAS |
| TC3 | Validation de l’email | OK | RAS |
| TC4 | Affichage des fonctionnalités | OK | RAS |
| TC5 | Vérification de la section de témoignages | OK | RAS |
| TC6 | Vérifications du footer | OK | RAS |

**Rapport technique :**

* Total de tests : 6
* Tests réussis : 6
* Durée totale : 9,56 secondes
* Rapport généré : cypress/results/test-results.json

**3.4 Configuration avancée : Gestion des environnements**

**3.4.1 Importance des environnements multiples**

* **Flexibilité :** Adapte les scripts aux différents environnements (staging, production).
* **Fiabilité :** Identifie les problèmes avant leur impact en production.
* **Efficacité :** Centralise les paramètres pour réduire les modifications manuelles.

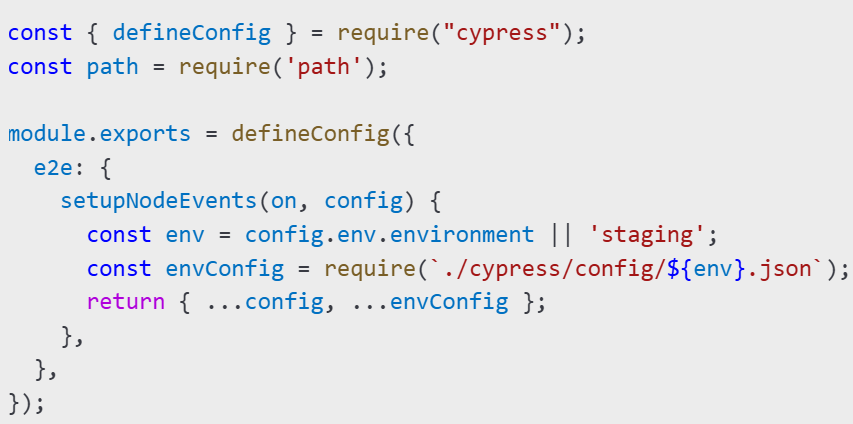
**3.4.2 Configuration des environnements dans Cypress**

Créer des fichiers JSON pour chaque environnement dans le dossier cypress/config :

**Exemple : staging.json**



**Charger la configuration dynamiquement dans cypress.config.js :**



**Exécution pour un environnement spécifique :**



**3.5 Stratégies de maintenance et d’évolution**

**3.5.1 Maintenance des tests**

* **Mises à jour régulières :** Les tests doivent être ajustés en fonction des modifications du code ou des nouvelles fonctionnalités.
* **Révision périodique des cas de tests :** Identifier et retirer les tests obsolètes.
* **Utilisation des hooks :** Simplifier la maintenance avec des fonctions globales (before, after, etc.).

Exemple d’utilisation de hooks dans Cypress :



**3.5.2 Évolutivité des tests**

* **Adoption d’un modèle modulaire :** Diviser les tests en composants réutilisables pour simplifier l’ajout de nouvelles fonctionnalités.
* **Automatisation dans le pipeline CI/CD :** Intégrer les tests dans des outils tels que Jenkins, GitLab CI/CD, ou Bitbucket Pipelines.

**3.6 Intégration avec le pipeline CI/CD**

**3.6.1 Rôle de l’automatisation**

L’intégration des tests dans le pipeline CI/CD garantit :

* Une détection rapide des régressions.
* Une validation continue des builds.

**3.6.2 Configuration avec Bitbucket Pipelines**

Voici un exemple de fichier bitbucket-pipelines.yml pour l’exécution des tests Cypress :



**Observation des résultats :** Les résultats des tests sont automatiquement remontés dans Bitbucket pour consultation.

**3.7 Limites et perspectives**

**3.7.1 Limites actuelles**

* **Adaptabilité :** Les tests sont spécifiquement conçus pour une application Bootstrap, ce qui limite leur réutilisation immédiate.
* **Environnement d’exécution :** La configuration des environnements multiples nécessite une gestion rigoureuse pour éviter les erreurs.
* **Temps d’exécution :** Bien que rapide, l’exécution peut être optimisée pour des suites de tests plus volumineuses.

**3.7.2 Perspectives**

* Étendre la solution à d’autres frameworks (React, Angular).
* Incorporer des outils de reporting avancés comme Allure ou Mochawesome.
* Augmenter la couverture des tests pour inclure des scénarios complexes (stress tests, tests de performance).