**Comment Cypress interagit avec un projet ?**

Cypress interagit directement avec le Document Object Model (DOM) de l'application web en cours de test. Il manipule les éléments comme un utilisateur le ferait (clics, saisies, navigation, etc.), en utilisant des commandes comme cy.get(), cy.click(), ou cy.type().

Durant l’exécution des tests automatisés dans le navigateur donné, il simule les interactions utilisateur pour vérifier que les éléments de l'UI (boutons, formulaires, etc.) fonctionnent comme attendu détectant ainsi les erreurs visuelles ou fonctionnelles liées à l'interface.

**À quel niveau Cypress opère-t-il dans un projet ?**

Cypress fonctionne principalement au niveau frontal (navigateur et UI), mais il interagit aussi avec d'autres niveaux en :

* Configuration initiale :

Cypress est installé dans le projet en tant que dépendance. Le fichier cypress.config.js est configuré pour indiquer l'URL de base et d'autres paramètres.

* Préparation des tests :

Définir des cas de test dans un ou plusieurs fichiers test.cy.js.

* Exécution des tests :

Cypress ouvre un navigateur et exécute les tests dans un environnement isolé. Il interagit avec le DOM, les API, et l'UI en fonction des scripts pensés et tout en surveillant les changements.

**Comment implémenter des tests avec Cypress à un projet tiers ?**

Implémenter des tests Cypress dans un projet tiers implique plusieurs étapes, en passant de la configuration de Cypress à l’écriture des fichiers de test. Vous procéderons donc à quelques étapes cruciales à savoir :

* Configurer l’environnement :

Modifiez le fichier de configuration cypress.config.js pour définir l’URL de base du projet tiers.

Cypress.config.js

const { defineConfig } = require("cypress");

module.exports = defineConfig({

e2e: {

baseUrl: 'https://url-du-projet-tiers.com', // url de l'application cible

supportFile: 'cypress/support/e2e.js', // Fichier support pour des commandes personnalisées

},

});

* Créer des scénarios de test

Les scénarios de test se trouvent dans le dossier cypress/e2e. Créez un fichier de test, par exemple test.cy.js, et écrivez vos tests :

* Ajouter des données de test (fixtures)

Si le projet nécessite des données pour simuler des scénarios spécifiques (par exemple, utilisateur connecté), utilisez le dossier cypress/fixtures.

Exemple d'utilisation :

Ajoutez un fichier JSON, user.json :

user.json

{

"username": "testuser",

"password": "password123"

}

Référez-vous à ces données dans vos tests :

test.cy.js

it('Connexion utilisateur', () => {

cy.fixture('user').then((user) => {

cy.visit('/login');

cy.get('#username').type(user.username);

cy.get('#password').type(user.password);

cy.get('button[type="submit"]').click();

});

});

* Ajouter des commandes personnalisées

Si vous devez répéter certaines actions, comme une connexion utilisateur, utilisez le fichier cypress/support/e2e.js pour créer des commandes :

e2e.js

Cypress.Commands.add('login', (username, password) => {

cy.visit('/login');

cy.get('#username').type(username);

cy.get('#password').type(password);

cy.get('button[type="submit"]').click();

});

Utilisez cette commande dans vos tests :

test.cy.js

it('Test après connexion', () => {

cy.login('testuser', 'password123');

cy.get('.welcome-message').should('exist');

});

* Intégration avec un projet tiers (conseils)

Authentification : Si le projet nécessite une authentification complexe, utilisez cy.intercept() pour simuler des requêtes d'authentification.

Données dynamiques : Préparez des fixtures ou connectez-vous à l’API du projet pour injecter des données en temps réel.

Enfin, identifiez les points d’échec pour ajuster vos tests ou signaler des bugs au propriétaire du projet tiers.