### **1.1 Introduction to Cypress**

Cypress is a modern front-end testing tool designed to address the challenges developers and QA engineers face when testing modern web applications. Unlike traditional testing tools, Cypress is built on a new architecture that makes it faster, more reliable, and easier to use. It enables developers to write tests that run in the browser, allowing them to test everything that runs in the browser, such as user interactions, server responses, and the application's overall behavior.

#### **1.1a Key Features of Cypress:**

1. **Fast and Reliable**: Cypress runs in the same run-loop as the application, providing better performance and reliability.
2. **Time Travel**: Cypress takes snapshots as your tests run, allowing you to hover over each command in the Test Runner to see exactly what happened at each step.
3. **Debuggability**: Developer tools are built into Cypress, making it easy to debug tests directly in the browser.
4. **Automatic Waiting**: Cypress automatically waits for commands and assertions before moving on, eliminating the need for arbitrary waits.
5. **Spies, Stubs, and Clocks**: Cypress has built-in support for spies, stubs, and clocks, making it easy to control and inspect the behavior of functions and timers.
6. **Network Traffic Control**: Cypress provides control over network requests and responses, allowing you to stub and intercept requests.

### **1.1b Architecture of Cypress**

Cypress has a unique architecture that differentiates it from other testing tools. Here's an overview of how it works:

#### **Components of Cypress:**

1. **Test Runner**: The Cypress Test Runner is an interactive runner that displays tests as they execute, offering a visual and intuitive way to see what’s happening in real-time.
2. **CLI (Command Line Interface)**: The Cypress CLI is used to run tests from the command line. It provides various options and configurations to customize test runs.
3. **Cypress Server**: This server handles all the communication between the Cypress Test Runner and the application under test. It intercepts and modifies network requests and responses, ensuring that tests are isolated and predictable.
4. **Browser**: Cypress runs directly in the browser, executing the same JavaScript code that runs in the application. This ensures that tests are executed in an environment as close to the real user experience as possible.

#### 

#### 

#### 

#### **1.1c How Cypress Works:**

1. **Installation**: Cypress can be installed via npm or yarn. Once installed, it includes everything you need to get started with testing.

npm install cypress --save-dev

**Test Writing**: Tests are written in JavaScript using Mocha and Chai syntax. They are stored in the cypress/integration folder.

Example Test:

describe('My First Test', () => {

it('Does not do much!', () => {

expect(true).to.equal(true)

})

})

**Test Execution**: Tests can be executed using the Cypress Test Runner, which provides a visual interface for running and debugging tests. Alternatively, tests can be run headlessly using the CLI for CI/CD integration.

npx cypress open # Opens the Test Runner

npx cypress run # Runs tests in headless mode

**Interacting with the DOM**: Cypress provides commands for interacting with the DOM, such as cy.visit(), cy.get(), cy.click(), and cy.type().

Example DOM Interaction:

describe('My Second Test', () => {

it('Visits a webpage and interacts with elements', () => {

cy.visit('https://example.com')

cy.get('input[name="username"]').type('myusername')

cy.get('input[name="password"]').type('mypassword')

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

})

})

**1.2 Assertions**: Cypress uses Chai assertions to validate the behavior of the application.

Example Assertion:

describe('My Third Test', () => {

it('Checks if the element is visible', () => {

cy.visit('https://example.com')

cy.get('h1').should('be.visible')

})

})

#### **1.3 Cypress Lifecycle:**

1. **Setup**: Before tests run, Cypress sets up the testing environment and initializes the application state.
2. **Execution**: Cypress executes the test commands sequentially, interacting with the application and making assertions.
3. **Teardown**: After the tests complete, Cypress cleans up the environment and prepares for the next test run.

### **1.4 Benefits of Using Cypress**

1. **Developer-Friendly**: Cypress is designed with developers in mind, providing a smooth and intuitive experience for writing and running tests.
2. **Integrated Testing**: It supports end-to-end, integration, and unit testing within the same framework.
3. **Fast Feedback Loop**: Cypress provides instant feedback, making it easy to identify and fix issues quickly.
4. **Consistent Results**: By running tests in the same browser context as the application, Cypress ensures that tests are consistent and reliable.