Cypress Question Bank

1. Different blocks in cypress

**Describe, It, before/before each, after/after each block**

These blocks help in organizing your test suite effectively and ensure proper setup and teardown actions are taken care of, providing a clean and reliable testing environment.

1. How we can mock the data in cypress?

cy.intercept() intercepts the network request made by the application to fetch data from the specified endpoint .

Instead of making the actual network request, Cypress will respond with the mocked data provided in the fixture file.

1. Where we can store mocking data?

Fixture file

1. What is use of commands.js file?

To create custom commands

The commands.js file in Cypress serves as a central location for defining custom commands that you want to use across your Cypress tests. This file allows you to extend Cypress' built-in command set with your own custom commands, making it easier to write and maintain tests.

Here are some use cases and the purposes of the commands.js file:

Reusable Actions, **Abstraction of Complex Interactions, Domain-Specific Language (DSL), Code Reusability**

1. What is cy.spy?

cy.spy() is a method provided by Cypress for creating spies in your test cases. Spies are used to monitor function calls, including how many times they were called, with what arguments, and what they returned.

cy.spy() is commonly used for:

Mocking and monitoring function calls.

Verifying that certain functions are called with expected arguments.

Counting how many times a function is called during a test.

This feature is particularly useful for testing callback functions, event handlers, and other parts of your application where you want to verify that certain functions are invoked as expected.

1. How we can get element in cypress? /How many types of locators we used in Cypress?

**Using CSS Selectors:** cy.get()

**Using Custom Attributes**: cy.get('[data-test-id="myButton"]')

**Chaining Commands**: cy.get('.container').find('button'),

cy.get('.myClass').first()

cy.get('.myClass').last()

cy.contains('Submit')

1. Different types of locators? / How many types of locators we used in Cypress?

By tag name , by id, by class, by link by partial text, custom attributes

1. Test case for login functionality

Test case to write

1. How we can click on invisible button?

In Cypress, you can force-click on invisible or hidden elements using the .click() command with the { force: true } option. This option overrides Cypress' default behaviour of not interacting with elements that are not visible or are obscured.

cy.get('.invisible-button').click({ force: true })

1. Types of hooks in cypress.

In Cypress, hooks are functions that allow you to run code at specific points during the test execution lifecycle. They provide a way to set up and tear down test fixtures, perform actions before or after each test, and customize test behaviour. There are several types of hooks available in Cypress:

1. Before All (before): Runs once before all tests in the test suite.
2. Before Each (beforeEach): Runs before each test in the test suite.
3. After Each (afterEach): Runs after each test in the test suite.
4. After All (after): Runs once after all tests in the test suite.
5. Tell me about your experience and which tools u used and aware about which Technologies?

Certainly! As a Cypress test engineer with three years of experience, I have had the opportunity to work extensively with Cypress for end-to-end testing of web applications. During my tenure, I have gained valuable experience in designing, implementing, and maintaining test automation frameworks using Cypress. Here's a summary of my experience:

1. Experience with Cypress: I have hands-on experience using Cypress to write end-to-end tests for web applications. This includes writing test cases, organizing test suites, and running tests in various environments.

2. Test Automation Frameworks: I have developed and maintained test automation frameworks using Cypress, ensuring scalability, reliability, and maintainability of the test suite.

3. CI/CD Integration: I am proficient in integrating Cypress tests into Continuous Integration (CI) pipelines using tools like Jenkins, Travis CI, or GitHub Actions. This involves setting up build configurations, triggering test runs, and analysing test results.

4. Programming Languages: I am proficient in JavaScript, which is the primary language used for writing tests in Cypress. Additionally, I have experience with other programming languages such as Python, JavaScript, which can be beneficial for test automation tasks.

5. Web Technologies: I am well-versed in web development technologies such as HTML, CSS, and JavaScript. This understanding helps me effectively interact with web elements and handle various scenarios in Cypress tests.

6. Version Control Systems: I have experience using Git and platforms like GitHub or GitLab for version control, collaboration, and code management. This ensures proper versioning and tracking of test automation code.

7. Agile Methodologies: I am familiar with Agile software development methodologies, including Scrum or Kanban. This experience enables me to work collaboratively within Agile teams, participate in sprint planning, and adapt to changing requirements.

8. Problem-Solving and Troubleshooting: I have honed my problem-solving skills through identifying, analyzing, and resolving issues encountered during test automation activities. This includes debugging failing tests, investigating root causes, and implementing effective solutions.

9. Continuous Learning and Adaptability: I am committed to continuous learning and staying updated with the latest advancements in test automation tools, technologies, and best practices. I am adaptable to new tools and methodologies to enhance the efficiency and effectiveness of test automation processes.

Overall, my experience as a Cypress test engineer has equipped me with the necessary skills, knowledge, and expertise to contribute effectively to test automation projects and ensure the delivery of high-quality software products.

Short answer

As a Cypress test engineer with three years of experience, I've worked extensively on automating end-to-end tests for web applications. Here's a brief overview of my experience:

1. Cypress Expertise: Skilled in writing, organizing, and running Cypress tests for web apps.

2. Framework Development: Developed and maintained scalable, reliable test automation frameworks using Cypress.

3. CI/CD Integration: Proficient in integrating Cypress tests into CI pipelines using tools like Jenkins or GitHub Actions.

4. Programming Languages: Proficient in JavaScript, Python, and other languages beneficial for test automation.

5. Web Technologies: Well-versed in HTML, CSS, and JavaScript, enabling effective interaction with web elements in Cypress tests.

6. Version Control: Experienced in using Git and platforms like GitHub for code management and collaboration.

7. Agile Methodologies: Familiar with Agile practices, facilitating collaborative work within Agile teams.

8. Problem-Solving Skills: Skilled in troubleshooting and resolving issues encountered during test automation activities.

9. Continuous Learning: Committed to staying updated with the latest advancements in test automation tools and methodologies.

Overall, I bring a strong foundation in Cypress testing and a dedication to delivering high-quality software through automation.

1. Explain the folder structure in Cypress?

Cypress --Root folder

Downloads

E2e folder where we store specs files

Fixture folder for payload .json, .js, .xls

Support for custom commands

Config to configure plugins

Package.json

**Downloads folder**

Long answer

**1. Package.json file:** this folder contains the project related information which you setup at the time of installation.

i. Name, version, description,

ii. Git repository information

iii. Scripts commands

iv. License

v. Dev Dependencies: this will hold the modules & its versions installed and saved. Such as cypress, and other plugins

**2. Package-lock.json file:** This will lock the exact versions of dependencies mentioned in package.json. This file will automatically update on npm install/update any package.

**3. Cypress.config.js file:** Used for intelligent code completion. This will hold cypress specific configuration required for running and executing test scripts. Some plugins also require the configurations to be set here. We can add/update the information within this such as env, timeouts, reporter information, folder configurations etc.

**4. Node\_modules folder:** This folder contains all the installed packages using npm install

**5. Cypress folder:** Cypress folder is the main folder which further has 4 folders inside created by default by cypress.

**6. Downloads folder:** This folder is used to store temporary files which got downloaded from web during test script execution

**7. E2e folder:** This is the main folder where test files has to be stored. You can create sub-folders as well. Extension of the test files would be \*.cy.js (cy.ts for typescript). other types of test file extensions can be set in cypress configuration file.

**8. Fixture folder:** This folder has json files to save test data. You can create multiple files/fodler inside this.

**9.** Support folder: Support folder further contains 2 files. Those are commands.js and e2e.js.

**10. Commands.js file:** This file is used to create custom commands. Any script which we can frequently use for our tests execution by directly calling using

*cy.<custom-command-name>.*

**11. E2e file:** This file is used for keeping the script which we want to be processed before each test file runs. Global configurations.

1. What is cy.intercept ?

cy.intercept() is a powerful method provided by Cypress for intercepting and modifying HTTP requests made by your application. It allows you to stub or mock network requests, enabling you to control the behaviour of your application during testing. This is particularly useful for simulating various server responses or testing specific scenarios, such as error handling or edge cases.

1. Use of. find() ?

In Cypress, the .find() method is used to locate and select DOM elements within a parent element. It allows you to narrow down your selection to specific descendants of an element, making it easier to interact with nested elements in your tests.

cy.get('parent-selector').find('child-selector')

1. Use of .eq()

The .eq() method in Cypress is used to select a specific element from a collection of elements based on its index. It allows you to target and interact with a particular element within a set of matched elements.

cy.get('selector').eq(index);

Using .eq() is helpful when you need to interact with a specific element within a collection, such as selecting the nth item in a list, accessing a specific table row or column, or targeting a particular element in a set of matched elements

1. Hadling table in cypress?

Handling tables in Cypress involves selecting table elements and performing various actions such as retrieving data, asserting values, or interacting with table rows and cells. Cypress provides several methods to facilitate working with tables efficiently. Here's a guide on how to handle tables in Cypress:

// Select the entire table

cy.get('table');

// Select specific rows or cells within the table

cy.get('table').find('tr').eq(2).find('td').eq(1);

// Retrieve text content from a specific cell

cy.get('table td').eq(2).invoke('text').then(text => {

// Assert or manipulate the retrieved text

});

// Retrieve text content from all cells in a row

cy.get('table tr').eq(1).find('td').invoke('text').then(cellTexts => {

// Assert or manipulate the text content of each cell

});

// Assert the text content of a specific cell

cy.get('table td').eq(2).should('contain', 'expectedText');

// Assert the text content of all cells in a row

cy.get('table tr').eq(1).find('td').each(cell => {

cy.wrap(cell).should('contain', 'expectedText');

});

1. Which langauges cypress support?

javaScript, type script

1. Use of fixture folder, how to access this?

The fixtures folder in Cypress stores static data files. To access this data in tests, use cy.fixture() command followed by the filename (e.g., cy.fixture('example.json')). Load fixture data within a then() callback and use it in your test assertions or actions. Ensure fixture files are stored in the fixtures folder within your Cypress project directory.(or we can import fixture file using import statement)

1. How to upload file in cypress

Uploading a file in Cypress involves utilizing the cy.fixture() command to load the file, then using cy.get() to target the file input element, and finally, using the .attachFile() command to attach the file to the input.

cy.fixture('example.txt').then(fileContent => {

// Use file content in the test

});

Use cy.get() to select the file input element on the page. Ensure you have the correct selector for the input element.

cy.get('input[type="file"]').attachFile('example.txt');

1. what is the use of uploading file in cypress

Uploading files in Cypress is useful for testing features or functionality that involve file uploads in web applications. This capability allows you to simulate user interactions where users upload files through file input elements on web pages. Some common scenarios where uploading files in Cypress might be beneficial include:

**Testing File Upload Functionality:** Verifying that file upload functionality works as expected, including uploading various file types, handling large files, and validating file upload success.

**Testing File Processing**: Testing the behavior of the application after files are uploaded, such as verifying file processing, data extraction, or any subsequent actions triggered by file uploads.

**Testing Form Submissions**: Testing forms that include file upload fields as part of the form submission process, ensuring that files are correctly uploaded along with other form data.

**Testing User Experience:** Ensuring a smooth and intuitive user experience during file uploads, including handling of errors, progress indicators, and feedback to the user.

By simulating file uploads in Cypress tests, you can thoroughly validate the behaviour of your web application and ensure that file-related features are functioning correctly across different scenarios and edge cases. This helps improve the quality and reliability of your application, providing a better experience for users who interact with file upload functionality.

1. How to skip test case in cypress?

.skip() function.

1. What is alternative word for describe in cypress

context() is identical to describe() and specify() is identical to it() , so choose whatever terminology works best for you.

1. Can we write custom commands in cypress?

Yes, you can write custom commands in Cypress to encapsulate reusable actions or assertions. Define custom commands using Cypress.Commands.add() in a JavaScript file (e.g., commands.js) within the support folder of your Cypress project. Then, use these custom commands in your test cases like any other Cypress command, improving code readability and maintainability.

1. How to manage environment in cypress

managing the environment typically refers to handling different configurations or settings for your tests, such as URLs, API endpoints, credentials, and other variables that may vary across different testing environments (e.g., development, staging, production).

Managing environments in Cypress means setting up your tests to work correctly in different situations. You do this by storing basic settings in a file called cypress.json. Then, you can use special variables called environment variables to change these settings when needed, like switching between a testing site and a live site. You can also add extra features using plugins. Inside your tests, you can access these settings easily. This helps ensure your tests run smoothly no matter where they're being run or what settings they need to work with.

Cypress allows you to use configuration files (cypress.json by default) to store environment-specific settings. You can create multiple configuration files for different environments (e.g., cypress.dev.json, cypress.staging.json, cypress.prod.json), each containing environment-specific configurations.

1. How we can use traversing methods in cypress?

In Cypress, traversing methods are used to interact with elements on a webpage by navigating through the HTML structure. Here's how you can use traversing methods effectively:

**1. cy.get():** Start by selecting an element using `cy.get()`. This is the base command to begin traversing.

**2. Traversing Downwards:**

- Use `.find()` to find child elements of the selected element.

- Use `.children()` to select direct children of the selected element.

**3. Traversing Sideways:**

- Use `.siblings()` to select elements that share the same parent.

- Use `.next()` or `.prev()` to select the next or previous sibling of the selected element.

**4. Traversing Upwards:**

- Use `.parent()` to select the parent element of the selected element.

- Use `.closest()` to find the closest ancestor that matches a given selector.

**5. Filtering:**

- Use `.filter()` to reduce the set of matched elements based on a condition.

- Use `.eq()` to select a specific element from a set of matched elements by index.

**6. Chaining Traversal Methods:** You can chain multiple traversal methods together to precisely target the desired element.

Here's an example demonstrating the use of traversing methods in Cypress:

// Traverse downwards

cy.get('.parent')

.find('.child') // Find child elements

.children('.grandchild'); // Select direct children

// Traverse sideways

cy.get('.current')

.siblings('.other'); // Select siblings

// Traverse upwards

cy.get('.child')

.parent('.parent'); // Select parent element

// Filter elements

cy.get('ul li')

.filter(':contains("Item 3")'); // Filter based on content

// Chaining traversal methods

cy.get('.parent')

.find('.child')

.closest('.grandparent') // Find the closest ancestor

.find('.grandchild')

.eq(1); // Select the second matched element

By understanding and utilizing these traversing methods, you can efficiently navigate through the DOM structure in Cypress to interact with elements as needed during your tests.

1. How we can find element if there is no class and ID?

If an element doesn't have a class or an ID, you can still locate it using other attributes, its position relative to other elements, or its content. Here are several approaches you can take:

**1. Using Other Attributes:** Look for other attributes that uniquely identify the element, such as name, data-attributes, or aria-attributes.

**2. Using CSS Selectors:** Utilize CSS selectors to target elements based on their attributes, element types, or relationships with other elements. For example:

- cy.get('[data-testid="unique-id"]') to select by a custom data-testid attribute.

- cy.get('input[type="submit"]') to select by input type.

- cy.get('a[href="/about"]') to select by href attribute.

**3. XPath:** Use XPath expressions to locate elements based on their attributes or content. XPath provides powerful querying capabilities but can be more complex than CSS selectors. For example:

-cy.xpath('//button[@name="submit"]') to select a button with the name attribute equal to "submit".

**4. Relative Position:** If the element is located relative to another known element, you can use traversal methods to navigate to it. For instance:

cy.get('.parent').find('span'); // Find <span> within an element with class 'parent'.

**5. Content:** If the element's content is unique or predictable, you can use `contains()` to find it. However, be cautious with this approach as it may match multiple elements if the content is not specific enough.

**6. Combining Strategies:** You can combine multiple strategies to create a more specific selector. For instance:

cy.get('input[type="text"][placeholder="Enter your name"]'); // Select an input element with specific type and placeholder text.

Always prioritize selecting elements based on attributes or characteristics that are unique or stable across different states of your application to ensure reliable tests. If none of the above methods work reliably, consider working with your development team to add specific attributes or identifiers to the element to facilitate testing.

1. How we can find and validate logo of web application?

To find and validate the logo of a web application in Cypress, you can follow these steps:

**1. Locate the Logo Element:** Inspect the webpage to identify the element containing the logo. Typically, logos are displayed as images (`<img>` tags) or background images.

**2. Select the Logo Element:** Use Cypress commands like `cy.get()` to select the logo element. You can use various selectors such as class, ID, or other attributes specific to the logo element.

**3. Verify Logo Presence:** Use Cypress assertions to verify that the logo element exists on the page. You can use `should('exist')` to ensure the logo is present in the DOM.

**4. Verify Logo Attributes:** If the logo is an image (`<img>` tag), you can validate attributes like `src`, `alt`, or `title`. Ensure that the `src` attribute points to the correct image file and the `alt` attribute provides descriptive text for accessibility.

**5. Validate Logo Appearance:** If the logo is a background image or styled element, you can validate its appearance using CSS properties like `background-image`, `background-size`, or `background-position`. You may need to use custom Cypress commands or plugins to retrieve computed styles for validation.

**6. Check Logo Dimensions:** Optionally, you can validate the dimensions of the logo image to ensure it displays correctly. Use Cypress commands to retrieve the width and height attributes of the logo element and assert their values.

**7. Validate Logo Behaviour:** Test interactions with the logo, such as clicking on it to ensure it functions as expected. Use Cypress commands to simulate user actions and assertions to verify the resulting behaviour.

Here's an example code snippet demonstrating how you can find and validate a logo in Cypress:

describe('Logo Validation', () => {

it('should display the logo with correct attributes', () => {

cy.visit('/'); // Visit the webpage containing the logo

// Select the logo element using a specific selector

cy.get('.logo-img')

.should('exist') // Ensure the logo element exists

.should('have.attr', 'src', '/path/to/logo.png') // Validate the src attribute

.should('have.attr', 'alt', 'Logo') // Validate the alt attribute for accessibility

.and('have.css', 'width', '100px') // Validate width using CSS

.and('have.css', 'height', '50px'); // Validate height using CSS

});

});

Adjust the selectors and attributes according to your specific webpage structure and logo implementation. This test ensures that the logo is present, has correct attributes, and displays as expected, providing confidence in its functionality and appearance.

1. Questions about status code (204,404 etc)

**200 OK:** Indicates that the request has succeeded.

**201 Created:** Indicates that the request has been fulfilled and a new resource has been created.

**204 No Content:** Indicates that the server successfully processed the request but there is no content to return.

**400 Bad Request:** Indicates that the server cannot process the request due to client error.

**401 Unauthorized:** Indicates that the request requires user authentication.

**403 Forbidden:** Indicates that the server understood the request but refuses to authorize it.

**404 Not Found:** Indicates that the requested resource could not be found on the server.

**405 Method Not Allowed:** Indicates that the method specified in the request is not allowed for the resource.

**500 Internal Server Error:** Indicates that the server encountered an unexpected condition that prevented it from fulfilling the request.

**503 Service Unavailable:** Indicates that the server is currently unable to handle the request due to temporary overloading or maintenance of the server.

1. What are the different parameters of URL?

The parameters of a URL include:

1. Protocol: Defines the communication protocol (e.g., HTTP, HTTPS).

**2. Domain:** Identifies the server hosting the resource. https://www.example.com, www.example.com is the domain.

**3. Port:** Optional; specifies the communication endpoint on the server.

**4.Path:** Specifies the location of the resource on the server. https://www.example.com/products/index.html, /products/index.html is the path.

**5. Query Parameters:** Additional data passed to the server as key-value pairs. For example, in the URL https://www.example.com/search?q=cypress&category=testing, ?q=cypress&category=testing represents the query parameters.

**6. Fragment Identifier:** Identifies a specific section within the resource.

https://www.example.com/page#section1, #section1 is the fragment identifier.

**7. Username and Password:** Optional credentials for authentication.

For example, in the URL https://username:password@www.example.com, username and password are the credentials.

1. Types of URL(Query and Path)

There are two main types of URLs:

Query URLs: Contain query parameters in the form of key-value pairs appended to the base URL. URL: https://www.example.com/search?q=cypress&category=testing

Path URLs: Consist of a path component specifying the location of the resource on the server, following the domain. URL: https://www.example.com/products/index.html

1. How we can find the type of URL(Is it Path type or Query type)

To determine the type of URL:

**1. Query URL:** Contains a question mark (`?`) with query parameters.

**2. Path URL:** No question mark (`?`); consists of a path component following the domain.

Ex

function getUrlType(url) {

if (url.includes('?')) {

return 'Query URL';

} else {

return 'Path URL';

}

}

// Example usage:

const url1 = 'https://www.example.com/search?q=cypress&category=testing';

const url2 = 'https://www.example.com/products/index.html';

console.log(getUrlType(url1)); // Output: Query URL

console.log(getUrlType(url2)); // Output: Path URL

You can identify the type by checking for the presence of a question mark in the URL.

1. Difference between Put and patch

PUT completely updates or replaces the entire resource with the new data sent by the client. It expects the client to send all the information, even if only a small part has changed. It always has the same result for identical requests. PATCH updates just the part of the resource that has changed, allowing the client to send only the modified data. It might not always produce the same result for identical requests.

1. What are the parameters in response? (Status code,body etc)

In an HTTP response, there are several parameters that provide information about the outcome of the request and the content of the response. Some key parameters include:

**1. Status Code:** Indicates the outcome of the request, such as success, redirection, client error, or server error. Examples include 200 (OK), 404 (Not Found), and 500 (Internal Server Error).

**2. Headers:** Additional metadata sent with the response, including information about the server, content type, caching directives, and cookies.

**3. Body:** Contains the actual content of the response, which can be HTML, JSON, XML, text, or binary data, depending on the request and the server's configuration.

**4. Content-Length:** Specifies the size of the response body in bytes.

**5. Content-Type:** Indicates the type of content in the response body, such as text/html, application/json, or image/jpeg.

**6. Cookies:** Set by the server to store session information or other data on the client side.

These parameters provide crucial information about the response, allowing the client to interpret and process the data accordingly.

1. What is role of integration(e2e) folder in cypress?

The "e2e" folder in Cypress is where you store your end-to-end test scripts. It helps organize and isolate these tests, making it easier to manage and execute them separately from other types of tests. Tests within this folder simulate real user interactions with the application, covering the entire flow from start to finish.

1. What is default waiting time of cypress?

The default waiting time (timeout) in Cypress is 4 seconds. This means that Cypress will wait up to 4 seconds for various commands and assertions to complete before considering them as failed due to timeout. However, Cypress dynamically adjusts this timeout based on the behaviour of your application and the commands being executed

1. How we can override the default waiting time of cypress?

You can override the default waiting time (timeout) in Cypress by specifying a custom timeout value globally in your cypress.json configuration file using the "defaultCommandTimeout" option. For example:

1. If we write cy.log() outside the describe block what will happen?

In short, if you write `cy.log()` outside of a `describe` block in Cypress:

- It will execute immediately during test execution.

- The message passed to `cy.log()` will be logged to the Cypress command log.

- It won't be associated with a specific test case and won't affect the test outcome or assertions.

1. Expect assertion belong from which library?

The expect assertion typically belongs to the Chai assertion library in JavaScript. Chai is a popular assertion library that provides various styles of assertions, including expect, should, and assert.

1. Logical questions

(1). Count of total words in string,

(2). Fetching first 2 highest numbers in array

**CODE**

1. Which is most suitable language for cypress javascript or typescript?

In short, both JavaScript and TypeScript are suitable for Cypress. JavaScript offers ease of learning and flexibility, while TypeScript provides static typing for improved code safety and maintainability. The choice depends on your project's requirements and your team's preferences.

1. Headless mode

To run Cypress tests in headless mode, you can use the `cypress run` command with the `--headless` flag. Here's how:

npx cypress run --headless

This command will execute your Cypress tests in headless mode, meaning the tests will run in the background without opening a visible browser window. Headless mode is useful for running tests in environments without a graphical user interface, such as in a Continuous Integration (CI) pipeline.

Additionally, you can specify other options along with `--headless` to customize the test run, such as specifying the browser or providing environment variables. For example:

npx cypress run --headless --browser chrome

This command runs tests in headless mode using the Chrome browser. You can replace `chrome` with `firefox` or `electron` depending on the browser you want to use.

Make sure to navigate to your project directory containing the Cypress configuration file (`cypress.json`) before running the `cypress run` command.

1. Config file json

A configuration file is a file that contains settings and options for a particular application or system. In the context of Cypress, the configuration file (`cypress.json`) is used to specify various settings and preferences for running Cypress tests.

A JSON file, on the other hand, is a file format used for storing and transmitting data in a structured format. JSON (JavaScript Object Notation) files use key-value pairs and data types like strings, numbers, arrays, and objects to represent data.

In the case of Cypress, the configuration file (`cypress.json`) is indeed a JSON file. It follows the JSON format and contains settings and options represented as key-value pairs. You can edit this file using any text editor to customize Cypress settings such as the base URL, timeout values, test file patterns, and more.

In summary, while both a configuration file and a JSON file can be used to store settings and data, in the case of Cypress, the configuration file (`cypress.json`) is specifically a JSON file used to configure Cypress test runs.

1. What is Ci/CD pipeline

In simple terms, a CI/CD pipeline is like a conveyor belt for your software code. It helps automate the process of building, testing, and deploying your code changes so that you can deliver software updates quickly and reliably.

**Here's how it works:**

**1. Continuous Integration (CI):** Whenever a developer makes changes to the code and pushes it to a shared repository like Git, the CI pipeline automatically kicks in. It builds the code, runs tests to make sure everything works as expected, and alerts the team if there are any issues.

**2. Continuous Deployment (CD):** Once the code passes all tests in the CI phase, the CD pipeline takes over. It packages the code into a deployable format and automatically deploys it to different environments, like development, testing, and production. This ensures that the changes are safely rolled out to users without causing disruptions.

By using CI/CD pipelines, teams can release software updates more frequently, with fewer errors, and with less manual effort. It helps teams work faster, collaborate better, and deliver better software to their users.