Key Components

Cypress Test Runner:

Runs in the Browser: The Cypress Test Runner runs in the same browser context as the application being tested. This allows for real-time reloading and interaction with the application as if a user were interacting with it.
Interactive Interface: The Test Runner provides a visual interface for running tests, viewing logs, inspecting elements, and debugging.

Node.js Server:

Backend for Cypress: Cypress uses a Node.js server to handle various tasks such as serving the test files, managing network requests, and controlling the browser.
File System Access: The Node.js server allows Cypress to read from and write to the file system, enabling features like screenshots and videos.

Cypress Test Runner Core:

Command Queue: Commands in Cypress are queued and executed sequentially. Each command is sent to the browser, which then executes it and sends back the results.
Automatic Waiting: Cypress automatically waits for commands and assertions to complete, eliminating the need for manual waits or sleeps.

Browser Proxy:

Network Interception: Cypress acts as a proxy server to intercept and modify network requests. This allows for features like stubbing/mocking, request logging, and modifying responses.
Control over Network Layer: By controlling the network layer, Cypress can simulate different scenarios and ensure tests are isolated from external dependencies.

Test Execution:

Same-Loop Execution: Tests are executed in the same event loop as the application. This means there is no context switching, resulting in faster and more reliable tests.
Consistent State: Each test is run in a clean state, ensuring no side effects from previous tests.

Architectural Flow

Test Definition:

Tests are written in JavaScript (or TypeScript) using Cypress's API. These test files are loaded by the Cypress Test Runner.

Test Initialization:

When tests are executed, the Cypress Test Runner initializes the browser instance and loads the application under test.

Command Execution:

Test commands are added to a command queue. The Cypress Test Runner sends these commands to the browser for execution.
Each command is executed in the browser context, and the results are sent back to the Test Runner.

Network Requests:

The browser proxy intercepts all network requests made by the application. This allows Cypress to modify, stub, or log these requests.

Assertions and Reporting:

As commands are executed, Cypress automatically waits for elements to appear and actions to complete.
Assertions are made to verify the state of the application. Results, including screenshots and videos, are logged and reported.
Benefits of Cypress Architecture

Speed and Efficiency: Running tests in the same context as the application results in faster and more efficient test execution.
Real-Time Feedback: The interactive Test Runner provides immediate feedback, making it easier to debug and develop tests.
Automatic Waiting: Cypress’s automatic waiting for commands and assertions reduces the need for manual synchronization.
Network Control: The ability to intercept and modify network requests enables advanced testing scenarios and isolation of tests from external dependencies.
Consistent Testing Environment: Running each test in a clean state ensures consistency and reliability.