# Kiến trúc của Jest testing framework

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

1. Test Runner (Trung tâm điều khiển):
   * Khởi động quá trình kiểm thử.
   * Tương tác với tất cả các thành phần khác để điều phối quá trình kiểm thử.
   * Gọi Assertion Library để thực hiện các kiểm tra.
   * Yêu cầu Mocking System tạo các đối tượng giả lập khi cần.
2. Assertion Library:
   * Nhận yêu cầu từ Test Runner để thực hiện các kiểm tra.
   * Trả về kết quả kiểm tra cho Test Runner.
3. Mocking System:
   * Được gọi bởi Test Runner khi cần tạo đối tượng giả lập.
   * Cung cấp các đối tượng giả cho các bài kiểm thử.
4. Code Coverage:
   * Theo dõi quá trình thực thi của mã nguồn trong quá trình kiểm thử.
   * Báo cáo kết quả độ bao phủ cho Test Runner.
5. Configuration System:
   * Cung cấp cấu hình cho Test Runner và các thành phần khác.
   * Ảnh hưởng đến cách các thành phần khác hoạt động.
6. Snapshot Testing:
   * Được gọi bởi Test Runner khi cần thực hiện kiểm thử snapshot.
   * So sánh kết quả hiện tại với snapshot đã lưu trữ và báo cáo lại cho Test Runner.

Luồng tương tác điển hình có thể như sau:

1. Configuration System cung cấp cấu hình cho toàn bộ quá trình.
2. Test Runner bắt đầu thực thi các bài kiểm thử.
3. Khi cần, Test Runner gọi Mocking System để tạo các đối tượng giả lập.
4. Test Runner sử dụng Assertion Library để thực hiện các kiểm tra.
5. Code Coverage theo dõi quá trình thực thi.
6. Nếu cần, Snapshot Testing được gọi để so sánh kết quả với snapshot.
7. Test Runner tổng hợp kết quả từ tất cả các thành phần và báo cáo kết quả cuối cùng.

# Các đặc trưng:

Jest là một **framework kiểm thử JavaScript** mạnh mẽ và phổ biến, đặc biệt hữu ích cho các dự án React, nhưng cũng có thể áp dụng cho nhiều ứng dụng JavaScript khác. Dưới đây là các đặc trưng nổi bật của Jest:

**1. Dễ sử dụng**

* Jest được thiết kế để **cấu hình tối thiểu** hoặc không cần cấu hình, đặc biệt khi sử dụng với các dự án React.
* Tích hợp tốt với **Create React App**, giúp lập trình viên có thể nhanh chóng bắt đầu kiểm thử mà không cần cài đặt nhiều công cụ.

**2. Test Runner (Bộ chạy kiểm thử)**

* Jest là một **test runner** mạnh mẽ, giúp bạn chạy các bài kiểm thử một cách song song để tối ưu hiệu suất.
* Nó tự động phát hiện và chạy các tệp kiểm thử có phần mở rộng như .test.js hoặc .spec.js.

**3. Assertion Library (Thư viện khẳng định tích hợp)**

* Jest tích hợp sẵn một **thư viện khẳng định** mạnh mẽ thông qua hàm expect, giúp bạn có thể viết các kiểm thử như:

javascript

Copy

expect(value).toBe(expectedValue);

expect(array).toContain(item);

* Điều này giúp giảm thiểu sự phụ thuộc vào các thư viện bên ngoài và giảm độ phức tạp cho các dự án.

**4. Mocking (Giả lập)**

* Jest cung cấp các công cụ **mocking** tích hợp để giả lập các mô-đun, hàm, hoặc đối tượng:
  + **Mock Function**: jest.fn() giúp tạo ra các hàm giả lập (mock function) để theo dõi các cuộc gọi hoặc trả về giá trị mong muốn.
  + **Manual Mocks**: Jest cho phép bạn tạo các **mô-đun giả lập thủ công** bằng cách đặt các tệp giả lập vào thư mục \_\_mocks\_\_ hoặc sử dụng jest.mock() để giả lập mô-đun.
  + **Auto Mocking**: Jest có thể tự động giả lập các mô-đun mà không cần cấu hình thủ công.

**5. Snapshot Testing (Kiểm thử ảnh chụp nhanh)**

* **Snapshot testing** là tính năng nổi bật của Jest, đặc biệt hữu ích khi kiểm thử giao diện người dùng (UI):
  + Bằng cách sử dụng expect(component).toMatchSnapshot(), Jest sẽ tạo ra một "ảnh chụp" trạng thái hiện tại của UI hoặc kết quả hàm.
  + Các ảnh chụp được lưu vào file .snap, và Jest sẽ so sánh kết quả hiện tại với ảnh chụp trước đó để phát hiện sự khác biệt.
* Điều này giúp dễ dàng phát hiện những thay đổi không mong muốn trong giao diện hoặc logic của ứng dụng.

**6. Code Coverage (Đo độ bao phủ mã)**

* Jest tích hợp sẵn công cụ đo **code coverage** và có thể tạo báo cáo chi tiết về mức độ bao phủ của mã nguồn:
  + Nó cung cấp các chỉ số như **line coverage** (bao nhiêu dòng lệnh đã được kiểm thử), **function coverage** (bao nhiêu hàm đã được kiểm thử), và **branch coverage** (bao nhiêu nhánh điều kiện đã được kiểm thử).
  + Báo cáo có thể được xuất ra dưới nhiều định dạng như HTML, JSON, hoặc xem trực tiếp trong terminal.
  + Để kích hoạt code coverage, bạn chỉ cần thêm cờ --coverage khi chạy Jest.

**7. Watch Mode (Chế độ giám sát)**

* Jest có **chế độ giám sát** (watch mode), tự động chạy lại các bài kiểm thử khi phát hiện bất kỳ thay đổi nào trong mã nguồn:
  + Điều này rất hữu ích trong quá trình phát triển, giúp lập trình viên kiểm thử liên tục mà không cần phải chạy lại thủ công.
  + Nó còn hỗ trợ các tính năng như **chỉ kiểm thử những tệp thay đổi** hoặc **chỉ kiểm thử các phần bị ảnh hưởng**.

**8. Parallel Test Execution (Thực thi kiểm thử song song)**

* Jest tự động chia các bài kiểm thử thành các nhóm và chạy chúng song song trên nhiều **worker processes**. Điều này giúp tối ưu hóa thời gian chạy kiểm thử, đặc biệt là khi có nhiều bài kiểm thử.
* Jest sử dụng Node.js **worker threads** để thực hiện điều này, giúp tận dụng toàn bộ tài nguyên CPU.

**9. Isolation & Sandboxing (Cách ly và môi trường thử nghiệm)**

* Các bài kiểm thử trong Jest được chạy trong môi trường **cách ly** để đảm bảo rằng chúng không ảnh hưởng tới nhau.
* Jest sử dụng **sandboxing** để ngăn chặn việc chia sẻ trạng thái giữa các bài kiểm thử, đảm bảo tính độc lập và độ tin cậy của kết quả.

**10. Môi trường giả lập trình duyệt (JSDOM)**

* Jest tích hợp với **JSDOM**, một môi trường DOM giả lập, giúp mô phỏng các tính năng của trình duyệt mà không cần phải chạy trong một trình duyệt thật.
* Điều này đặc biệt hữu ích cho việc kiểm thử các ứng dụng React hoặc các ứng dụng web khác, cho phép bạn kiểm thử tương tác với DOM mà không cần phụ thuộc vào môi trường trình duyệt thật.

**11. Tích hợp tốt với TypeScript và Babel**

* Jest hỗ trợ **TypeScript** và **Babel** một cách liền mạch. Điều này giúp bạn có thể kiểm thử các ứng dụng viết bằng TypeScript hoặc sử dụng các tính năng JavaScript mới mà không cần cài đặt thêm quá nhiều cấu hình.
* Bạn có thể dễ dàng cấu hình Jest để làm việc với TypeScript thông qua các transformer như ts-jest.

**12. Tích hợp với Continuous Integration (CI)**

* Jest rất dễ tích hợp vào các hệ thống **CI/CD** như **Jenkins**, **Travis CI**, **CircleCI**, và các công cụ tương tự.
* Bạn có thể dễ dàng tự động hóa việc chạy kiểm thử trong các pipeline CI để đảm bảo rằng mã nguồn luôn đạt chất lượng trước khi được triển khai.

**13. Tính năng định tuyến thông minh (Smart Test Selection)**

* Jest có khả năng **lựa chọn kiểm thử thông minh**: chỉ chạy những bài kiểm thử liên quan đến những phần mã đã thay đổi, thay vì chạy lại toàn bộ các bài kiểm thử.
* Tính năng này giúp tiết kiệm thời gian khi làm việc trên các dự án lớn.

**14. Cấu hình dễ dàng và linh hoạt**

* Jest cho phép cấu hình thông qua tệp jest.config.js, hoặc bạn có thể đặt cấu hình trực tiếp trong tệp package.json. Một số tùy chọn cấu hình bao gồm:
  + Tệp hoặc thư mục nào sẽ được kiểm thử.
  + Các mô-đun hoặc tệp nào sẽ bị bỏ qua.
  + Cấu hình môi trường kiểm thử (Node.js, JSDOM, v.v.).

**15. Plugin và Extensions**

* Jest có thể mở rộng tính năng thông qua các **plugin** và **extensions**. Bạn có thể tích hợp Jest với các công cụ khác như **Enzyme** (dành cho React), hoặc các công cụ kiểm thử khác.
* Nhiều plugin bên thứ ba cũng có sẵn để mở rộng khả năng của Jest.