ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

ĐINH THỊ LOAN

TÌM HIỂU VÀ XÂY DỰNG CÔNG CỤ HỖ TRỢ KIỂM THỬ CÁC HỆ THỐNG HƯỚNG DỊCH VỤ

LUẬN VĂN THẠC SĨ NGÀNH CÔNG NGHỆ THÔNG TIN

Hà Nội – 2018

ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

ĐINH THỊ LOAN

TÌM HIỂU VÀ XÂY DỰNG CÔNG CỤ HỖ TRỢ KIỂM THỬ CÁC HỆ THỐNG HƯỚNG DỊCH VỤ

Ngành: Công nghệ thông tin
Chuyên ngành: Kỹ thuật phần mềm
Mã số: 60480103

LUẬN VĂN THẠC SĨ KỸ THUẬT PHẦN MỀM
Người hướng dẫn khoa học: TS. VÕ ĐÌNH HIẾU

Hà Nội - 2018

LỜI CAM ĐOAN

Tôi xin cam đoan luận văn được hoàn thành trên cơ sở nghiên cứu, tổng hợp và thực nghiệm về bài toán phát triển và kiểm thử hệ thống ESB trong việc nâng cao chất lượng sản phẩm, hệ thống ứng dụng dịch vụ trong ngân hàng.

Luận văn này là mới, các đề xuất trong luận văn do chính tôi thực hiện, qua quá trình nghiên cứu đưa ra và không sao chép nguyên bản từ bất kỳ một nguồn tài liệu nào khác.

LỜI CẢM ƠN

Lời đầu tiên tôi xin gửi lời cảm ơn chân thành và biết ơn sâu sắc tới TS. Võ Đình Hiếu, người thầy đã chỉ bảo và hướng dẫn tận tình cho tôi trong suốt quá trình học thạc sĩ và trong suốt quá trình nghiên cứu và thực hiện luận văn này.

Tôi xin chân thành cảm ơn sự dạy bảo, giúp đỡ, tạo điều kiện của các thầy, cô trường Đại học Công nghệ, Đại học Quốc gia Hà Nội trong suốt quá trình tôi học tập tại trường.

Cuối cùng, tôi xin gửi lời cảm ơn chân thành tới gia đình, bạn bè, đồng nghiệp - những người luôn ở bên tôi trong lúc khó khăn, động viên, khuyến khích tôi trong cuộc sống và công việc.

Tôi xin chân thành cảm ơn!

Tác giả

Đinh Thị Loan

MỤC LỤC

DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT

STT	Từ/Cụm từ	Tên viết tắt
1	Application Programming Interface	API
2	Continuous Deployment	CD
3	Continuous Integration	CI
5	Enterprise Application Intergration	EAI
6	Enterprise resource planning	ERP
7	Enterprise Service Bus	ESB
8	Internet Banking	IB
9	Quality Assurance	QA
10	Service Oriented Architecture	SOA
11	Test Compatibility Kit	TCK
12	User Acceptance Testing	UAT
13	Distributed Version Control System	DVCS

DANH MỤC CÁC HÌNH VẼ, ĐỒ THỊ

MỞ ĐẦU

Kiến trúc hướng dịch vụ (SOA) cung cấp một cách tiếp cận linh hoạt cho kiến trúc hệ thống phần mềm cho doanh nghiệp hiện nay. Hệ thống xây dựng theo kiến trúc SOA có tính mở rộng cao và khả năng sử dụng lại tốt. Các dịch vụ trên hệ thống được công khai trên internet thông qua các giao diện API giúp cho việc kết nối các ứng dụng dễ dàng. Ngôn ngữ mô tả dịch vụ web (WSDL) và các tiêu chuẩn dịch vụ web khác như WS-policy cung cấp giao thức kết nối các ứng dụng trong hệ thống SOA với nhau. Quá trình ảo hóa các chức năng nghiệp vụ của doanh nghiệp là mục tiêu chính của kiến trúc hướng dịch vụ. Các dịch vụ có thể được triển khai trên các nền tảng công nghệ khác nhau như Java, .NET… Bên yêu cầu gửi thông điệp tới bên nhận và nhận lại phản hồi mà không cần quan tâm đến quá trình xử lý bên trong của bên nhận.

Công nghệ trục tích hợp (Enterprise Service Bus - ESB) là một loại kiến trúc phần mềm, chứa một tập các luật và nguyên tắc cho việc tích hợp nhiều ứng dụng khác nhau (về nền tảng, ngôn ngữ,...) vào một hay nhiều hệ thống. Áp dụng công nghệ trục tích hợp ESB giúp cho các thành phần trong hệ thống có tính tái sử dụng cao, chi phí cho việc phát triển và tích hợp các ứng dụng ngoài hay ứng dụng của bên thứ ba thấp.

Tuy nhiên, tích hợp nhiều ứng dụng khác nhau trên cùng một hệ thống cũng làm cho quá trình kiểm thử trở nên khó khăn, phức tạp hơn và yêu cầu cũng trở nên khắt khe hơn. Kiến trúc ESB có thể kết nối nhiều ứng dụng với nhau, kể cả ứng dụng trong và ngoài doanh nghiệp, vì vậy, quá trình kiểm thử hệ thống phải xem xét bao quát nhiều yếu tố: các nhà cung cấp dịch vụ, các thành phần dịch vụ, người dùng dịch vụ, giao tiếp giữa các thành phần.

Quá trình kiểm thử hệ thống sử dụng kiến trúc ESB tập trung vào giao tiếp giữa các thành phần và các tính năng có sự trao đổi tích hợp thông tin, hay nói cách khác là các API. Vì vậy, phần kiểm thử trên giao diện người dùng không cần tập trung chú trọng. Ngoài ra, quá trình kiểm thử cần được thực hiện song song, tự động hóa với quá trình phát triển, khi tích hợp một thành phần mới vào hệ thống, giúp rút ngắn thời gian cũng như tiết kiệm chi phí.

Hiện nay, quá trình kiểm thử các hệ thống sử dụng kiến trúc ESB gặp phải những khó khăn về xây dựng môi trường kiểm thử, sức ép về thời gian phát triển ngắn, các công cụ hỗ trợ chưa nhiều hoặc phải mất phí. Việc này dẫn tới quy trình kiểm thử chưa được tự động hóa, quy trình bị rút ngắn hoặc bỏ qua, khi xảy ra lỗi tại một ứng dụng trong hệ thống sẽ đòi hỏi việc tìm lỗi và sửa đổi nhiều ứng dụng cùng lúc, gây mất thời gian và tốn kém tài nguyên, các lỗi không được kiểm soát chặt chẽ.

Vì vậy, luận văn này nghiên cứu, tìm hiểu, đưa ra đề xuất về quy trình kiểm thử tự động ứng dụng ESB xây dựng trên nền tảng MuleESB áp dụng quy trình tích hợp liên tục và chuyển giao liên tục. Đồng thời luận văn cũng đưa ra công cụ hỗ trợ cho quy trình, giải quyết vấn đề tự động hóa sinh ra các ca kiểm thử, giúp rút ngắn thời gian quy trình kiểm thử.

Ngoài phần mở đầu và kết luận, luận văn được tổ chức thành các chương như sau. Chương 1 khái quát khái niệm SOA, ESB. Lợi ích của việc sử dụng kiến trúc ESB trong việc phát triển ứng dụng doanh nghiệp. Một số khái niệm liên quan đến việc kiểm thử ứng dụng. Chương 2 đưa ra thực trạng, khó khăn của việc kiểm thử trên hệ thống ESB, phân tích các vấn đề cần giải quyết. Chiến lược kiểm thử, đề xuất giải pháp, quy trình kiểm thử hệ thống và trình bày về công cụ tự phát triển hỗ trợ quy trình được trình bày ở chương 3. Chương 4 đưa ra các bước áp dụng thực tế của quy trình với một ứng dụng đơn giản xây dựng dựa trên MuleESB. Chương tổng kết tóm tắt kết quả đạt được, các điểm hạn chế và định hướng phát triển trong tương lai.

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT VÀ CÁC KHÁI NIỆM LIÊN QUAN

Ngày nay, việc phát triển phần mềm càng trở nên phức tạp và khó kiểm soát do sự xuất hiện của nhiều công nghệ mới tạo nên môi trường phát triển và nền tảng không đồng nhất, trong khi nhu cầu trao đổi, chia sẻ và tương tác giữa các ứng dụng ngày càng tăng. Trong những năm gần đây, việc phát triển hệ thống phần mềm đang dần chuyển sang xu thế hướng dịch vụ (Service-oriented Architecture). Trong đó, ESB là kiến trúc được ứng dụng mang lại nhiều lợi ích cho doanh nghiệp. Kiến trúc ESB có khả năng kết nối nhiều thành phần trên nhiều nền tảng khác nhau, hỗ trợ việc trao đổi thông tin qua lại trong hệ thống. MuleSoft là một trong những công ty cung cấp giải pháp ESB đầu tiên với sản phẩm là nền tảng MuleESB được đánh giá rất cao. Tuy nhiên vấn đề mới đặt ra là cần đảm bảo được khả năng kiểm soát lỗi tốt song song với quá trình phát triển khi mà có càng nhiều thành phần mới được tích hợp thêm. Những kỹ thuật kiểm thử như kiểm thử hộp đen, kiểm thử hộp trắng, kiểm thử hộp xám và các cấp độ kiểm thử từ kiểm thử đơn vị đến kiểm thử chức năng, kiểm thử tích hợp, kiểm thử hồi quy là những kỹ thuật cần thiết để áp dụng trong vấn đề này. Ngoài ra các quy trình tích hợp, chuyển giao và triển khai liên tục cũng cần được áp dụng để hỗ trợ quy trình kiểm thử.

Để giúp làm rõ hơn những nội dung trong các chương tiếp theo, chương này sẽ giới thiệu các khái niệm cơ bản nêu trên một cách chi tiết hơn.

1.1. Kiến trúc hệ thống

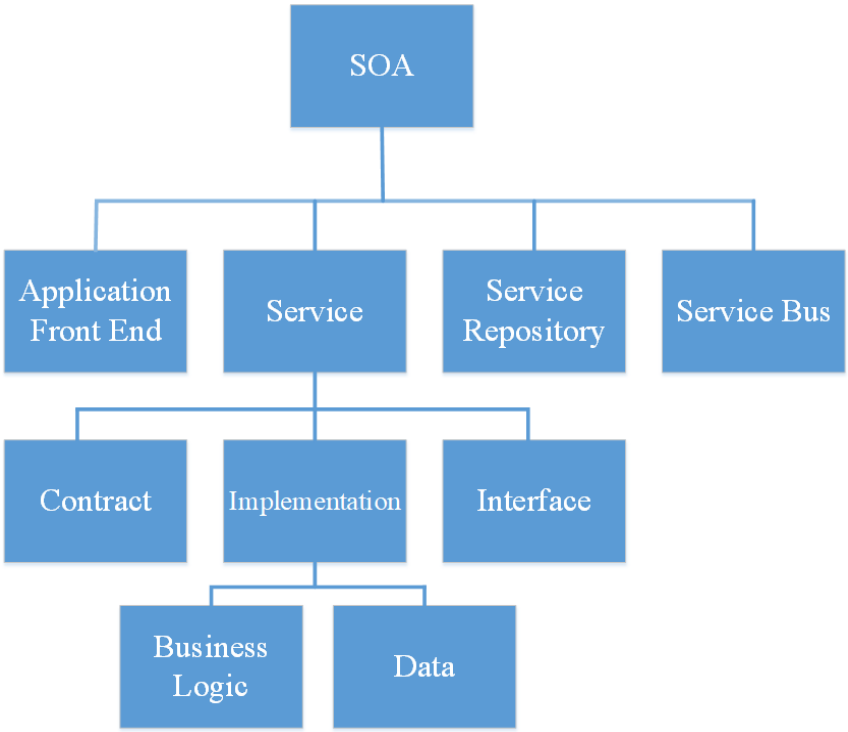
1.1.1. Kiến trúc hướng dịch vụ

Kiến trúc hướng dịch vụ (Service Oriented Architecture - SOA) [1] [2] là một chiến lược xây dựng kiến trúc phần mềm. Đây là quá trình tích hợp các thành phần độc lập kết nối với nhau một cách linh động thông qua các giao thức được định nghĩa sẵn, và tính tái sử dụng cao.

Khái niệm dịch vụ trong hệ thống SOA được hiểu là một chức năng được xác định rõ ràng, khép kín và không phụ thuộc vào ngữ cảnh hoặc trạng thái của các dịch vụ khác.

Một kiến trúc hướng dịch vụ được dựa trên 4 khái niệm trừu tượng chính: ứng dụng đầu cuối, dịch vụ, kho dịch vụ và trục tích hợp (xem Hình *1.1***Error! Reference source not found.**). Mặc dù giao diện người dùng ứng dụng gọi đến các quy trình nghiệp vụ, các dịch vụ cung cấp chức năng nghiệp vụ mà ứng dụng đầu cuối và các dịch vụ khác có thể sử dụng. Một dịch vụ bao gồm một triển khai cung cấp dữ liệu cho logic nghiệp vụ, một hợp đồng dịch vụ chỉ định chức năng, cách sử dụng và các ràng buộc cho một khách hàng của dịch vụ và một giao diện để kết nối. Kho lưu trữ dịch vụ lưu trữ các hợp đồng dịch vụ của

các dịch vụ riêng lẻ của một kiến trúc SOA và trục tích hợp dịch vụ kết nối các giao diện và dịch vụ đầu cuối [3].



Hình 1.1: Các công nghệ trong hệ thống SOA

Ứng dụng đầu cuối là lớp có chức năng kích hoạt và điều khiển mọi hoạt động của hệ thống ứng dụng doanh nghiệp. Có nhiều loại ứng dụng đầu cuối. Một ứng dụng đầu cuối cung cấp giao diện tương tác với người dùng như ứng dụng web hoặc một rich-client. Tuy nhiên, một ứng dụng đầu cuối không nhất thiết phải tương tác trực tiếp với người dùng. Các chương trình chạy theo lô (Batch programming) hay các tiến trình chạy tự động có thể gọi đến một chức năng nào đó trong hệ thống hoặc là kết quả của một sự kiện cũng được coi là ứng dụng dầu cuối.

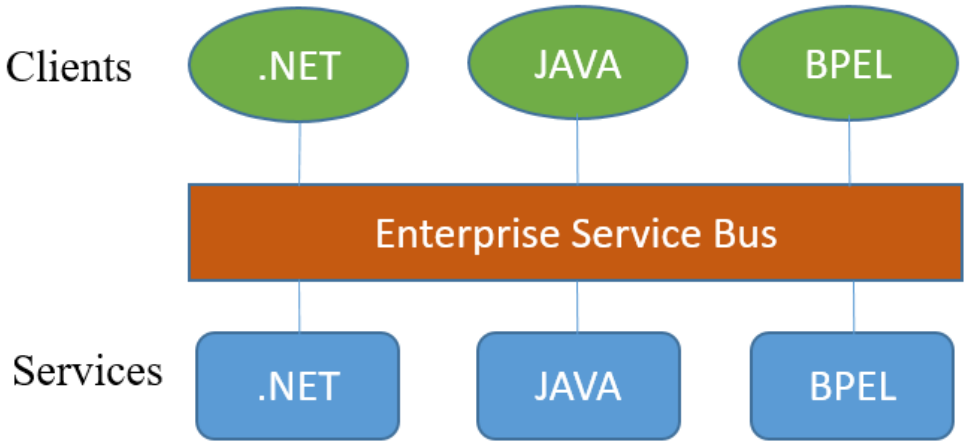
Kho chứa dịch vụ được sử dụng để tích hợp các ứng dụng của các doanh nghiệp, những doanh nghiệp này thường có yêu cầu khác nhau, các kho lưu trữu được thông khai thông qua mạng Internet. Các yêu cầu này có thể bao gồm các vấn đề pháp lý (điều khoản và điều kiện sử dụng), kiểu trình bày, bảo mật, đăng ký người dùng, đăng ký dịch vụ, thanh toán và quản lý phiên bản. SOA là cấp độ cao hơn của phát triển ứng dụng, chú trọng đến quy trình nghiệp vụ và dùng giao tiếp chuẩn để nhằm che giấu cách thức phát triển bên trong từng ứng dụng. Các thành phần được nối kết qua cổng giao tiếp, có tính kế thừa các thành phần đang tồn tại, và sự tương tác giữa chúng không cần quan tâm đến việc chúng được phát triển trên nền tảng công nghệ nào. Điều này khiến hệ thống có thể mở rộng và tích hợp một cách dễ dàng. Kiến trúc SOA có những ưu điểm như: tính tái sử dụng, tính

linh hoạt, các thành phần trong kiến trúc liên kết không chặt, ít có sự ràng buộc với nhau. Các dịch vụ trong kiến trúc SOA có tính tự trị, có quyền kiểm sóa dựa vào logic bên trong của dịch vụ đó. SOA cung cấp khả năng tương thích giữa nhiều nền tảng ngôn ngữ, tính đóng gói, các thành phần hoạt động phi trạng thái và người dùng có thể tìm kiếm, sử dụng dịch vụ theo nhu cầu.

Một trục tích hợp (Service bus) kết nối các thành phần tham gia của hệ thống SOA với nhau bao gồm dịch vụ và các ứng đụng dầu cuối. Khái niệm trục tích hợp sẽ được trình bày cụ thể trong phần 1.1.2. Công nghệ trục tích hợp.

1.1.2. Công nghệ trục tích hợp

Công nghệ trục tích hợp (Enterprise Service Bus - ESB) [4] [5] là một kiến trúc phần mềm, chứa một tập các luật và nguyên tắc cho việc tích hợp nhiều ứng dụng khác nhau về nền tảng, ngôn ngữ,... vào một hay nhiều hệ thống. Xây dựng hệ thống nền tảng trục tích hợp cho doanh nghiệp từ đầu đòi hỏi rất nhiều thời gian, công sức và tiền bạc. Hệ thống nền tảng trục tích hợp có tính tái sử dụng cao, chi phí cho việc phát triển và tích hợp các ứng dụng ngoài hay ứng dụng của bên thứ ba thấp.



Hình 1.2. Kiến trúc hệ thống nền tảng trục tích hợp

Hệ thống nền tảng trục tích hợp cung cấp khả năng gọi dịch vụ đồng bộ và không đồng bộ tạo điều kiện thuận lợi cho sự tương tác giữa các ứng dụng khác nhau. Việc xử lý và chuyển đổi thông tin hoặc làm giàu thêm thông tin được hiện bên trong lớp ESB nên gần như trong suốt với các ứng dụng thành phần. Ngoài ra hệ thống nền tảng trục tích hợp còn cung cấp khả năng định tuyến phân phối các thông điệp, giúp theo dõi, kiểm soát thông điệp, thiết lập các luồng thông điệp hoặc sự kiện mới. ESB cũng hỗ trợ nhiều loại hình tương tác: Request/response, Request/multi-response, Event propagation,...

Như vậy khi hệ thống được xây dựng với kiến trúc trục tích hợp sẽ có khả năng phân phối thông tin cho toàn bộ hệ thống một cách nhanh chóng và dễ dàng mặt khác vẫn ẩn đi các nền tảng phía sau của kiến trúc phần mềm và giao thức mạng. Hệ thống vẫn sẽ đảm bảo

thông tin được chuyển đi thậm chí khi vài thành phần hoặc mạng bị ngừng hoạt động, gián đoạn. Thông tin trao đổi giữa các thành phần được định tuyến, lưu vết. Quá trình triển khai cũng có thể thực hiện từng phần, không nhất thiết phải chuyển toàn bộ dịch vụ hay toàn bộ các ứng dụng trong một lần.

Mô hình trục tích hợp tránh cho bên yêu cầu không cần phải biết rõ việc bên cung cấp dịch vụ xử lý như thế nào, từ khía cạnh nhà cung cấp dịch vụ lẫn nhà phát triển. ESB sẽ chịu trách nhiệm về việc truyền/nhận và phân phối thông điệp từ nơi gửi đến nơi nhận và đảm bảo đáp ứng dược yêu cầu mà không cần biết đến nguồn gốc của thông điệp. Chính lớp ESB cũng là trong suốt đối với các ứng dụng kết nối với nó. Logic ứng dụng có thể gọi hoặc phân phối dịch vụ bằng cách sử dụng một loạt các mô hình và các kỹ thuật lập trình mà không cần phải xem xét việc kết nối đi qua lớp ESB như thế nào. Việc kết nối đến một ESB không làm thay đổi công nghệ phát triển của ứng dụng.

ESB hỗ trợ sự tăng lên nhanh chóng các dịch vụ và các ứng dụng tích hợp trong các hệ thống của tổ chức doanh nghiệp theo đòi hỏi của nghiệp vụ kinh doanh có thể vượt quá khả năng của công nghệ của các hệ thống đó. ESB giúp giảm thiểu dư thừa dữ liệu và dữ liệu không nhất quán. Việc xây dựng và phát triển nền tảng ESB được coi như đặt viên gạch đầu tiên trong quá trình xây dựng kiến trúc hướng dịch vụ (SOA).

1.1.3. Xây dựng hệ thống trục tích hợp dựa trên nền tảng MuleESB

Đa số các hệ thống hiện đại bao gồm các ứng dụng độc lập cần phải trao đổi thông tin với nhau, tạo thành hệ thống tích hợp. ESB là một hướng tiếp cận giúp cho người lập trình xử lý thông điệp truyền giữa các thành phần ứng dụng độc lập mà không cần phải tốn nhiều công sức trong việc chuyển đổi dữ liệu giúp các ứng dụng giao tiếp với nhau. Trong khi đó các giải pháp ESB bản thương mại như Oracle Service Bus, IBM Websphere Enterprise Service Bus … gây tốn kém chi phí cho doanh nghiệp, các giải pháp này còn mang tính đóng, lập trình viên không thể kiểm soát được nội dung bên trong của mã nguồn. MuleESB [6] đưa ra cách giải quyết vấn đề thường gặp ở các doanh nghiệp trong việc tích hợp, kiến trúc hệ thống.

Mule framework

Mule [7] là một trong những dự án mã nguồn mở đầu tiên cung cấp giải pháp tổng thể và đủ lớn để xây dựng nên một hệ thống ESB. Nó cung cấp một bộ đầy đủ các tính năng tích hợp cần thiết cho một doanh nghiệp.

Mule là một nền tảng tích hợp dựa trên Java, cho phép các nhà phát triển kết nối các ứng dụng với nhau một cách nhanh chóng và dễ dàng, giúp các ứng dụng trao đổi dữ liệu với nhau. Mule cho phép tích hợp các hệ thống hiện có, bất kể các công nghệ khác nhau mà các ứng dụng sử dụng, bao gồm JMS, Dịch vụ Web, JDBC, HTTP, và nhiều hơn nữa. Kiến trúc ESB có thể được triển khai ở mọi nơi, có thể tích hợp và sắp xếp các sự kiện theo thời gian thực hoặc theo lô và có kết nối chung. Mule xây dựng trên nền tảng Anypoint

Platform. Anypoint Studio là một công cụ giúp dễ dàng phát triển một ứng dụng trên nền tảng MuleESB. Được xây dựng từ nền tảng của IDE Eclipse, Anypoint Studio cho phép lập trình viên có thể kéo thả các thành phần để tạo nên các dòng điều khiển (flow), để có thể chuyển đổi dữ liệu gửi đi từ ứng dụng này sang dữ liệu nhận vào của ứng dụng kia. MuleSoft cung cấp cả giải pháp chạy tích hợp Mule server trên Anypoint Studio lẫn chạy độc lập ứng dụng trên Mule server (Standalone).

Mule nhẹ nhưng có khả năng mở rộng cao, cho phép người dùng bắt đầu từ việc kết nối một vài ứng dụng và tăng số lượng ứng dụng tham gia vào hệ thống theo thời gian. Một hệ thống theo kiến trúc ESB quản lý tất cả các tương tác giữa các ứng dụng và các thành phần một cách minh bạch, bất kể chúng tồn tại trong cùng một máy ảo hay trên Internet, và bất kể giao thức truyền tải cơ bản mà chúng sử dụng. MuleSoft là nhà cung cấp duy nhất được Gartner đánh giá là công cụ đứng đầu trong việc phát triển hệ thống tích hợp, đặc biệt là ứng dụng theo kiến trúc trục tích hợp(xem *Hình* 1.3**Error! Reference source not found.**).

MuleESB là một nền tảng mã nguồn mở xây dựng bằng ngôn ngữ Java cho phép phát triển ứng dụng trục tích hợp (ESB) là cầu nối giữa các ứng dụng với nhau, giúp cho việc giao tiếp giữa các ứng dụng nhanh chóng và dễ dàng. MuleESB cho phép tích hợp nhiều loại công nghệ khác nhau, qua các giao thức như JMS, Web Services, JDBC, HTTP,...

Việc triển khai ứng dụng phân tán trên môi trường mạng giúp cho việc kết nối giữa các ứng dụng dễ dàng, tuy nhiên lại gây ra khó khăn trong giao tiếp giữa các ứng dụng do việc khác biệt về công nghệ, nền tảng. MuleESB giải quyết vấn đề này bằng việc cung cấp một trục tích hợp có chức năng nhận và định tuyến thông điện giữa các ứng dụng với nhau.

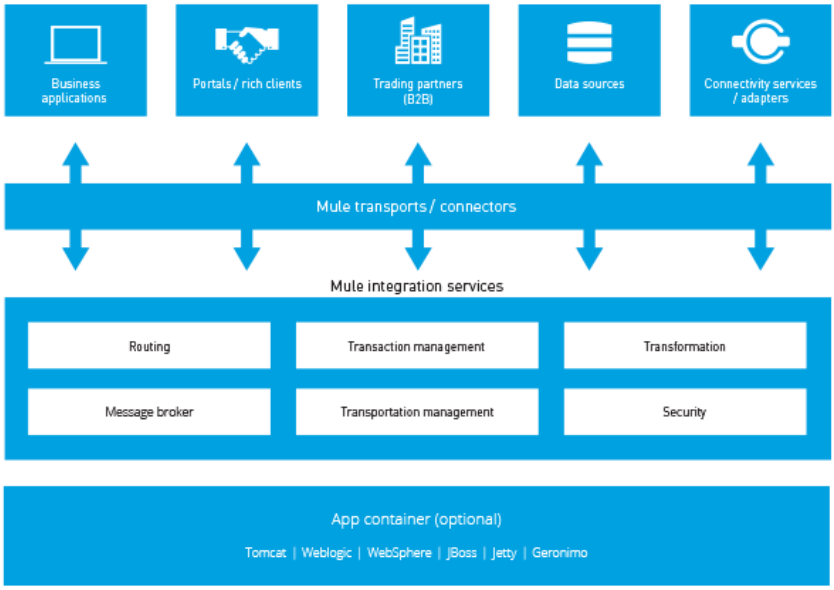
Hình 1.3: Nền tảng tích hợp cho doanh nghiệp [8]

MuleESB hoạt động như một bộ chứa các dịch vụ có khả năng tái sử dụng. MuleESB che giấu các dịch vụ khỏi định dạng thông điệp và các giao thức, tách biệt luồng nghiệp vụ với xử lý thông điệp, cho phép gọi dịch vụ ở các điểm độc lập. Ngoài ra, MuleESB cung cấp khả năng định tuyến, phân loại, sắp xếp thứ tự các thông điệp dựa trên nội dung và các quy tắc quản lý luồng nghiệp vụ cũng như chuyển đổi dữ liệu qua lại giữa các định dạng và giao thức khác nhau.

Kiến trúc MuleESB

Hình *1.4* mô tả kiến trúc của MuleESB. Trong luồng xử lý, bộ chuyển đổi (Transformer) có vai trò chuyển đổi định dạng thông điệp thành các loại định dạng phù hợp với nơi nhận thông điệp, trước khi được xử lý và định tuyến. Các bộ chuyển đổi (Transformer) là chìa khoá để trao đổi dữ liệu, dữ liệu chỉ được chuyển đổi khi cần thiết thay vì chuyển đổi thành định dạng chung, thông điệp có thể được gửi qua các kênh truyền khác nhau.

Việc tách biệt giữa luồng logic nghiệp vụ và cách thức truyền nhận dữ liệu cho phép mở rộng kiến trúc hệ thống và dễ dàng tuỳ biến luồng nghiệp vụ.



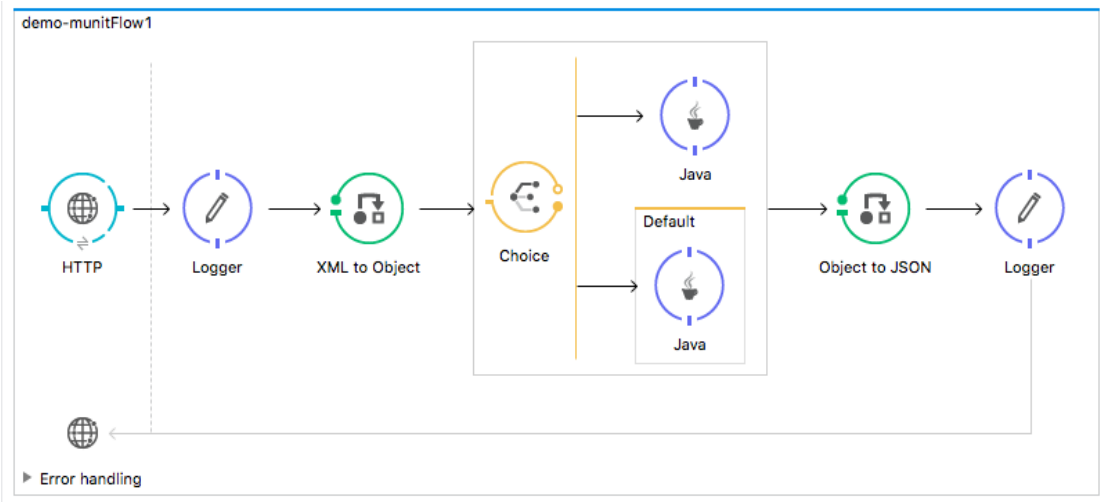
Hình 1.4: Kiến trúc MuleESB *[6]*.

Xử lý dữ liệu

Bộ chuyển đổi (Transformer): Khi một thông điệp được gửi đi giữa các ứng dụng, MuleESB tiếp nhận thông điệp, chuyển đổi định dạng thông điệp, phân loại và điều hướng sang dịch vụ nhận cần thiết.

Các thành phần (Components) chứa logic nghiệp vụ để xử lý dữ liệu bên trong thông điệp. Nó không chứa thông tin nào về cách nhận và gửi của bản thân thông điệp đó

Điều hướng thông điệp giữa các thành phần như Hình 1.5 là một ví dụ về luồng cơ bản của MuleESB. Quản lý luồng (Flow control) đảm bảo việc thông tin đúng đắn sẽ được chuyển đi đến đúng đích dựa vào các điều kiện được ghi trong thông điệp. Mule hỗ trợ nhiều loại khác nhau để quản lý việc xử lý và điều hướng thông điệp, để thêm các thành phần này, chỉ cần định nghĩa các thẻ XML trong file cấu hình của một ứng dụng Mule, Mule studio sẽ tự động tìm và xử lý nội dung theo vai trò của thành phần đó.



Hình 1.5. Mô hình luồng xử lý trên MuleESB

Ứng dụng thực tế sử dụng MuleESB

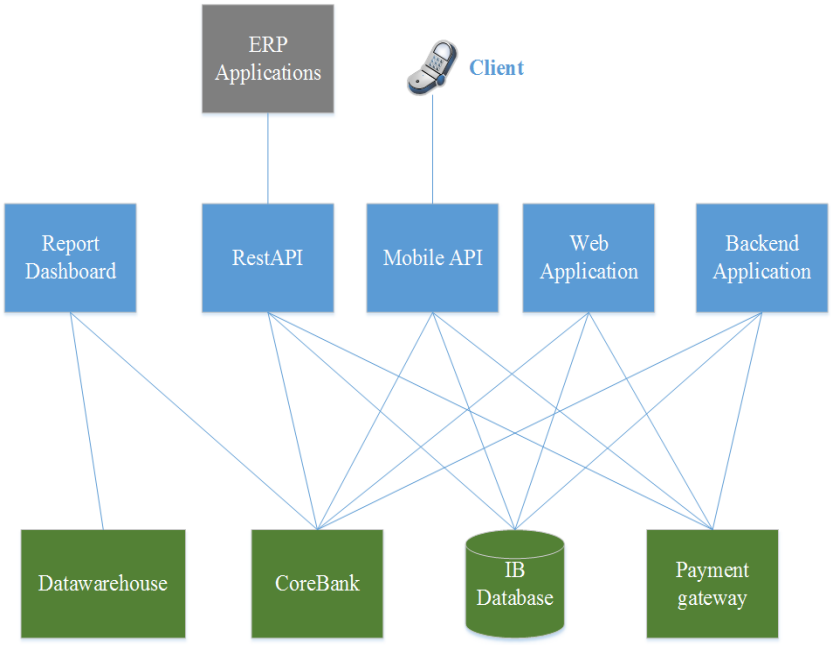
MuleESB được sử dụng rộng rãi để phát triển ứng dụng ESB, đặc biệt trong ngành tài chính, ngân hàng. Ví dụ sau đây trình bày về một hệ thống ngân hàng điện tử sử dụng MuleESB để phát triển ứng dụng ESB, giúp giảm thiểu chi phí phát triển và bảo trì, nâng cao chất lượng sản phẩm.

Internet Banking (IB) là hệ thống ngân hàng điện tử dành cho Khách hàng doanh nghiệp sử dụng các dịch vụ của VietinBank như: chuyển tiền, chi lương, thanh toán chuỗi hóa đơn, nộp ngân sách nhà nước, báo cáo...Hệ thống bao gồm các ứng dụng phía khách hàng, các ứng dụng quản trị của ngân hàng và các hệ thống lõi của ngân hàng (core banking). Các ứng dụng trong hệ thống được xây dựng trên các nền tảng khác nhau như .NET, java, .M… thậm chí có những ứng dụng xây dựng trên nền tảng công nghệ cũ như Visual Basic.

Trong hệ thống IB, ứng dụng RestAPI cung cấp các đầu dịch vụ cho các ứng dụng ERP của doanh nghiệp ngoài kết nối vào và thực hiện các giao dịch. Mobile API cung cấp cho ứng dụng chạy trên thiết bị di động, khách hàng có thể sử dụng ứng dụng trên thiết bị cầm tay để thực hiện giao dịch. Web application là nơi cung cấp các giải pháp giao dịch cho doanh nghiệp trên nền tảng web. Report Dasboard hỗ trợ báo cáo cho người dùng ngân hàng. Bankadmin màn hình quản trị hệ thống, xử lý lỗi xảy ra cho các giao dịch của khách

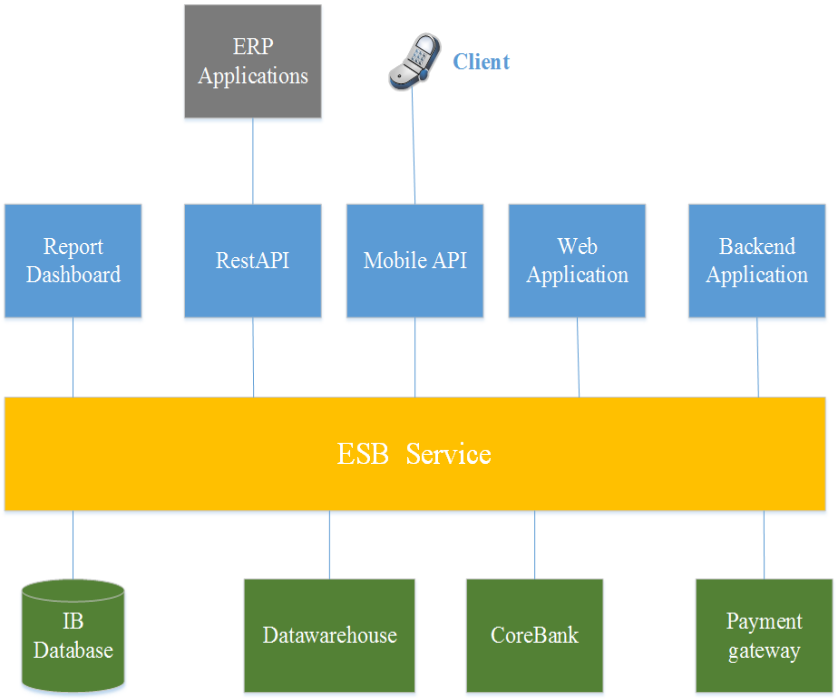
hàng gửi tới. Database là nơi lưu trữ thông tin khách hàng và giao dịch của hệ thống Internet Banking. Core bank thực hiện nghiệp vụ giao dịch, sao kê ngân hàng. Datawarehouse là kho dữ liệu phục vụ báo cáo. Payment gateway thực hiện nhiệm vụ cổng thanh toán, kết nối với các đối tác bên ngoài như nhà cung cấp dịch vụ, kho bạc nhà nước, chi cục thuế, ...

Hình 1.6 mô tả kiến trúc hệ thống Internet Banking xây dựng theo mô hình kết nối điểm-điểm (point-to-point). Với kiến trúc này, hệ thống sẽ bao gồm nhiều kết nối giữa các ứng dụng khác nhau. Việc này dẫn đến quá trình bảo trì và mở rộng hệ thống gặp nhiều khó khăn, khả năng kiểm soát lỗi kém. Ngoài ra, kết nối point-to-point đối với hệ thống này dẫn đến các quy trình nghiệp vụ của các ứng dụng trên lặp lại và chồng chéo nhau, gây tốn chi phí phát triển và bảo trì.



Hình 1.6. Kiến trúc hệ thống IB cũ

Hình 1.7 mô tả hệ thống Internet Banking sau khi phát triển sử dụng một lớp ESB thực hiện điều hướng thông điệp và xử lý kết hợp với quy trình nghiệp vụ để giảm thiểu việc phát triển chồng chéo nhiều chức năng giống nhau, đồng thời giảm thiểu số lượng các kết nối giữa các ứng dụng. Việc tích hợp hệ thống qua ứng dụng ESB giúp giảm đáng kể số lượng kết nối giữa các ứng dụng, loại bỏ phát triển trùng lặp các chức năng gần giống nhau trên các ứng dụng. Ngoài ra, việc tích hợp còn giúp tiết kiệm kiệm chi phí triển khai, bảo trì. MuleESB là một framework nhẹ, việc triển khai diễn ra tự động và nhanh chóng, nên thời gian ngắt của ứng dụng trong lúc triển khai nhỏ (dưới 60 giây), đảm bảo hệ thống chạy thông suốt thời gian dài.



Hình 1.7. Kiến trúc hệ thống IB mới

1.2. Tích hợp và triển khai liên tục

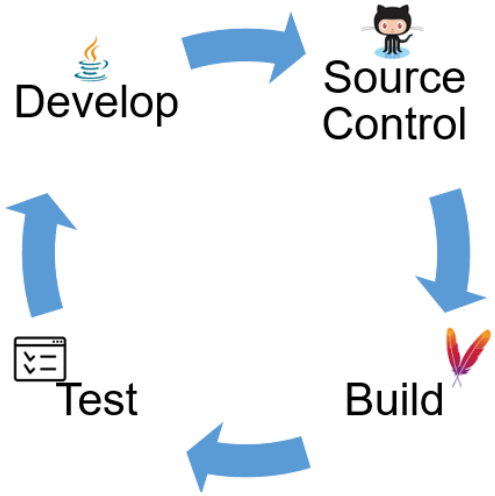
1.2.1. Tích hợp liên tục

Theo định nghĩa của Martin Fowler [9], tích hợp liên tục – Continuous Intergration là phương pháp phát triển phần mềm đòi hỏi các lập trình viên trong nhóm tích hợp ứng dụng thường xuyên. Mỗi ngày, các thành viên đều phải theo dõi và phát triển công việc của họ ít nhất một lần. Việc này sẽ được một nhóm khác kiểm tra tự động, nhóm này sẽ tiến hành kiểm thử truy hồi để phát hiện lỗi nhanh nhất có thể. Đây là phương pháp tiếp cận này giúp giảm bớt vấn đề về tích hợp hơn và cho phép phát triển phần mềm gắn kết nhanh hơn. Các nhóm phát triển sử dụng phương pháp Agile hay dùng tích hợp liên tục để đảm bảo mã nguồn của toàn dự án luôn dịch được và chạy đúng. Đây là một thực tiễn phát triển yêu cầu các lập trình viên tích hợp mã vào một kho lưu trữ được chia sẻ trong các khoảng thời gian đều đặn, giúp loại bỏ các vấn đề của việc tìm lỗi xảy ra trong pha lập trình. Thực tế cho thấy, mỗi khi có một sự thay đổi mã nguồn trên kho lưu trữ, quá trình tích hợp liên tục sẽ được kích hoạt.

Với tầm quan trọng của kiểm thử, việc sử dụng một giải pháp tích hợp liên tục - Continuous Integration (CI) trong cả quá trình phát triển là vô cùng cần thiết. CI đem lại nhiều thuận lợi trong kiểm thử như phát hiện sớm các vấn đề của hệ thống trong vòng đời phát triển, đảm bảo các ca kiểm thử được chạy hết trước khi triển khai một phiên bản mới. Hệ thống CI có khả năng tự động cập nhật và biên dịch mã nguồn, chạy các ca kiểm thử đã được định nghĩa trước cho mỗi một commit. Ngoài ra CI còn giúp cho việc đóng gói phần

mềm dễ dàng, kết hợp với các giải pháp triển khai liên tục -Continuos Delivery (CD) triển khai nhanh gọn hơn.

Quy trình của một hệ thống CI (xem Hình *1.8*) bắt đầu khi có một sự thay đổi (commit) mới được đẩy lên, hệ thống CI sẽ tự lấy mã nguồn mới về bằng lệnh gọi từ svn hoặc git, sau đó thực hiện dịch mã. Hệ thống sẽ tự động gửi email về cho các thành viên nếu như việc dịch mã bị lỗi. Bước tiếp theo khi bước biên dịch thành công, các ca kiểm thử đã được định nghĩa sẽ được chạy toàn bộ, nếu có ca kiểm thử thất bại, hệ thống sẽ gửi email thông báo cho đội phát triển. Sau khi chạy thành công các ca kiểm thử, hệ thống CI tiến hành đóng gói và triển khai ứng dụng lên máy chủ (nếu cần). Quá trình tích hợp sẽ được tiến hành theo cấu hình của đội phát triển ứng dụng.

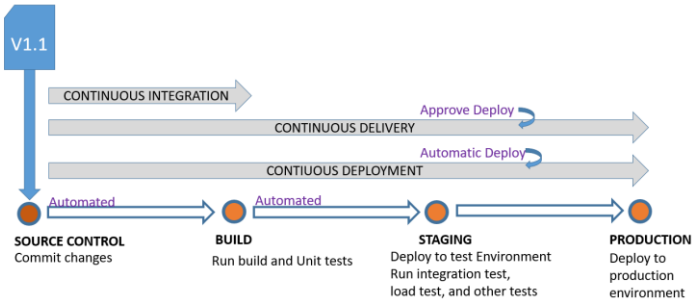


Hình 1.8: Quy trình tích hợp liên tục

Tích hợp liên tục giúp giảm thiểu rủi ro nhờ việc phát hiện lỗi sớm, tăng chất lượng phần mềm nhờ việc tự động kiểm thử và kiểm tra, đồng thời giảm thiểu những quy trình thủ công lặp đi lặp lại, thay vào đó là thực hiện tự động. Hiện tại, có nhiều công cụ hỗ trợ việc tích hợp liên tục như TFS, TeamCity, Hudson, Jenkin, Travis, ...

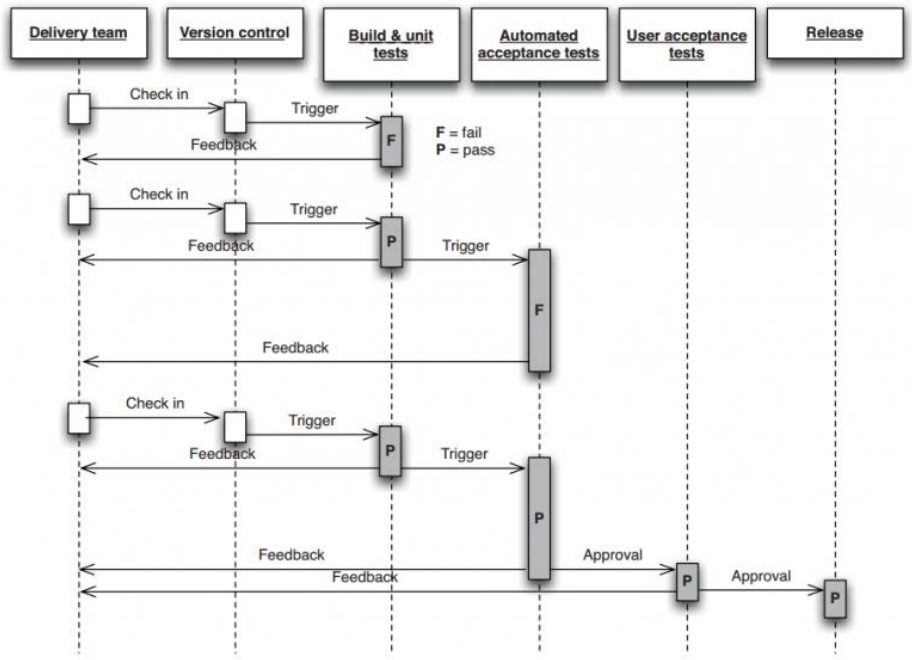
1.2.2. Chuyển giao liên tục

Trong khi tích hợp liên tục là quy trình để dịch và kiểm thử tự động, thì việc chuyển giao liên tục (Continuous Delivery) cao hơn một mức, đó là triển khai ứng dụng sau khi kiểm thử thành công lên môi trường kiểm thử hoặc staging. Chuyển giao liên tục cho phép lập trình viên tự động hóa phần kiểm thử bên cạnh việc sử dụng keierm thử đơn vị để kiểm tra phần mềm qua nhiều thước đo trước khi triển khai cho khách hàng. Những bài kiểm thử này bao gồm: kiểm thử giao diện, kiểm thử tải, kiểm thử tích hợp và kiểm thử giao diện API. Nó tự động hoàn toàn quy trình triển khai phần mềm. Hình *1.9* mô tả quá trình chuyển giao liên tục.



Hình 1.9: Quy trình chuyển giao liên tục

Chuyển giao liên tục được thực hiện bằng cách sử dùng dòng triển khai (Deployment Pipeline - Hình *1.10*). Dòng triển khai chia quy trình chuyển giao phần mềm thành các giai đoạn. Mỗi giai đoạn có mục tiêu xác minh chất lượng của các tính năng mới từ một góc độ khác nhau để kiểm định chức năng và tránh lỗi ảnh hưởng đến người dùng. Pipeline sẽ cung cấp phản hồi cho nhóm trong việc cung cấp tính năng mới. Ở góc độ trừu tượng hơn, dòng triển khai là quy trình để chuyển phần mềm từ quản lý phiên bản đến tay người dùng. Mỗi thay đổi đến phần mềm sẽ đi qua một quy trình phức tạp để được phát hành.



Hình 1.10: Dòng triển khai

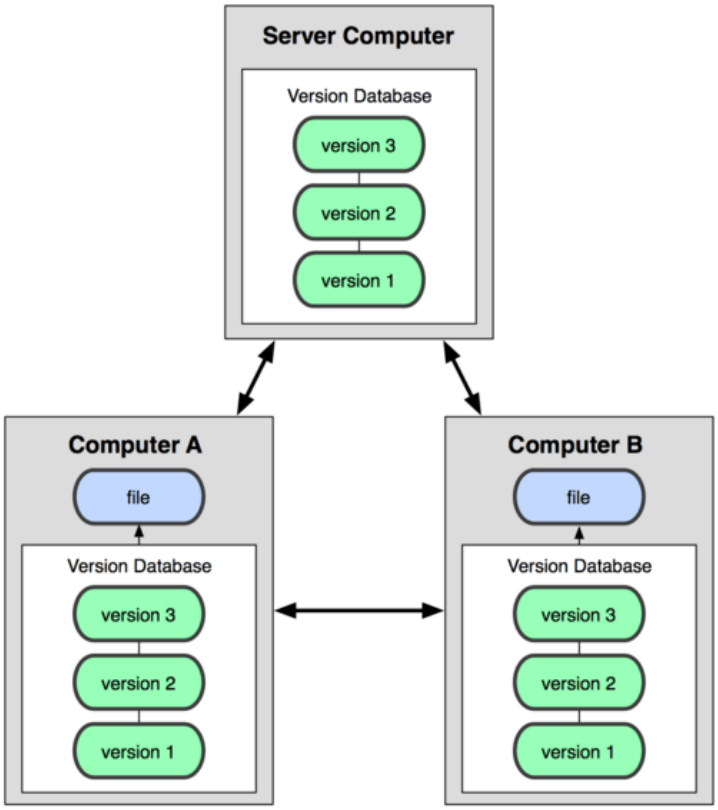
1.2.3. Một số công cụ hỗ trợ

Github

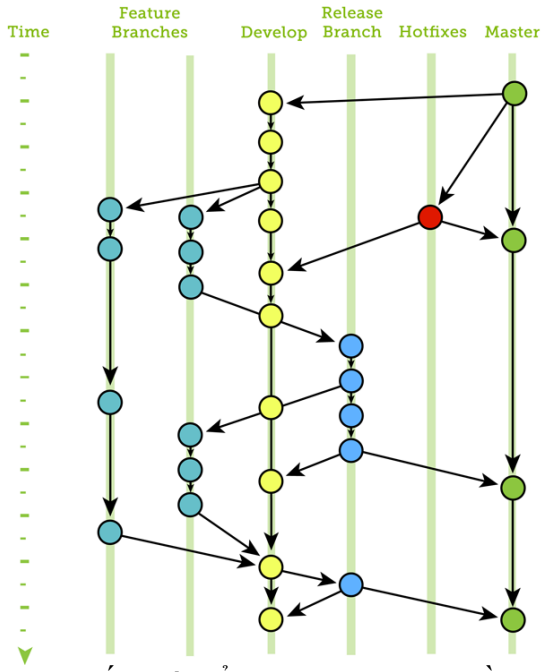
Việc quản lý mã nguồn của ứng dụng ESB được sử dụng trong luận văn áp dụng công cụ github. Git là một Hệ thống quản lý phiên bản phân tán (Distributed Version Control System). Ngoài Git, còn có một số hệ thống quản lý mã nguồn phân tán như: Mercurial,

CVS, Subversion… Trong đó Subversion và Git là một trong số những phiên bản quản lý mã nguồn phân tán phổ biến nhất hiện nay. DVCS (Distributed Version Control System) là khái niệm cơ bản của Git, đây là hệ thống giúp mỗi máy tính có thể lưu trữ nhiều phiên bản khác nhau của một mã nguồn được nhân bản (clone) từ một kho chứa mã nguồn (repository), mỗi thay đổi vào mã nguồn trên máy tính sẽ có thể ủy thác (commit) rồi đưa lên máy chủ nơi đặt kho chứa chính. Và một máy tính có quyền truy cập khác cũng có thể nhân bản lại mã nguồn từ kho chứa hoặc nhân bản lại một tập hợp các thay đổi mới nhất trên máy tính kia. Trong Git, thư mục làm việc trên máy tính gọi là Working Tree. mô tả mô hình hoạt động của DVCS. Git mang lại nhiều lợi thế trong lập trình do tính dễ sử dụng, an toàn và nhanh chóng, giúp cho quy trình phát triển phần mềm trong một nhóm hiệu quả và đơn giản hơn bằng cách phân nhánh (xem).

Github là một dịch vụ máy chủ repository công cộng, mỗi người có thể tạo tài khoản trên đó để tạo ra các kho chứa của riêng mình để có thể làm việc.

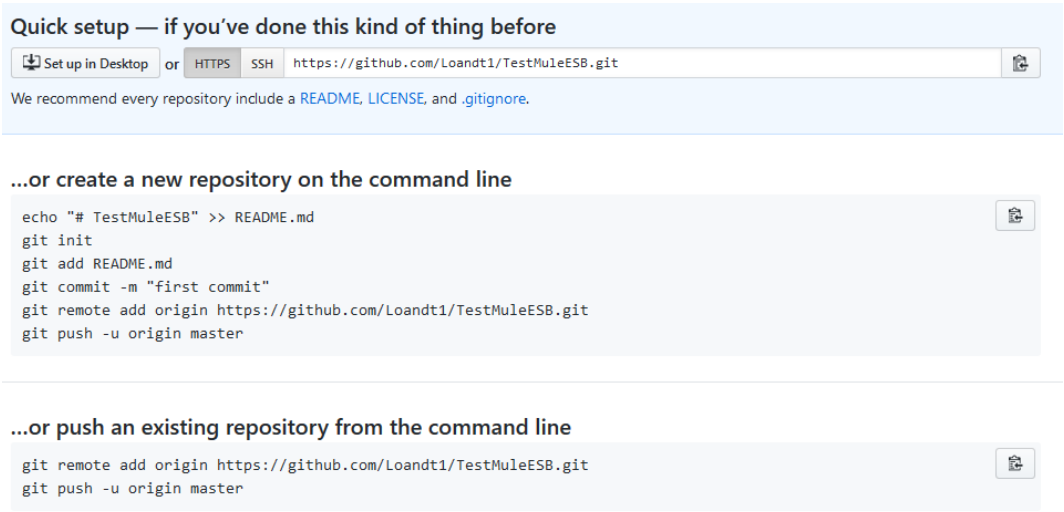


Hình 1.11: Mô hình hoạt động của DVCS *[10]*.



Hình 1.12: Cấu trúc tổ chức kho mã nguồn trên Git

Git cung cấp giao diện dòng lệnh với các lệnh cơ bản như .



Hình 1.13: Các dòng lệnh trên Git

Ngoài ra, các IDE như Eclipse còn cung cấp các plugins hỗ trợ giao diện đồ họa cho việc làm việc trên Git.

Maven

Maven là công cụ quản lý mã nguồn và thư viện phụ thuộc một cách tự động, được sử dụng cho các ứng dụng trên nền tảng Java, ngoài ra còn có các nền tảng khác như C#, Ruby, Scala… Được phát triển với mục đích tương tự như Apache Ant nhưng có khái niệm và cách hoạt động khác, Maven hỗ trợ việc tự động hóa quá trình quản lý dự án phần mềm

như: khởi tạo, biên dịch, kiểm thử, đóng gói và triển khai sản phẩm. Maven được phát triển bằng ngôn ngữ Java và chạy trên được nhiều nền tảng khác nhau như Windows, Linux, Mac OS… thể hiện cấu hình Maven để quản lý dự án phần mềm.

Khái niệm POM (Project Object Model) trong Maven để quản lý các thư viện phụ thuộc, quá trình biên dịch, đóng gói và cài đặt lên kho. Trong tập tin POM còn chứa các tác vụ định phục vụ quá trình chạy các hàm kiểm thử, biên dịch.

```
<modelVersion>4.0.0</modelVersion>
<groupId>com.mycompany</groupId>
<artifactId>muleesbbeginner</artifactId>
<version>1.0.0-SNAPSHOT</version>
<packaging>mule</packaging>
<name>Mule muleesbbeginner Application</name>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

    <mule.version>3.8.3</mule.version>
    <mule.tools.version>1.2</mule.tools.version>
</properties>
```

Hình 1.14: Quản lý mã nguồn sử dụng Maven

Các thư viện phụ thuộc trong dự án được cấu hình theo phiên bản chuẩn để các lập trình viên trong một nhóm có thể tuân theo. Các hệ thống lớn, phức tạp thường đòi hỏi việc đóng gói và triển khai liên tục, quản lý, nâng cấp và bảo trì mất nhiều thời gian. Maven sinh ra để giải quyết được vấn đề này.

Ngoài việc có thể được tích hợp như một plugin trong các môi trường phát triển, maven còn cung cấp giao diện quản lý bằng dòng lệnh (command line). Các câu lệnh cơ bản phục vụ quá trình phát triển bao gồm: dọn dẹp (clean), biên dịch (build), kiểm thử (test), đóng gói (packag) và triển khai (deploy). Ngoài ra còn có các thuộc tính bổ xung phục vụ cho việc đóng gói ứng dụng.

Jenkins

Để hạn chế tối đa các lỗi xảy ra trong quá trình phát triển, ta nên sử dụng Jenkins để đặt lịch kiểm thử hồi quy cho bộ mã nguồn phát triển. Quá trình chạy các ca kiểm thử mất nhiều thời gian vì vậy việc tự động hoá giúp tiết kiệm được chi phí phát triển ứng dụng. Jenkins cung cấp lịch sử biên dịch và chạy thử để giúp quá trình tìm lỗi diễn ra dễ dàng hơn. Lập trình viên được cung cấp các thông tin cần thiết để khoanh vùng lỗi.

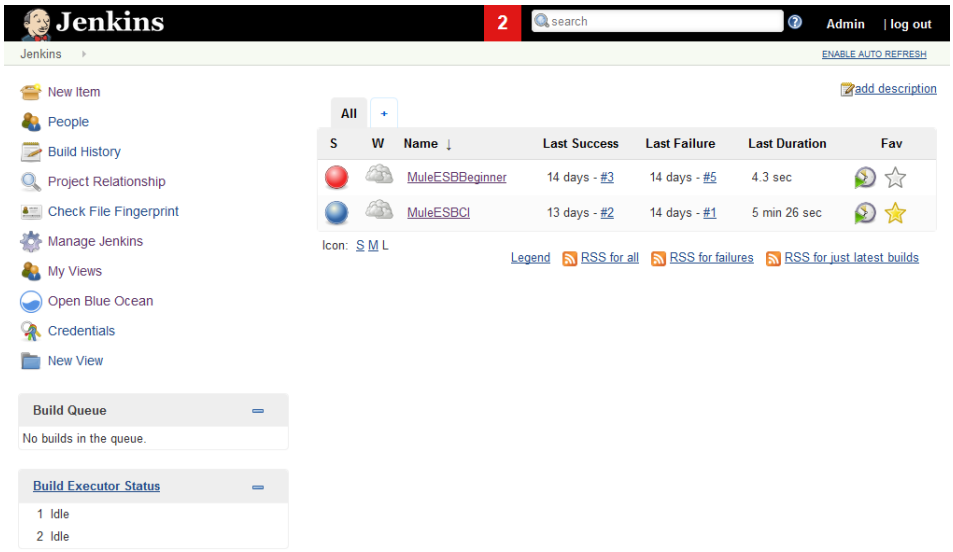
Jenkins là thư viện mã nguồn mở cho phép quản lý mã nguồn và triển khai một cách tự động, cả khi dự án đang trong giai đoạn phát triển. Nó giúp khép kín quy trình phát triển phần mềm một cách tự động theo mô hình Agile nói chung và việc tích hợp liên tục nói riêng. Jenkins được phát triển trên nền tảng Java, nó hỗ trợ nhiều nền tảng khác nhau và có

thể kết hợp được nhiều công cụ khác. Hiện tại, Jenkins hỗ trợ tích hợp hơn 400 plugins. Jenkins có thể chạy trên các hệ điều hành Windows, Linux, Mac OS và Solaris.

Chức năng nổi bật của Jenkins bao gồm: Quản lý, giám sát các tác vụ trong quá trình phát triển ứng dụng, lấy mã nguồn từ SVN/git, kiểm tra, dịch và triển khai ứng dụng thông qua việc gọi các thư viện hỗ trợ như Maven, Apache Ant...

Jenkins là một phần mềm cho phép tích hợp liên tục. Jenkins sẽ được cài đặt trên một máy chủ nơi diễn ra quá trình biên dịch. Một quy trình làm việc rất đơn giản về cách thức hoạt động của Jenkins bao gồm: kiểm tra sự thay đổi của mã nguồn, nếu có thay đổi sẽ chuyển sang bước 2, nếu không, hệ thống không đi tiếp; cập nhật mã nguồn, biên dịch và chạy các ca kiểm thử (nếu có cấu hình yêu cầu kiểm thử); kết quả dịch và chạy được xuất trên màn hình quản lý của Jenkins (Dashboard).

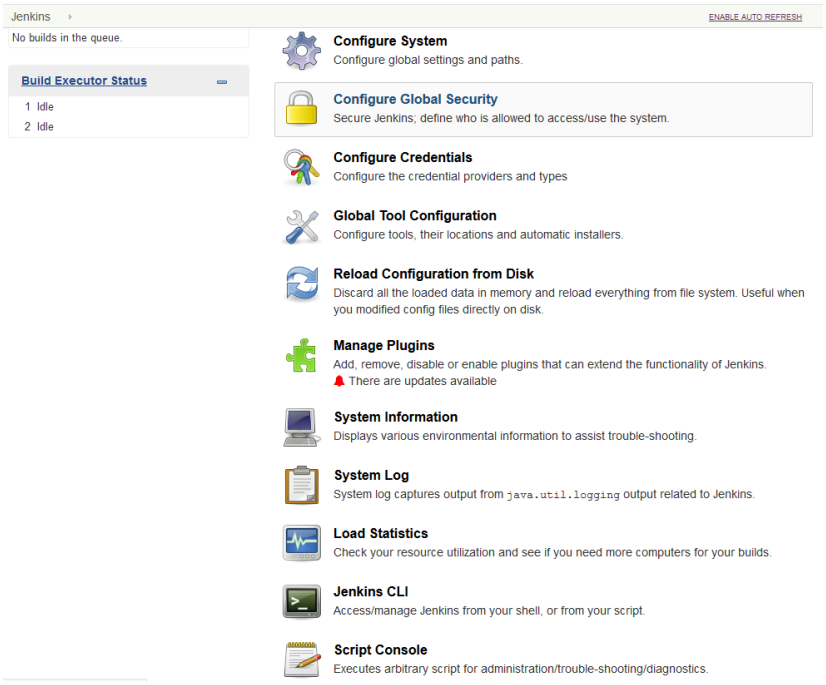
Jenkins cung cấp menu quản lý các cấu hình và tác vụ tại đường dẫn "Manage Jenkins" (xem).



Hình 1.15: Màn hình chính Jenkins

Jenkins cung cấp các màn hình cấu hình cho ứng dụng (xem)

Cấu hình hệ thống (Configure System): Đây là có thể quản lý các đường dẫn đến các công cụ khác nhau để sử dụng trong các bản dịch, chẳng hạn như các JDK, các phiên bản của Ant và Maven, cũng như các tùy chọn bảo mật, máy chủ email và các chi tiết cấu hình trên toàn hệ thống khác. Khi plugin được cài đặt, Jenkins sẽ tự động thêm các trường cấu hình bắt buộc sau khi các plugin được cài đặt.

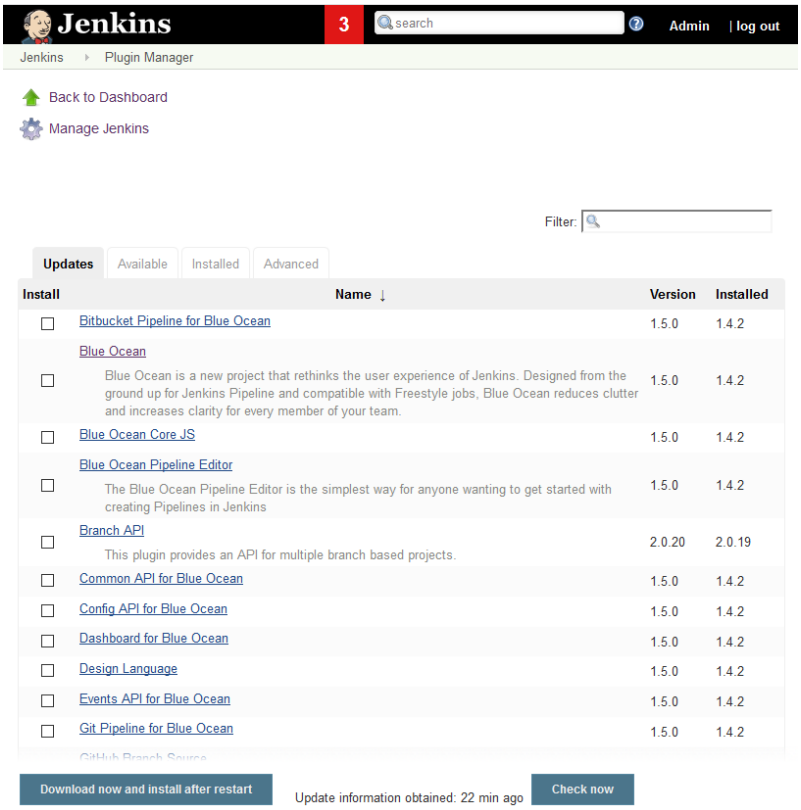


Hình 1.16: Cấu hình tùy chỉnh của Jenkins

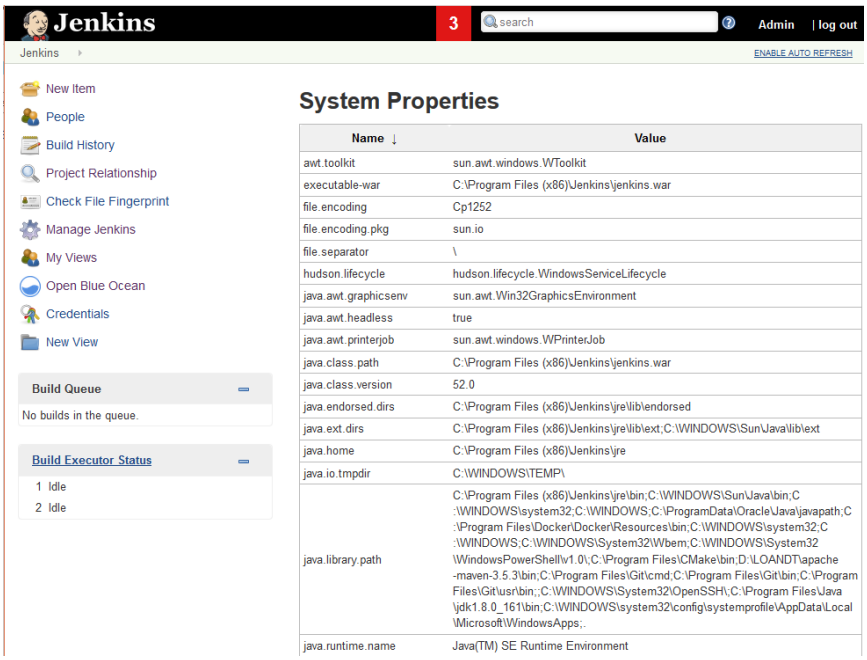
Tải lại cấu hình từ ổ đĩa (Reload Configuration from Disk): Jenkins lưu trữ tất cả các hệ thống của nó và xây dựng các chi tiết cấu hình công việc dưới dạng các tệp XML được lưu trữ trong thư mục home của Jenkins. Ở đây cũng có tất cả lịch sử biên dịch được lưu trữ. Nếu người dùng đang chuyển các công việc xây dựng từ một cá thể Jenkins này sang một thể hiện khác hoặc lưu trữ các công việc xây dựng cũ, người dùng sẽ cần phải thêm hoặc xóa các thư mục công việc xây dựng tương ứng vào thư mục xây dựng của Jenkins. Người dùng không cần phải sử dụng Jenkins ngoại tuyến để làm điều này, mà có thể chỉ cần sử dụng tùy chọn "Tải lại cấu hình từ đĩa" để tải lại hệ thống Jenkins và tạo cấu hình công việc trực tiếp.

Quản lý Plugin (Manage Plugin): Màn hình hỗ trợ cài đặt một loạt các plugin của bên thứ ba ngay từ các công cụ quản lý mã nguồn khác nhau như Git, Mercurial hoặc ClearCase, để mã hóa báo cáo số liệu về mức độ bảo hiểm và chất lượng mã. Plugins có thể được cài đặt, cập nhật và loại bỏ thông qua màn hình này (xem).

Thông tin hệ thống (System Information): Màn hình này hiển thị danh sách tất cả các thuộc tính hệ thống Java và biến môi trường hệ thống hiện tại. Ở đây người ta có thể kiểm tra chính xác phiên bản Java Jenkins đang chạy, người dùng nào đang chạy tác vụ. cho thấy một số thông tin về giá trị tên có sẵn trong phần này.



Hình 1.17: Quản lý plugins



Hình 1.18: Thông tin hệ thống của Jenkins

Lịch sử hệ thống (System Log): Màn hình System Log là cách thuận tiện để xem các tập tin lịch sử Jenkins trong thời gian thực. Việc sử dụng chính của màn hình này là để tìm và khắc phục sự cố.

Thống kê tải (Load Statistics): Các trang này hiển thị dữ liệu đồ họa về mức độ bận của phiên bản Jenkins về số lượng các bản biên dịch đồng thời và độ dài của hàng đợi biên dịch cho biết một dự án cần phải đợi bao lâu trước khi được thực thi. Những số liệu thống kê này có thể đưa ra ý tưởng tốt về việc liệu các công suất phụ hay các nút biên dịch bổ sung có được yêu cầu từ quan điểm cơ sở hạ tầng hay không.

Tập lệnh kịch bản (Script Console)**:** Màn hình này cho phép chạy các tập lệnh Groovy trên máy chủ. Nó rất hữu ích cho việc xử lý sự cố xảy ra do ứng dụng yêu cầu một cấu hình mạnh.

Hẹn giờ tắt (Prepare for Shutdown): Nếu cần phải tắt Jenkins, đặc biệt khi máy chủ Jenkins đang thực thi một phiên bản của dự án, có thể sử dụng tính năng "Prepare for Shutdown". Cuối cùng, khi tất cả các bản thực thi hiện tại đã hoàn thành, Jenkins sẽ tự động tắt hệ thống

1.3. Kiểm thử

Mỗi sản phẩm phần mềm khi phát triển đều phải trải qua giai đoạn kiểm thử để đảm bảo chất lượng sản phẩm. Vì vậy việc kiểm thử là một trong những giai đoạn quan trọng trong quá trình phát triển bất cứ một ứng dụng nào.

Kiểm thử phần mềm là hoạt động khảo sát thực tiễn sản phẩm hay dịch vụ phần mềm trong đúng môi trường dự định triển khai phần mềm đó, nhằm cung cấp cho các bên liên quan thông tin về chất lượng của sản phẩm hay dịch vụ phần mềm. Mục đích của kiểm thử phần mềm là tìm ra các lỗi hay khiếm khuyết nhằm đảm bảo chương trình hoạt động đạt được hiệu quả tối đa. "Kiểm thử phần mềm là quá trình thực thi một chương trình với mục đích tìm lỗi" [11].

Theo bảng chú giải thuật ngữ chuẩn IEEE – IEEE Standard Glossary of Software Engineering Technology: "Kiểm thử phần mềm là quá trình khảo sát một hệ thống hay thành phần dưới những điều kiện xác định, quan sát và ghi lại các kết quả, và đánh giá một khía cạnh nào đó của hệ thống hay thành phần đó"

Đảm bảo chất lượng phần mềm Quality Assurance (QA) là việc đảm bảo không xảy ra lỗi, thiếu sót trong quá trình phát triển, chuyển giao , sử dụng phần mềm. Kiểm thử [12] là kỹ thuật đảm bảo chất lượng ở pha lập trình và sau lập trình. Đây là một phần con của hoạt động QA và là hoạt động chủ yếu của QA. Kiểm thử bao gồm hai phần : Verification (kiểm chứng) và validation (thẩm định). Hoạt động kiểm chứng (verification) trả lời cho câu hỏi "Are you building it right?", mục tiêu là tìm ra lỗi lập trình so với thiết kế, đây là công việc của người phát triển. Hoạt động thẩm định (Validation) trả lời cho câu hỏi "Are you building the right thing?", mục tiêu là kiểm thử chấp nhận (UAT).

1.3.1. Kiểm thử hộp đen

Kiểm thử hộp đen xem chương trình như một hộp đen, kiểm thử viên không cần quan tâm đến việc cấu trúc và hoạt động bên trong của chương trình, thay vào đó, kiểm thử viên tập trung tìm các đặc điểm mà chương trình thực hiện không đúng như đặc tả của nó. Các ca kiểm thử được sinh ra từ đặc tả người dùng (user requirement) của chương trình.

Các phương pháp kiểm thử hộp đen bao gồm kiểm thử tương đương, phân tích giá trị biên, kiểm thử mọi cặp, kiểm thử fuzz. Kiểm thử hộp đen không có mối liên quan nào tới mã nguồn của chương trình, kiểm thử viên chỉ cần quan tâm đến việc một chức năng có hành xử đúng như đặc tả người dùng đưa ra. Do kiểm thử viên không biết được phần mềm bên trong hoạt động thế nào, nên họ cần phải đưa ra nhiều ca kiểm thử khác nhau để có thể bao quát hết được chức năng. Kiểm thử hộp đen mang tính đánh giá khách quan nhưng theo chiều hướng thăm dò mù.

1.3.2. Kiểm thử hộp trắng

Kiểm thử hộp trắng là một chiến lược kiểm thử khác, trái ngược với kiểm thử hộp đen. Kiểm thử hộp trắng cho phép khảo sát cấu trúc bên trong của chương trình. Chiến lược này xuất phát từ dữ liệu kiểm thử bằng sự kiểm thử tính logic của chương trình. Người kiểm thử viên (thường là lập trình viên) sẽ truy cập vào cấu trúc dữ liệu và giải thuật cùng với mã nguồn của chương trình.

Các loại kiểm thử hộp trắng bao gồm: kiểm thử giao diện lập trình ứng dụng (API testing), đây là phương pháp kiểm thử của ứng dụng sử dụng các API. Kiểm thử bao phủ mã lệnh (code coverage) tạo các ca kiểm thử để đáp ứng một số tiêu chuẩn về bao phủ mã lệnh. Các phương pháp gán lỗi (Fault injection), các phương pháp kiểm thử hoán chuyển (Mutation testing methods), kiểm thử tĩnh (static testing).

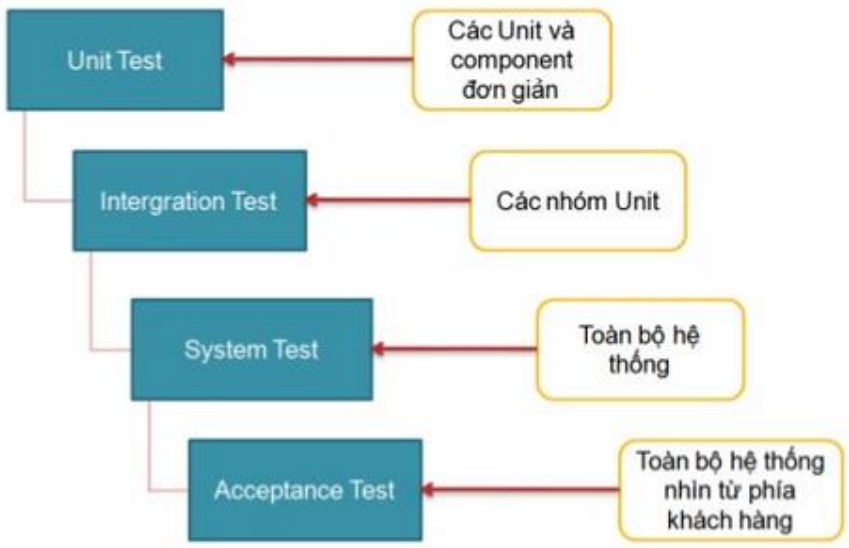
Phương pháp kiểm thử hộp trắng cũng có thể được sử dụng dể đánh giá sự hoàn thành của một bộ kiểm thử mà được tạo cùng với các phương pháp kiểm thử hộp đen. Điều này cho phép các nhóm phần mềm khảo sát các phần của một hệ thống ít khi được kiểm tra và đảm bảo rằng những điểm chức năng quan trọng nhất đã được kiểm tra.

1.3.3. Kiểm thử hộp xám

Kiểm thử hộp xám đòi hỏi phải có sự truy cập tới cấu trúc dữ liệu và giải thuật bên trong cho những mục đích thiết kế các ca kiểm thử, nhưng là kiểm thử ở mức người sử dụng hay mức hộp đen. Việc thao tác tới dữ liệu đầu vào và định dạng dữ liệu đầu ra là không rõ ràng, bởi vì đầu vào và đầu ra ở bên ngoài “hộp đen” chương trình. Sự khác biệt này đặc biệt quan trọng khi kiểm thử tích hợp – intergration testing giữa hai thành phần (modules) mã nguồn được viết bởi hai lập trình viên khác nhau, trong đó chỉ giao diện là được đưa ra để kiểm thử. Kiểm thử hộp xám có thể cũng bao gồm cả thiết kế đối chiếu để quyết định, ví dụ, giá trị biên không thay đổi.

1.3.4. Các cấp độ kiểm thử

Quá trình kiểm thử được chia thành nhiều cấp độ như thể hiện trong Hình 1.19.



Hình 1.19: Sơ đồ các cấp độ kiểm thử

1.3.4.1. Kiểm thử đơn vị

Kiểm thử đơn vị- Unit Test là việc kiểm thử từng thành phần cụ thể của chương trình, do lập trình viên thực hiện. Một đơn vị có thể là một phương thức, thủ tục hay một lớp của chương trình, các thành phần này có kích thước nhỏ và hoạt động đơn giản. Do đó, kiểm thử đơn vị không có gì phức tạp, kết quả lỗi xảy ra dễ dàng khắc phục được.

Kiểm thử đơn vị thường do lập trình viên thực hiện. Công đoạn này làm càng sớm càng tốt trong quá trình viết mã nguồn và xuyên suốt trong quá trình phát triển phần mềm. Thông thường, kiểm thử đơn vị thường yêu cầu kiểm thử viên có kiến thức về thiết kế và mã nguồn của chương trình. Mục đích của kiểm thử đơn vị là đảm bảo thông tin được xử lý và trả về từ thành phần con là chính xác, trong mối tương quan với dữ liệu nhập và chức năng của đơn vị đó.

1.3.4.2. Kiểm thử tích hợp

Kiểm thử tích hợp- Intergration Test kết hợp các thành phần của một ứng dụng và kiểm thử như một ứng dụng đã hoàn thành. Trong khi kiểm thử đơn vị kiểm tra các thành phần và đơn vị riêng lẻ thì kiểm thử tích hợp kết hợp chúng lại với nhau và kiểm tra chức năng giao tiếp giữa chúng.

Mục tiêu chính của kiểm thử tích hợp là phát hiện lỗi giao tiếp xảy ra giữa các thành phần, đơn vị. Hơn nữa, kiểm thử tích hợp thực hiện tích hợp các đơn vị đơn lẻ thành các hệ thống nhỏ và cuối cùng là nguyên hệ thống hoàn chỉnh cho việc kiểm thử ở mức hệ thống.

Trong kiểm thử đơn vị, lập trình viên tìm lỗi liên quan đến chức năng và cấu trúc nội tại của từng đơn vị. Có một số phép kiểm thử đơn giản trên giao tiếp giữa đơn vị và các thành phần liên quan khác, tuy nhiên mọi giao tiếp liên quan đến đơn vị chỉ thật sự được keierm tra đầy đủ khi thực hiện liên kết vào với nhau trong quá trình kiểm thử tích hợp.

Thông thường, kiểm thử tích hợp được thực hiện trên các đơn vị đã được kiểm tra qua kiểm thử đơn vị. Việc tích hợp giữa các đơn vị gây ra những tình huống hoàn toàn khác với hành vi của từng đơn vị khi hoạt động riêng lẻ. Một chiến lược cần quan tâm trong kiểm thử tích hợp đó là nên tích hợp dần từng phần. Lúc này, ta cần kiểm thử giao tiếp của các đơn vị mới thêm vào hệ thống có các đơn vị đã tích hợp trước đó, điều này sẽ làm giảm số lượng các ca kiểm thử cần làm, giảm được sai sót trong quá trình kiểm thử.

Có 4 loại trong kiểm thử tích hợp. Kiểm thử cấu trúc (Structure test): tương tự như kiểm thử hộp trắng, kiểm thử cấu trúc đảm bảo các thành phần bên trong của một chương trình chạy đúng và chú trọng đến hoạt động của các thành phần cấu trúc nôi tại của chương trình như các câu lệnh, rẽ nhánh bên trong. Kiểm thử chức năng (Functional Test): tương tự như kiểm thử hộp đen, kiểm thử chức năng chỉ chú trọng đến chức năng của chương trình, mà không quan tâm đến hoạt động bên trong của ứng dụng, nó chỉ khảo sát chức năng của chương trình theo yêu cầu kỹ thuật. Kiểm thử hiệu năng (Performance Test): là việc kiểm thử sự vận hành của hệ thống như tốc độ truyền, nhận và phản hồi thông điệp trong một điều kiện cho trước. Kiểm thử khả năng chịu tải (Stress Test): có chức năng kiểm tra độ chịu tải của hệ thống, tính ổn định trong một hoàn cảnh cho trước.

1.3.4.3. Kiểm thử hệ thống

Kiểm thử hệ thống – System Test là kiểm thử thiết kế và toàn bộ hệ thống sau khi tích hợp có thỏa mãn yêu cầu người dùng đặt ra hay khong.

Kiểm thử hệ thống bắt đầu sau khi đã tích hợp thành công các thành phần của hệ thống với nhau. Ở mức độ này, kiểm thử viên chú trọng vào việc đánh giá về hoạt động, thao tác, độ tin cậy và các yêu cầu khác liên quan đến chất lượng của toàn hệ thống như các yêu cầu phi chức năng.

Với kiểm thử tích hợp chú trọng sự giao tiếp giữa các đơn vị, thành phần, còn kiểm thử hệ thống chú trọng các hành vi và lỗi trên toàn hệ thống. Mức kiểm thử này thích hợp cho việc phát hiện lỗi giao tiếp với các hệ thống hoặc ứng dụng của bên ngoài. Việc kiểm thử hệ thống đòi hỏi nhiều thời gian, công sức, tính chính xác và khách quan của người kiểm thử.

Kiểm thử hệ thống bao gồm sáu loại. Kiểm thử chức năng (Functional Test) đảm bảo các hành vi của hệ thống thỏa mãn đúng yêu cầu thiết kế. Kiểm thử hiệu năng (Performance Test) đảm bảo tối ưu việc phân bổ tài nguyên hệ thống nhằm đạt được các chỉ tiêu như thời gian xử lý, phản hồi,.. Kiểm thử tải (Stress Test hoặc Load Test) đảm bảo hệ thống vận hành được trong điều kiện khắc nghiệt như lượng truy cập tăng cao, đường truyền kém… Kiểm thử cấu hình (Configuraion Test) kiểm thử hệ thống với các cấu hình ở các môi trường, thiết bị khác nhau. Kiểm thử bảo mật (Security Test) đảm bảo tính toàn vẹn, bảo mật của dữ liệu và của hệ thống. Kiểm thử khả năng phục hồi (Recovery Test) đảm bảo hệ thống có thể khôi phục lại trạng thái ổn định trước khi xảy ra mất mát tài nguyên, điều này đặc biệt quan trọng đối với các hệ thống tài chính – ngân hàng.

1.3.4.4. Kiểm thử chấp nhận

Thông thường, sau giai đoạn kiểm thử hệ thống sẽ là kiểm thử chấp nhận (Acceptance Test). Bước này do khách hàng đưa ra yêu cầu thực hiện. Quá trình kiểm thử này có ý nghĩa quan trọng trong việc xác định xem chương trình có đáp ứng được như mong đợi của khách hàng hay không.

Ngoài các cấp độ trên, còn có một số cấp độ kiểm thử khác như kiểm thử hồi quy và kiểm thử tính đúng.

Kiểm thử hồi quy là "sự kiểm tra lại có lựa chọn của một hệ thống hay thành phần để xác minh là những sự thay đổi không gây ra những hậu quả không mong muốn" (Theo chuẩn IEEE610-90). Trên thực tế, phần mềm đã trải qua các bước kiểm tra trước đó vẫn có thể thực hiện kiểm tra lại. Đây là sự lặp lại các bước kiểm thử để chỉ ra rằng hệ thống phần mềm không bị thay đổi, ngoại trừ tới mức như yêu cầu

Tính đúng đắn là yêu cầu tối thiểu của phần mềm, là mục đích chủ yếu của kiểm thử. Kiểm thử tính đúng đắn – Correctness Testing, kiểm thử viên có thể biết hay không biết chi tiết bên trong của các thành phần phần mềm được kiểm thử như luồng điều khiển, dòng dữ liệu,…

CHƯƠNG 2. KHÓ KHĂN VÀ CÁC VẤN ĐỀ CẦN GIẢI QUYẾT

Các giải pháp tích hợp tạo nên xương sống của các hệ thống dịch vụ với các tiến trình xử lý dữ liệu thời gian thực. Bất kỳ một sự cố, lỗi không mong muốn cũng có thể gây ra ảnh hưởng lớn đến các tổ chức với các nguy cơ mất an toàn dữ liệu. Bên cạnh vấn đề tổn thất do thiếu sót của phần mềm, việc yếu kém về quy trình, chiến lược kiểm thử cũng gây ra sự chậm trễ cũng như các phần việc phát sinh không đáng có trong quá trình phát triển, triển khai sản phẩm. Các giải pháp tích hợp hiện nay thường được kiểm thử rất ít hoặc bỏ qua kiểm thử, hoặc nếu có thì được làm thủ công và không mang tính thường xuyên. Nguyên nhân dẫn đến tình trạng này xuất phát từ nhiều nguyên nhân. Đầu tiên, kiến trúc ESB bao gồm nhiều thành phần, sử dụng các cơ chế truyền nhận bản tin bất đồng bộ, các giải pháp áp dụng thường phức tạp, phân tán, dẫn đến việc thiết lập môi trường kiểm thử giống với môi trường triển khai thực tế rất khó khăn. Ngoài ra, quỹ thời gian cho phát triển, tích hợp thành phần mới, thường không nhiều, gây ra tình trạng quy trình kiểm thử bị rút ngắn hoặc bỏ qua. Mặt khác hiện tại, có rất ít công cụ hỗ trợ kiểm thử các API, việc kiểm thử đều phải sử dụng qua các công cụ riêng lẻ hoặc kiểm tra từ lớp ngoài, giao diện người dùng. Việc này dẫn tới khi xảy ra lỗi tại một ứng dụng trong hệ thống sẽ đòi hỏi việc tìm lỗi và sửa đổi nhiều ứng dụng cùng lúc, gây mất thời gian và tốn kém tài nguyên, các lỗi không được kiểm soát chặt chẽ. Chương này sẽ giới thiệu và phân tích những khó khăn đang gặp phải và đưa ra các vấn đề cần tập trung giải quyết cho các khó khăn đó.

2.1. Môi trường kiểm thử

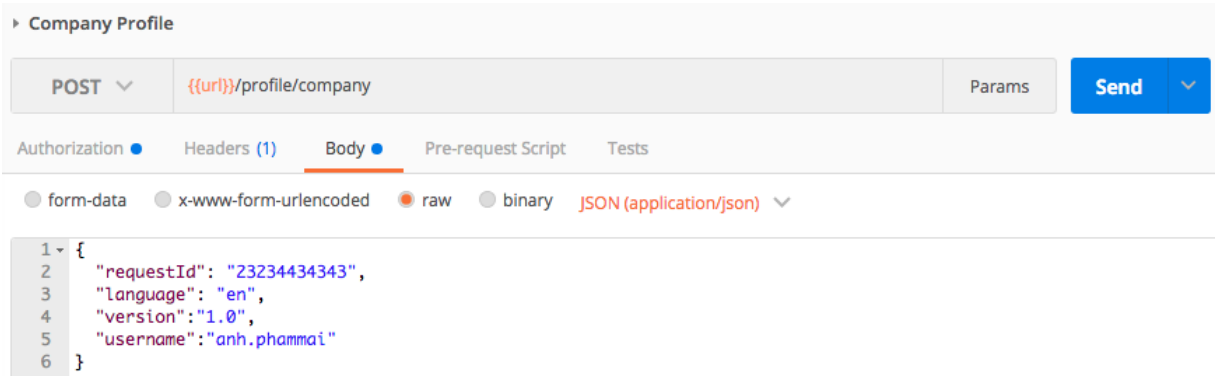
Kiến trúc ESB cung cấp kết nối giữa các thành phần khác biệt nhau trong một hệ thống hoặc giữa các hệ thống khác nhau thông qua một hạ tầng thông tin chung. Kiến trúc này bao gồm số lượng lớn các thành phần và các tính năng. Để có thể kiểm thử hiệu quả hệ thống sử dụng kiến trúc ESB, tất cả các thành phần và tính năng của hệ thống đều phải được bao quát đến. Tuy nhiên các thành phần trong hệ thống lại được xây dựng từ các ngôn ngữ khác nhau, thậm chí nằm trên các hạ tầng của các công ty khác nhau. Chính vì vậy, trong nhiều trường hợp việc giả lập môi trường kiểm thử giống với môi trường triển khai thực tế là rất khó khăn. Lấy ví dụ khi hệ thống ngân hàng tích hợp thêm một cổng thanh toán, bằng cách nào để ta xác định được việc kết nối và thanh toán đến cổng thanh toán đó thành công khi mà lập trình viên không có được môi trường chạy thực tế trên môi trường phát triển. Ngoài ra, các hệ thống truyền tin còn có cơ chế gửi bản tin bất đồng bộ, hoặc gặp phải các vấn đề về tính toán song song, những vấn đề này cũng rất khó có thể kiểm thử và phát hiện. Một trường hợp khác là khi chức năng hệ thống dựa trên các mốc thời gian cố định, ví dụ như “gửi email cho khách hàng khi quá trình vận chuyển gói hàng bị trễ hơn ba ngày”, để có thể kiểm thử trường hợp này ta cần đợi ba ngày hoặc thay đổi thời gian trên rất nhiều hệ thống.

2.2. Công cụ hỗ trợ kiểm thử

Quá trình kiểm thử hệ thống sử dụng kiến trúc ESB chủ yếu tập trung vào giao tiếp giữa các thành phần trong hệ thống. Vì vậy, quy trình kiểm thử không chú trọng vào phần kiểm thử giao diện người dùng mà tập trung vào các API của các thành phần hệ thống. Hiện nay, công cụ hỗ trợ kiểm thử API đang phổ biến là SoapUI và Postman. Ngoài ra, đối với việc kiểm thử ứng dụng thông thường, lập trình viên thường sử dụng JUnit để hỗ trợ quá trình kiểm thử. Đối với ứng dụng xây dựng trên Mule framework, Mulesoft có hỗ trợ bộ thư viện kiểm thử MUnit

Postman

Postman là công cụ cho phép kết nối với các API, đặc biệt là các ứng dụng viết theo giao thức RESTful. Người dùng có thể gọi vào các API mà không cần thiết viết mã nguồn. Lập trình viên có thể thực hiện truyền trực tiếp các tham số dưới dạng text, json, xml, html,…(xem *Hình 2.1*)



Hình 2.1: Tham số trên Postman

Thay vì việc phải viết đoạn mã nguồn như *Hình 2.2*:

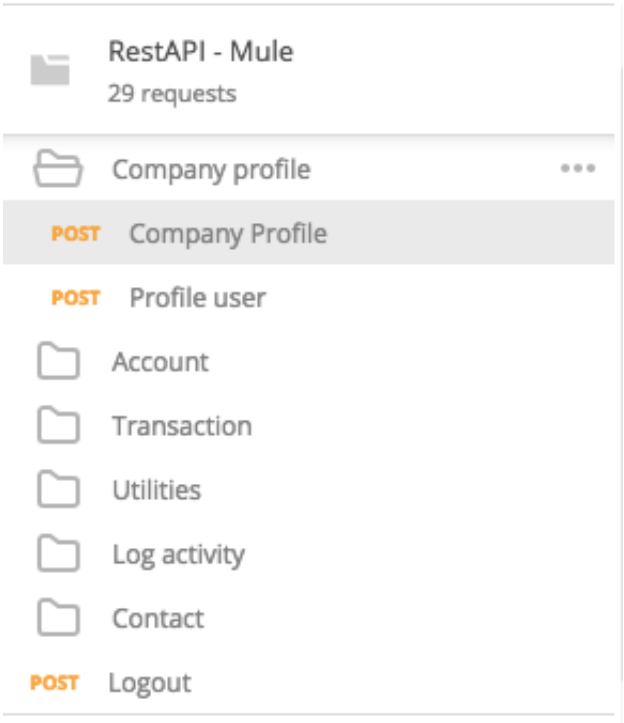
```
HttpResponse<String> response = Unirest.post("https://webdemo.vietinbank.vn/efast-service/profile/company")
  .header("content-type", "application/json")
  .header("cache-control", "no-cache")
  .header("postman-token", "4677e4c9-b55f-6338-76e5-860df9658a96")
  .body("{\n  \"requestId\": \"23234434343\",\n  \"language\": \"en\",\n  \"version\":\"1.0\",\n\"username\":\"anh.
      phammai\"\n}\n\n")
  .asString();
```

Hình 2.2: Mã nguồn gọi API

Postman hỗ trợ tất cả các phương thức của HTTP như GET, POST, PUT, PATCH, DELETE,…và còn cho phép lưu lại lịch sử các lần gọi, đồng bộ các lời gọi theo tài khoản đã đăng ký. Ngoài ra postman có hỗ trợ các phương thức xác thực như OAuth1.0, OAuth2.0,… Postman cho phép thay đổi header của các lời gọi tuỳ theo phương thức xác thực.

Postman đưa ra cho người dùng các lựa chọn phiên bản miễn phí và phiên bản thương mại mất phí. Phiên bản miễn phí cung cấp các chức năng cơ bản như tạo nhóm lời gọi, tự động chạy lời gọi theo nhóm, giám sát các API dựa trên các lời gọi (*Hình 2.3*). Trong khi

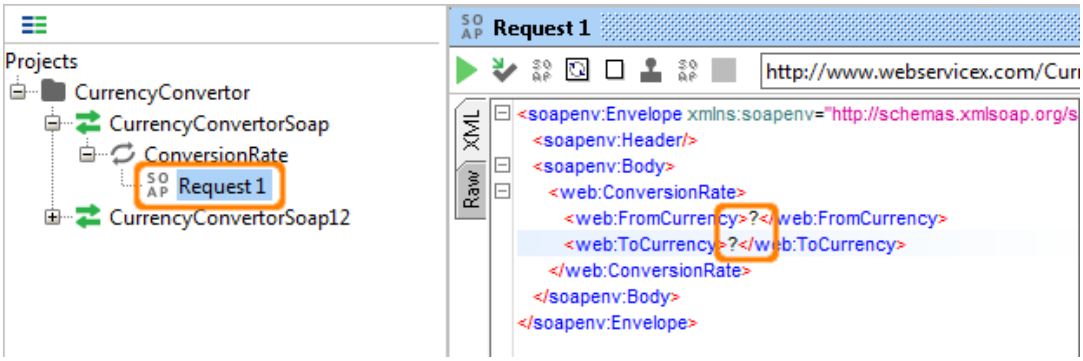
đó, phiên bản mất phí cung cấp thêm các chức năng nâng cao hơn như làm việc theo nhóm, tạo tài liệu API, giám sát chặt chẽ hơn…



Hình 2.3: Quản lý các lời gọi API

SOAPUI

Ngoài Postman, SOAPUI [13] cũng là công cụ hỗ trợ kiểm thử API được sử dụng phổ biến hiện nay. Đây là công cụ kiểm thử có nền tảng mã nguồn mở hàng đầu cho phép kiểm thử viên thực hiện các loại kiểm thử như: kiểm thử chức năng, hồi quy, thử tải một cách tự động trên các Web API khác nhau. Hỗ trợ tất cả các giao thức chuẩn và công nghệ để kiểm thử các loại API. Ngoài ra, SOAPUI cung cấp giao diện đồ hoạ để người dùng không chuyên dễ dàng tương tác (*Hình 2.4*).



Hình 2.4: Lời gọi API trên SOAPUI

SOAPUI có thể tích hợp được với nhiều công cụ phổ biến như Apache Maven, HUDSON, JUnit, Apache Ant… SOAPUI cung cấp hai phiên bản, một phiên bản miễn phí và một phiên bản trả phí. Phiên bản miễn phí đáp ứng hầu hết các yêu cầu của lập trình viên

về các giao thức kết nối, việc chạy tự động và quản lý các ca kiểm thử. Phiên bản mất phí cung cấp thêm chức năng tạo báo cáo cho các ca kiểm thử đã chạy.

Như vậy, có thể thấy, trong khi Postman thuần tuý chỉ là cung cấp khả năng thực hiện các lời gọi đến các API của ứng dung, thì SoapUI là công cụ nâng cao hơn tập trung vào tạo các ca kiểm thử, tích hợp với các công cụ hỗ trợ khác. Tuy nhiên tính năng xuất báo cáo các ca kiểm thử đã chạy lại ở phiên bản mất phí và công cụ này chưa có khả năng tự sinh ca kiểm thử.

JUnit

JUnit là một bộ thư viện mã nguồn mở được sinh ra nhằm hỗ trợ việc viết và chạy các mã nguồn kiểm thử trên ngôn ngữ Java. Được phát triển đầu tiên bởi Erich Gamma và Kent Beck, JUnit là một bước tiến hóa quan trọng của phát triển hướng kiểm thử (Test Driven Development). JUnit cung cấp cả giao diện đồ họa và giao diện dòng lệnh giúp việc viết mã nguồn kiểm thử dễ dàng hơn. Lập trình viên có thể dùng JUnit để xây dựng tích lũy các bộ kiểm thử để đo lường tiến độ và phát hiện các ảnh hưởng không mong muốn. Các bộ kiểm thử có thể chạy được liên tục và cung cấp kết quả luôn tại màn hình. mô tả kiến trúc của JUnit.

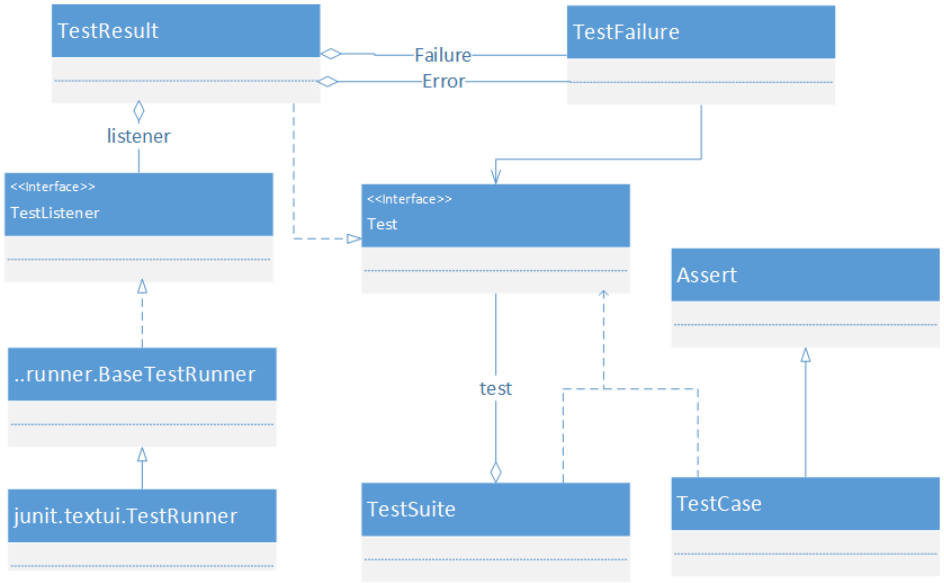
JUnit cung cấp các chú thích hỗ trợ việc kiểm thử, một số chú thích hay sử dụng bao gồm: Before, After, BeforeClass, AfterClass, Ignore, RunWith, Test. Ngoài ra, JUnit cũng cung cấp các lớp hỗ trợ cơ bản như Assert, TestCase, TestResult, TestSuite.

Assert chứa một tập các phương thức xác nhận để viết một ca kiểm thử. Chỉ các ca kiểm thử thất bại mới được ghi nhận lại toàn bộ lịch sử chạy. Một vài phương thức quan trọng hay sử dụng trong lớp Assert như kiểm tra xem hai biến có bằng nhau hay không kể cả biến thuộc loại nguyên thủy hay biến loại đối tượng, kiểm tra xem một điều kiện là đúng, sai, bị rỗng, hay hai đối tượng có cùng trỏ tới một địa chỉ trùng nhau. Ngoài ra lớp này còn cung cấp phương thức so sánh hai mảng dữ liệu.

TestCase kiểm tra xác định các trường hợp nào chạy kiểm thử nhiều. Mộ số phương thức quan trọng của lớp này bao gồm: countTestCase đếm số trường hợp kiểm tra bằng cách chạy, createTestResult tạo mới một đối tượng TestResult mặc định, getName lây tên của TestCase. Run là một phương thức chạy kiểm thử, thu thập kết quả với một đối tượng TestResult, setName cấu hình tên của một TestCase, setUp thiết lập các cấu hình ví dụ: mở kết nối…, tearDown khắc phục như: đóng kết nối…, toString trả về TestCase dưới dạng xâu.

TestResult thu thập kết quả của việc thực hiện một trường hợp thử nghiệm. Đây là một thể hiện các mẫu tham số.Các khuôn khổ kiểm tra phân biệt giữa thất bại (fail) và sai sót (defect). Một thất bại được dự đoán và kiểm tra với khẳng định. Lỗi là vấn đề không

lường trước được như việc truy cập quá số phần tử của mảng. Một số phương pháp quan trọng của lớp TestResult như addError thêm một lỗi vào danh sách các lỗi, addFailure thêm một thất bại đối với danh sách của những thất bại, endTest thông báo kết quả là một thử nghiệm đã được hoàn thành, errorCount đếm số lỗi được phát hiện, errors trả về một Enumeration cho các lỗi, failureCount đếm các số thất bại được phát hiện, run chạy một ca kiểm thử, runCount đếm số lượng các ca kiểm thử chạy, startTest thông báo kết quả là một thử nghiệm sẽ được bắt đầu, stop đánh dấu việc dừng kiểm thử.



Hình 2.5: Kiến trúc JUnit

Một TestSuite là một hỗn hợp các ca kiểm thử, chạy một tập hợp các trường hợp thử nghiệm theo gói. TestSuite cung cấp các phương thức quan trọng như: addTest thêm một bộ thử nghiệm, addTestResult thêm các TestCase, countTestCase đếm số lượng trường hợp thử nghiệm, run chạy các ca kiểm thử và thu thập kết quả của chúng trong cùng một TestSuite, setName thiết lập tên cho phần mềm cần kiểm tra, testAt trả về kết quả kiểm tra ở một vị trí nhất định, testCount đếm số ca kiểm thử trong gói kiểm thử đang thực hiện, warning trả về một cảnh báo thiếu hay thừa thông tin, ví dụ như: thư viện phụ thuộc... Các gói kiểm thử (Test Suite) cho phép gom và chạy các ca kiểm thử cùng nhau, để chạy các gói này, JUnit cung cấp các chú thích để hỗ trợ việc quản lý các ca kiểm thử như:

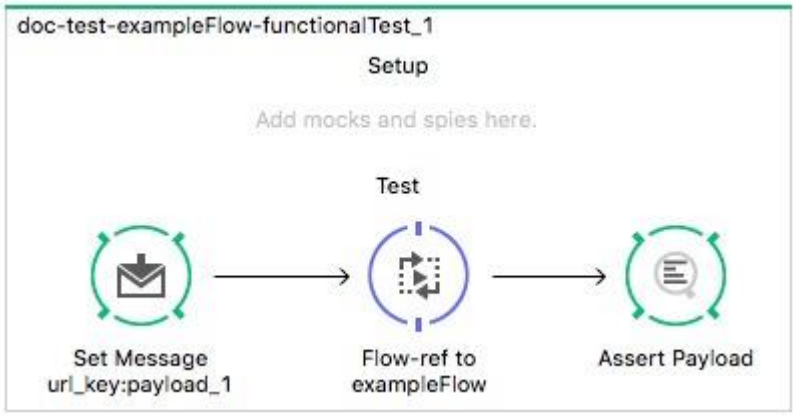
```
@Runwith(Suite.class)
@SuiteClasses(test1.class,test2.class……) or
@Suite.SuiteClasses ({test1.class, test2.class……})
```

MUnit

MUnit là bộ thư viện hỗ trợ kiểm thử trên ứng dụng Mule, cho phép xây dựng các ca kiểm thử tự động để kiểm thử việc tích hợp và API của ứng dụng ESB được phát triển trên nền tảng Mule.

MUnit có thể được kết hợp với Maven và Surefile để tích hợp các thư viện độc lập. MUnit cho phép tạo ca kiểm thử bằng việc sử dụng Mule code hoặc java, vô hiệu hoá các điểm kết nối cuối được gửi đến, vô hiệu hoá các kết nối đầu cuối, mô phỏng các điểm cuối gọi ra, theo dõi các tiến trình xử lý nội dung message, xác minh các mesage gọi xử lý, tạo ra các ca kiểm thử đơn vị bao gồm cả các kiểm thử tích hợp tại môi trường local. MUnit cho phép gọi một máy chủ FPT/SFTP, DB hay mail.

mô tả việc sử dụng MUnit code để thực hiện kiểm thử một luồng trong MuleESB



Hình 2.6: MUnit Code

mô tả việc sử dụng MUnit test trên ngôn ngữ Java

```
    //doc-test-exampleFlow2Test1, result must be "exampleSub_Flow2"
    @Test
    public void test() throws Exception{
        String myStringPayload = "payload_1";
        MuleMessage messageToBeReturned = muleMessageWithPayload(myStringPayload);
        MessageProcessorMocker mocker = whenMessageProcessor("set-payload");
        mocker.thenReturn(messageToBeReturned);

        MuleEvent resultMuleEvent = runFlow("exampleFlow2", testEvent(myStringPayload));
        Assert.assertEquals(myStringPayload, resultMuleEvent.getMessage().getPayload());
        System.out.println(resultMuleEvent.getMessage().getPayload());
    }
```

Hình 2.7: MUnit viết trên Java

CHƯƠNG 3. QUY TRÌNH KIỂM THỬ CHO ỨNG DỤNG ESB

Hiện nay, có nhiều nghiên cứu tập trung vào việc kiểm thử ứng dụng ESB, Các chiến lược này bao gồm ba chiến thuật chính: kiểm thử đơn vị, kiểm thử tích hợp và kiểm thử hồi quy.

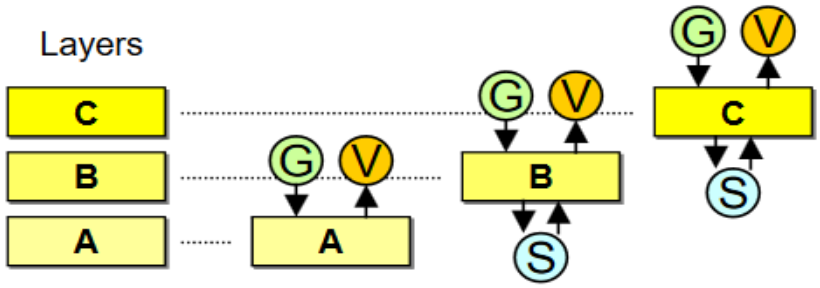
Với đặc điểm của kiến trúc ESB cấu thành từ nhiều thành phần, ta cần một chiến lược kiểm thử bao quát được giao tiếp trong nội tại hệ thống. Chính vì vậy, ta cần phải thực hiện kiểm thử riêng lẻ (kiểm thử đơn vị) từng thành phần càng nhiều càng tốt nhằm tránh các lỗi gây ra bởi các thành phần không phải thành phần kiểm thử trong ca kiểm thử. Ta cần đảm bảo được rằng khi tích hợp một thành phần vào tập hợp các thành phần đã được kiểm thử thì khả năng lỗi xảy ra sẽ rơi vào thành phần mới được tích hợp chứ không phải các thành phần đã được kiểm thử.

Sau khi đã đảm bảo các chức năng hoạt động riêng biệt của từng thành phần hoạt động đúng, ta sẽ tiến hành kiểm thử tích hợp giữa các thành phần. Quá trình này sẽ được lặp lại mỗi khi có sự thay đổi ở bất cứ thành phần nào (kiểm thử hồi quy). Trong chương này, luận văn sẽ đề xuất chiến thuật kiểm thử và xây dựng công cụ sinh mã kiểm thử tự động cho ứng dụng ESB, đồng thời xây dựng luồng triển khai liên tục cơ bản cho một ứng dụng ESB.

3.1. Hướng tiếp cận

3.1.1. Kiểm thử đơn vị

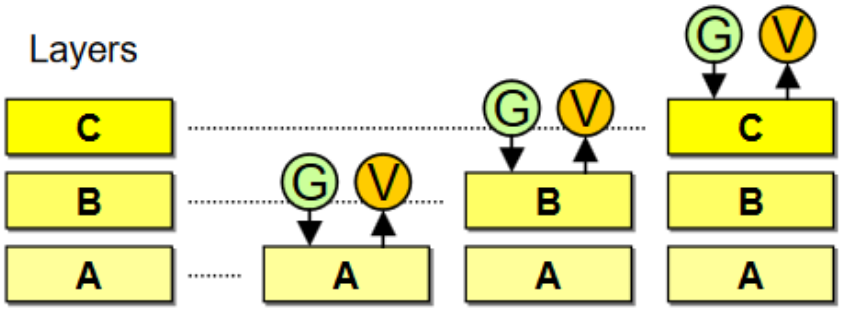
Kiểm thử đơn vị tập trung vào kiểm thử một thành phần nhất định trong hệ thống. Ta cần tách biệt các phụ thuộc bên ngoài khi kiểm thử riêng biệt một thành phần. Đối với thành phần ở lớp thấp nhất điều này là dễ dàng vì không có nhiều phụ thuộc với các thành phần khác. Vì vậy ta có thể sử dụng bộ sinh dữ liệu để tạo các ca kiểm thử và bộ kiểm chứng để đưa ra kết quả (xem Hình *3.1*, thành phần A). Đối với các thành phần ở lớp cao hơn, ta cần giả lập các thành phần phụ thuộc (thành phần B,C). Điểm thuận lợi của chiến thuật này đó là ta có thể kiểm thử độc lập các thành phần, và kiểm thử song song nhiều thành phần tại nhiều lớp. Kiểm thử đơn vị tập trung vào một bộ phận của hệ thống nên sẽ không thể phát hiện ra các vấn đề khi tích hợp các thành phần. Tuy nhiên, kiểm thử ở mức độ này đảm bảo rằng khi thực hiện kiểm thử tích hợp các chức năng của từng thành phần đã được kiểm thử đúng.



Hình 3.1: Kiểm thử đơn vị theo lớp *[14]*

3.1.2. Kiểm thử tích hợp

Kiểm thử tích hợp tập trung vào tương tác giữa các thành phần. Kiểm thử tích hợp tập trung vào thành phần tại lớp thấp nhất trước vì các thành phần này có ít các phụ thuộc ngoài nhất. Kiểm thử trước các lớp thấp hơn cũng làm giảm sự không rõ ràng khi kiểm thử các lớp cao hơn. Nếu một ca kiểm thử tích hợp thất bại ở các lớp cao, khả năng lỗi xảy ra ở các lớp thấp sẽ thấp hơn vì các lớp này đã được kiểm thử trước đó. Vì vậy, ta có thể khoanh vùng lỗi ở các thành phần mới hoặc tương tác giữa thành phần mới và các thành phần đã được kiểm thử. Chiến thuật này cũng có thể được gọi là kiểm thử từ dưới lên.



Hình 3.2: Kiểm thử tích hợp *[14]*

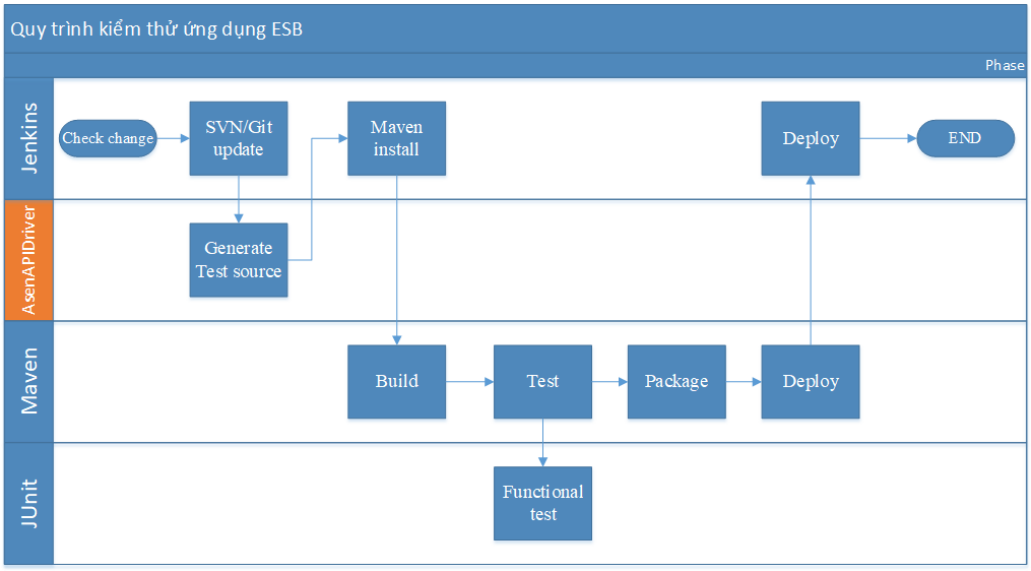
3.1.3. Kiểm thử hồi quy

Kiểm thử hồi quy [1] là việc kiểm thử lại một lần nữa chức năng do việc thay đổi chức năng khác có hoặc không có liên quan đến chức năng được kiểm thử.

Trong quá trình phát triển và bảo trì ứng dụng, việc sửa đổi một hay một vài chức năng dễ gây các lỗi tiềm ẩn ảnh hưởng đến chức năng khác. Vì vậy kiểm thử hồi quy là rất cần thiết trong trường hợp này, đặc biệt đối với các hệ thống ESB, những hệ thống mà việc thay đổi các ứng dụng khác nó có liên quan đến dễ gây ảnh hưởng tiềm ẩn.

3.2. Quy trình kiểm thử ứng dụng ESB

Hình 3.3 thể hiện quy trình đề xuất kiểm thử tự động cho hệ thống ESB.



Hình 3.3. Quy trình kiểm thử hệ thống ESB

Để kích hoạt quy trình, trên công cụ Jenkins, sử dụng một kích (trigger) có chức năng kích hoạt quá trình chạy tự động mỗi khi có thay đổi trên kho mã nguồn SVN hoặc git. Khi có sự thay đổi mã nguồn, Jenkins thực hiện lấy mã nguồn về máy chủ và gọi quá trình sinh mã kiểm thử tự động bằng công cụ. Sau khi sinh mã nguồn kiểm thử thành công, Jenkins tự động gọi quá trình biên dịch, chạy các ca kiểm thử, đóng gói và triển khai ứng dụng. Quá trình biên dịch và đóng gói ứng dụng được thực hiện bởi bộ thư viện Maven, đây là một thư viện mã nguồn mở để quản lý các thư viện phụ thuộc của phần mềm, cung cấp khả năng build và đóng gói phần mềm. Maven kích hoạt quá trình biên dịch mã nguồn, kiểm thử chức năng và kiểm thử đơn vị cho ứng dụng. Nếu quá trình kiểm thử thành công, Maven tự động đóng gói ứng dụng lên thư mục cài đặt sẵn. Từ đó, công cụ Jenkins sẽ triển khai ứng dụng lên máy chủ.

Do hệ thống dựa trên kiến trúc trục tích hợp chứa nhiều luồng xử lý và nhiều kết nối, nên việc viết các mã nguồn kiểm thử bằng tay gây tốn kém thời gian và dễ dẫn đến thiếu sót. Vì vậy, để đảm bảo quy trình diễn ra tự động luận văn này đề xuất thêm việc xây dựng công cụ thực hiện sinh mã nguồn kiểm thử chức năng cho ứng dụng xây dựng dựa trên nền tảng MuleESB, công cụ này có tên là AsenAPIDriver. AsenAPIDriver xây dựng trên nền tảng Java, có chức năng quét mã nguồn và danh sách các ca kiểm thử, từ đó sinh ra bộ mã nguồn kiểm thử tự động cho ứng dụng.

Như vậy, toàn bộ quá trình *kiểm* thử và triển khai này được thực hiện tự động bằng việc tích hợp các công cụ mã nguồn mở: Jenkins, Maven, JUnit, MUnit, Subversion và công cụ sinh mã kiểm thử tự động AsenAPIDriver.

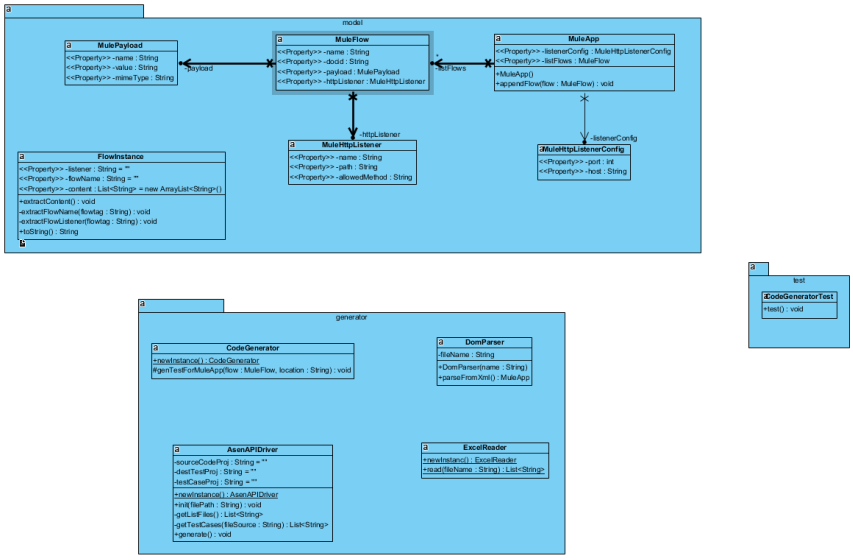
3.3. Xây dựng ứng dụng AsenAPIDriver

AsenAPIDriver được xây dựng trên nền tảng Java. Mục tiêu của thư viện là sinh ra các ca kiểm thử tự động cho các ứng dụng xây dựng dựa trên nền tảng MuleESB.

Ứng dụng AsenAPIDriver có chức năng đọc các luồng xử lý trong tập tin cấu hình ứng dụng MuleESB, kết hợp với danh sách các ca kiểm thử, thực hiện sinh các mã nguồn kiểm thử.

Việc chạy các ca kiểm thử qua AsenAPIDriver được quản lý và thực hiện bằng cách cấu hình qua Maven. Quá trình kiểm thử kết hợp sử dụng JUnit hỗ trợ việc quản lý và chiết xuất báo cáo kiểm thử.

Hình *3.4* mô tả biểu đồ lớp của ứng dụng AsenAPIDriver. AsenAPIDriver bao gồm 2 package chính. Package model chứa các lớp định nghĩa các đối tượng trong một ứng dụng Mule với các thuộc tính đi kèm. Package business chứa các lớp thực hiện các chức năng chính của ứng dụng là trích xuất đối tượng từ file XML, đọc thông tin các ca kiểm thử từ file excel và sinh ra các ca kiểm thử.



Hình 3.4. Biểu đồ lớp AsenAPIDriver

Các ứng dụng MuleESB định nghĩa các khối, các luồng trong tệp cấu hình xml. Trong đó mỗi thành phần tương ứng với một thẻ trong tệp xml. Dựa trên thông tin của các thẻ này, ta có thể lấy được thông tin của các thành phần trong ứng dụng.

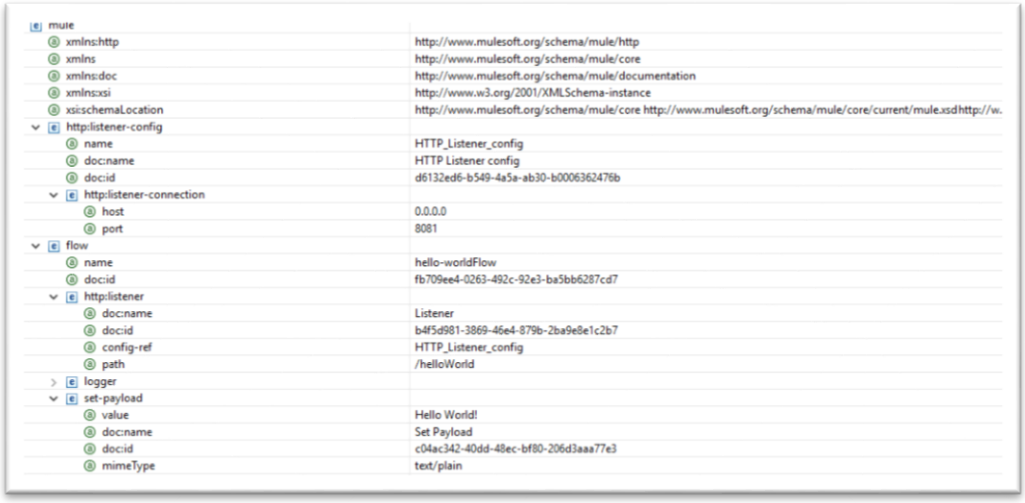
Hình *3.5* mô tả ví dụ tệp xml cấu hình của một ứng dụng MuleESB đơn giản. Ứng dụng này bao gồm một luồng nhận yêu cầu tại cổng 8081, trả về kết quả dạng text với giá trị là “Hello World!”.

```
<?xml version="1.0" encoding="UTF-8"?>

<mule xmlns:http="http://www.mulesoft.org/schema/mule/http" xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/
http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/mule/http/current/mule-http.xsd">
    <http:listener-config name="HTTP_Listener_config" doc:name="HTTP Listener config" doc:id="d6132ed6-b549-4a5a-ab30-b0006362476b" >
        <http:listener-connection host="0.0.0.0" port="8081" />
    </http:listener-config>
    <flow name="hello-worldFlow" doc:id="fb709ee4-0263-492c-92e3-ba5bb6287cd7" >
        <http:listener doc:name="Listener" doc:id="b4f5d981-3869-46e4-879b-2ba9e8e1c2b7" config-ref="HTTP_Listener_config" path="/helloWorld"/>
        <logger level="INFO" doc:name="Logger" doc:id="d254b372-8ea9-4b59-bb07-04cc2bf715a5" message="#[attributes.requestPath]"/>
        <set-payload value="Hello World!" doc:name="Set Payload" doc:id="c04ac342-40dd-48ec-bf80-206d3aaa77e3" mimeType="text/plain"/>
    </flow>
</mule>
```

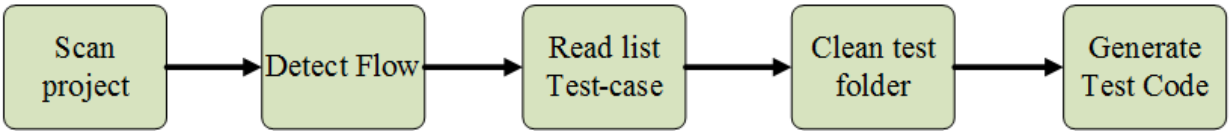
Hình 3.5. Ví dụ tệp XML cấu hình ứng dụng MuleESB

Hình *3.6* mô tả rõ hơn các nút tương ứng với các thuộc tính của các thẻ trong tệp tin cấu hình xml trên.



Hình 3.6. Các nút trong tệp xml

Các bước thực hiện sinh mã nguồn kiểm thử tự động dùng AsenAPIDriver (xem Hình *3.7*)



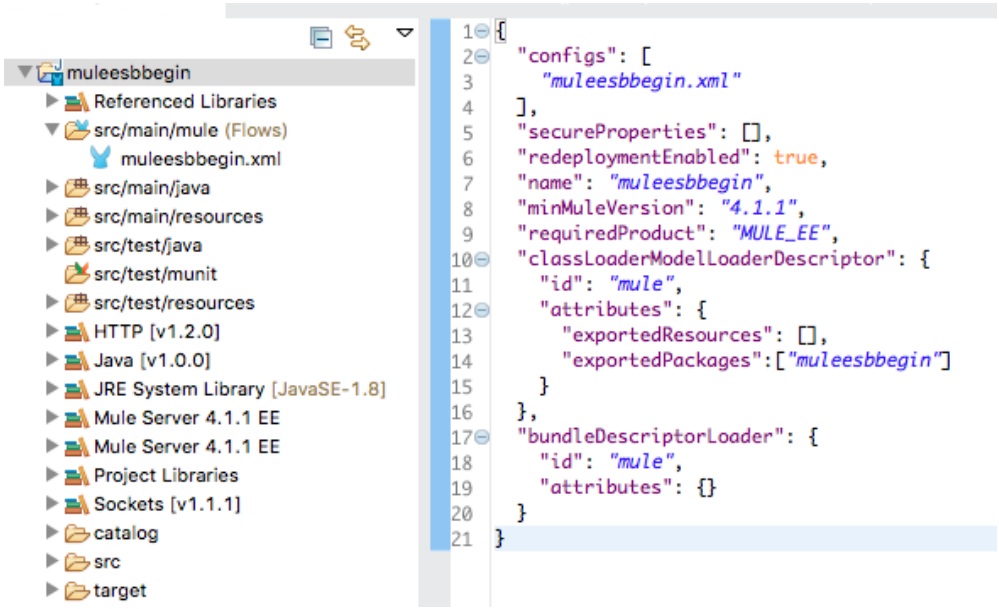
Hình 3.7: Các bước sinh mã kiểm thử tự động

Bước 1: Thư viện thực hiện quét bộ mã nguồn xây dựng trên nền tảng MuleESB, từ tập tin cấu hình chính của ứng dụng, công cụ xác định được đâu là nơi định nghĩa các luồng xử lý của ứng dụng. Hình *3.8* hiển thị tập tin cấu hình khởi tạo của ứng dụng MuleESB, tập tin này chỉ ra đâu là nơi chứa các luồng xử lý (ở đây là muleesbbegin.xml).

Bước 2: từ các tập tin định nghĩa các luồng xử lý, công cụ AsenAPIDriver thực hiện đọc và xác định các luồng, tên luồng, đường dẫn gọi vào từng luồng cụ thể. (xem Hình *3.8*).

Bước 3: Với mỗi luồng tương ứng, chương trình xác định và đọc các tập tin excel ca kiểm thử. Các tập tin này được cấu hình sẵn trong thư mục tài nguyên kiểm thử (test/resources) của ứng dụng.

Bước 4: Trước khi sinh ra mã nguồn kiểm thử tự động, công cụ thực hiện dọn dẹp thư mục mã nguồn kiểm thử.



Hình 3.8: Cấu hình khởi tạo của ứng dụng

Bước 5: Từ các luồng xác định và danh sách các ca kiểm thử, công cụ thực hiện sinh ra các mã nguồn kiểm thử tương ứng. Mỗi một luồng sẽ có một tập tin mã nguồn kiểm thử, số lượng phương thức và cách thức chạy ca kiểm thử phụ thuộc vào số lượng các ca kiểm thử, tuỳ vào từng luồng cụ thể. Hình 3.9 hiển thị một lớp kiểm thử được sinh tự động từ công cụ AsenAPIDriver.

```
import org.mule.api.MuleEvent;
import org.mule.munit.runner.functional.FunctionalMunitSuite;
import org.junit.Assert;
import org.junit.Test;

public class userprofile extends FunctionalMunitSuite {
    @Test
    public void Test1() throws Exception {
        String myStringPayload = "{\"requestId\":\"123580\",\"username\":\"anh.phammai\"}";
        String myStringOutput = "{\"requestId\":\"123580\",\"username\":\"anh.phammai\",\"fullName\":\"Pham Mai Anh\",\"age\":\"20.00\",\"date
        MuleEvent resultMuleEvent = runFlow("user-profile",testEvent(myStringPayload) );
        Assert.assertEquals(myStringOutput, resultMuleEvent.getMessage().getPayload() );
    }
    @Test
    public void Test2() throws Exception {
        String myStringPayload = "{\"requestId\":\"235813\",\"username\":\"dinhvung\"}";
        String myStringOutput = "{\"requestId\":\"235813\",\"username\":\"dinhvung\",\"fullName\":\"Vu Dinh Vung\",\"age\":\"23.00\",\"dateOfB
        MuleEvent resultMuleEvent = runFlow("user-profile",testEvent(myStringPayload) );
        Assert.assertEquals(myStringOutput, resultMuleEvent.getMessage().getPayload() );
    }
}
```

Hình 3.9: Mã nguồn kiểm thử bằng phương pháp sinh tự động

Mã nguồn kiểm thử được sinh bởi AsenAPIDriver là mã nguồn sử dụng MUnit để gọi các luồng và truyền vào các tham số cho trước. Kết quả trả ra được so sánh với kết quả đầu ra mong đợi lấy từ thư mục test/resources.

Các đoạn mã nguồn sử dụng MUnit được chú thích bằng các ký pháp của JUnit, quá trình chạy các đoạn mã nguồn kiểm thử cho ra kết quả ngay tại màn hình IDE và được xuất thành báo cáo.

CHƯƠNG 4. THỰC NGHIỆM

Trong chương này, luận văn sẽ xây dựng một hệ thống phần mềm nhỏ dựa vào MuleESB như một ví dụ. Từ ứng dụng đó, dựa vào quy trình thực hiện ở phần trước, luận văn sẽ đưa ra cách thức cài đặt, sinh mã kiểm thử và tích hợp liên tục hoàn chỉnh. Thông qua phần cài đặt và triển khai, luận văn sẽ đánh giá những kết quả đạt được và những điểm cần phải bổ sung.

4.1. Ứng dụng MuleESB mẫu

Để kiểm tra thực nghiệm quy trình kiểm thử ở chương trước, luận văn xây dựng một ứng dụng ESB trên nền tảng MuleESB. Ứng dụng có tên IB-ESB, là một ứng dụng ngân hàng điện tử, có chức năng cung cấp các đầu dịch vụ(end-point) cho các ứng dụng phía ngoài như sau: tra cứu thông tin doanh nghiệp, tra cứu thông tin người dùng, chuyển khoản trong hệ thống, vấn tin tài khoản, vấn tin lịch sử giao dịch.

Chức năng tra cứu thông tin doanh nghiệp có đầu vào là thông tin định danh của doanh nghiệp trong ngân hàng (cifno), từ thông tin cifno, hệ thống vấn tin vào hệ thống lõi của ngân hàng (corebank), kết hợp với thông tin đăng ký dịch vụ ngân hàng điện tử trên cơ sở dữ liệu của hệ thống IB, ứng dụng IB-ESB sẽ trả ra thông tin doanh nghiệp bao gồm: Số cifno, tên doanh nghiệp, địa chỉ, mã số thuế, ngày đăng ký dịch vụ ngân hàng điện tử, trạng thái đăng ký dịch vụ.

Chức năng tra cứu thông tin người dùng yêu cầu thông tin đầu vào là tên đăng nhập của người dùng, từ đó ứng dụng trả ra thông tin tương ứng cho người dùng bao gồm: tên đăng nhập, họ và tên đầy đủ, ngày sinh, chức vụ, định danh và tên doanh nghiệp, số chứng minh nhân dân, ngày đăng ký và trạng thái đăng ký ngân hàng điện tử.

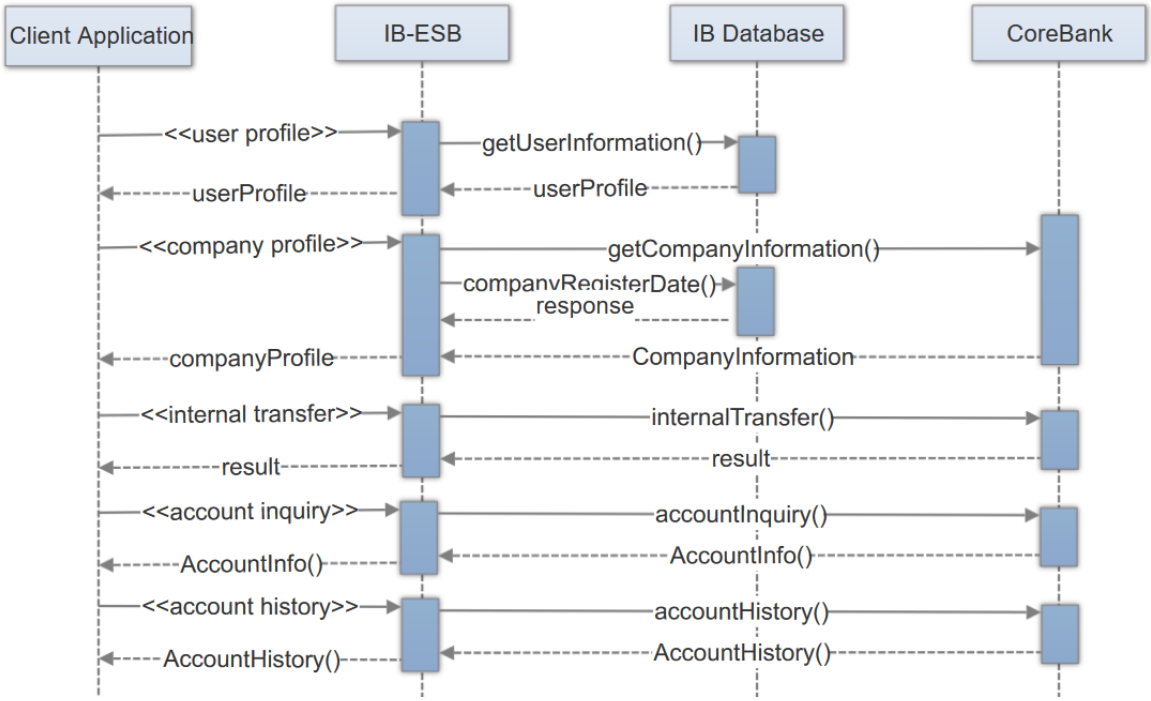
Chức năng chuyển khoản trong hệ thống cho phép khách hàng thực hiện chuyển tiền vào tài khoản mở cùng ngân hàng, đầu vào yêu cầu của dịch vụ bao gồm: tài khoản chuyển, số tiền chuyển, tài khoản nhận, tên tài khoản nhận và nội dung giao dịch. Đầu ra của dịch vụ xác nhận khách hàng chuyển khoản thành công cùng thông tin phí hoặc các thông báo lỗi tương ứng.

Vấn tin tài khoản trả ra thông tin số tài khoản, số dư, chi nhánh mở tài khoản, trạng thái tài khoản. Đầu vào yêu cầu của dịch vụ là thông tin số tài khoản.

Lịch sử giao dịch là đầu dịch vụ tra cứu lịch sử giao dịch bao gồm một danh sách các giao dịch ghi có/ghi nợ của tài khoản trong một khoảng thời gian. Đầu vào yêu cầu của dịch

vụ bao gồm số tài khoản, khoảng thời gian lấy lịch sử, khoảng thời gian không quá 31 ngày kể từ ngày hiện tại.

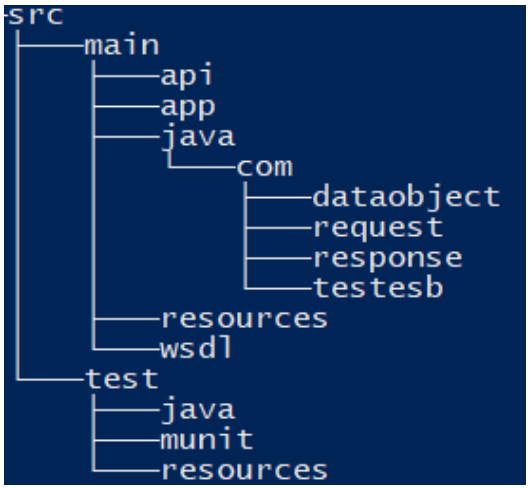
Hình 4.1 mô tả luồng xử lý ứng dụng IB-ESB.



Hình 4.1: Sơ đồ tuần tự ứng dụng IB-ESB

Cách thức tổ chức mã nguồn của ứng dụng mẫu

Mã nguồn của ứng dụng mẫu được chia thành các gói (package), các thư mục tương ứng với mã nguồn mẫu, mã nguồn ca kiểm thử, như trong *Hình 4.2*

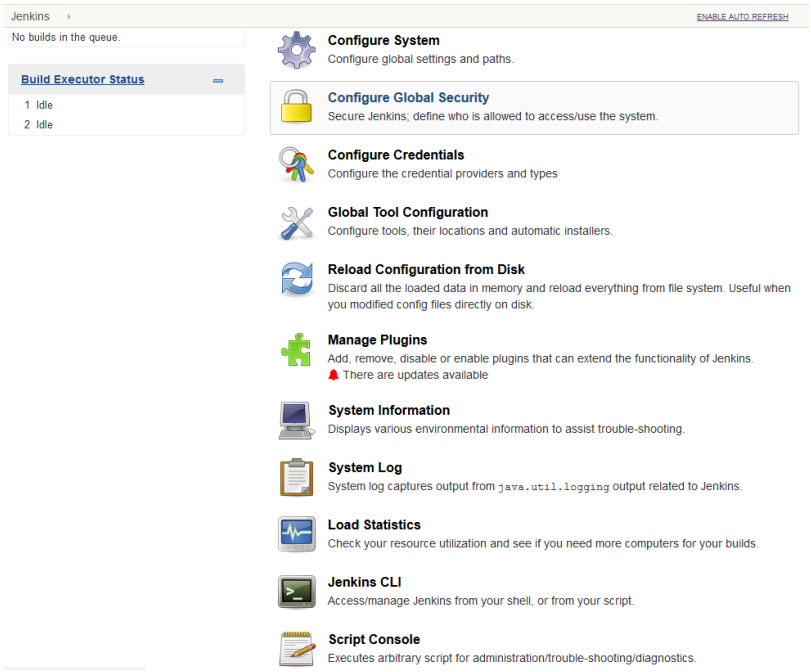


Hình 4.2: Cách phân chia thư mục trên ứng dụng MuleESB

Theo đó, mã nguồn của dự án được quản lý trên kho quản lý mã nguồn github tại đường dẫn https://github.com/Loandt1/TestMuleESB.git. Các thư viện phụ thuộc của ứng dụng được quản lý bằng maven.

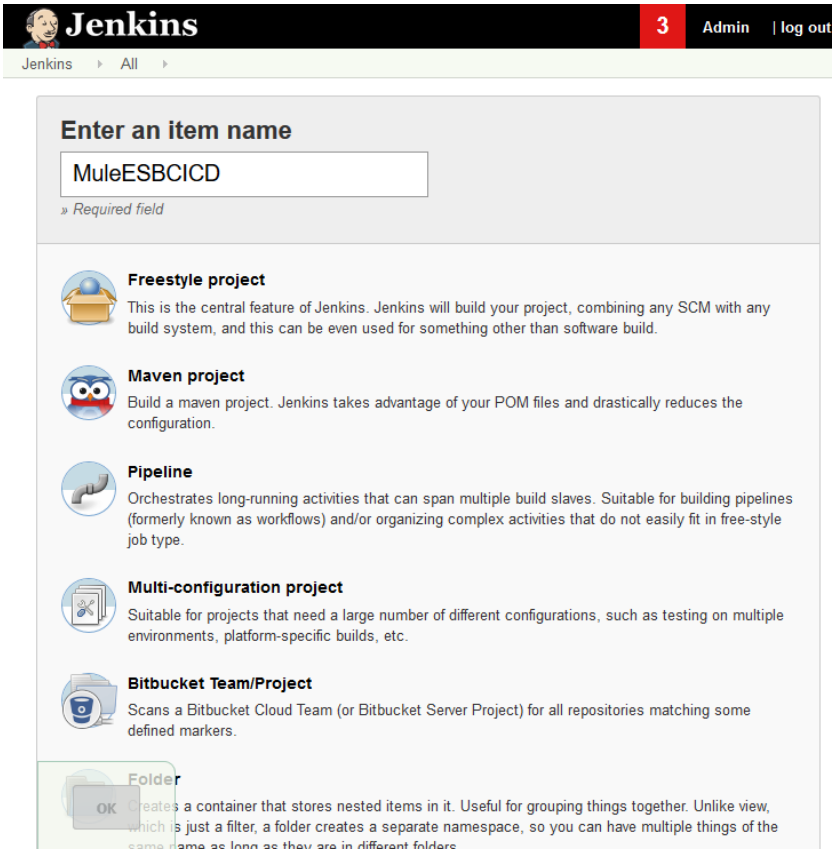
4.2. Tích hợp quy trình kiểm thử

Bước 1: Tại màn hình chính, chọn "New Item"



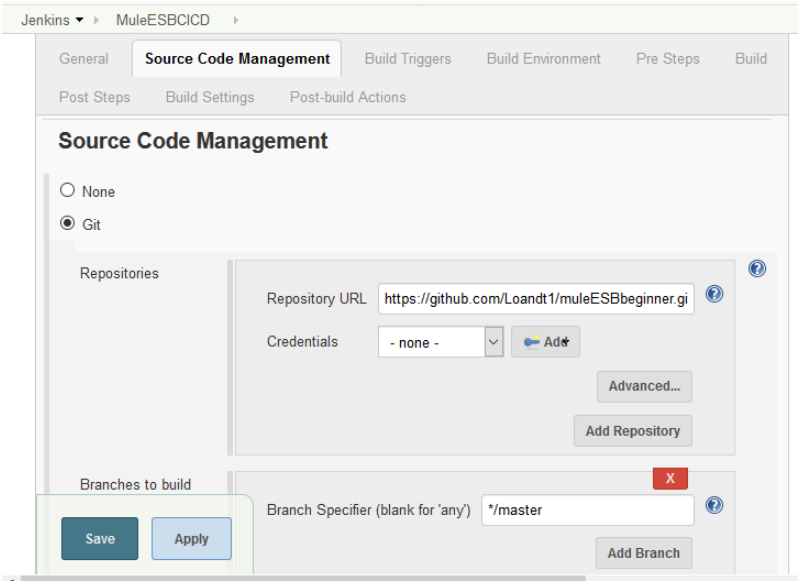
Hình 4.3: Màn hình quản lý của Jenkins

Bước 2: tại màn hình tạo mới (Hình *4.4***Error! Reference source not found.**), chọn tên ứng dụng và loại ứng dụng, ở đây, ta chọn "Maven Project".



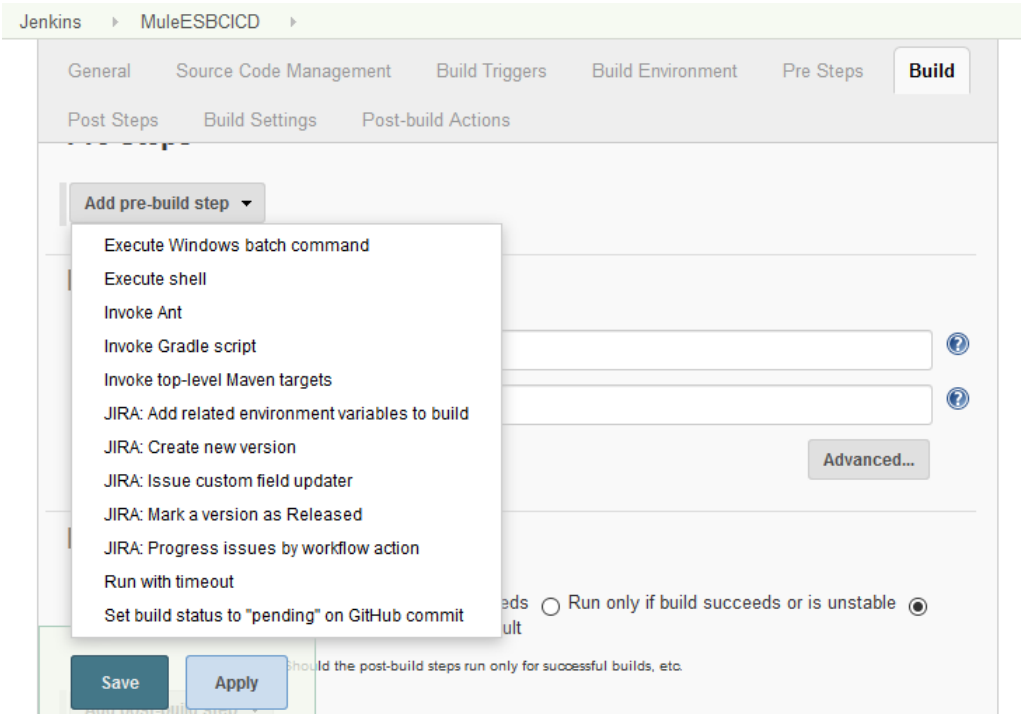
Hình 4.4: Tạo một tác vụ trên Jenkins

Bước 3: nhập thông tin cấu hình cho ứng dụng bao gồm: kho mã nguồn, môi trường dịch, các bước dịch,… (Hình *4.5***Error! Reference source not found.**)



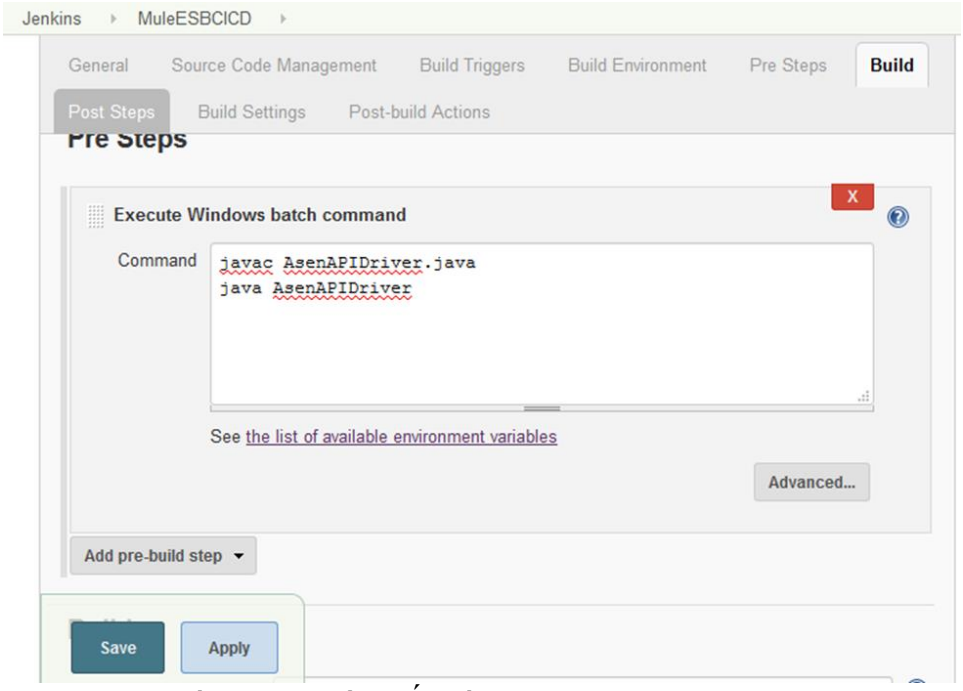
Hình 4.5: Thông tin chi tiết cấu hình tác vụ

Tại màn hình cấu hình (Hình *4.6*), Jenkins cho phép cấu hình thêm các câu lệnh để hỗ trợ quá trình dịch qua shell.



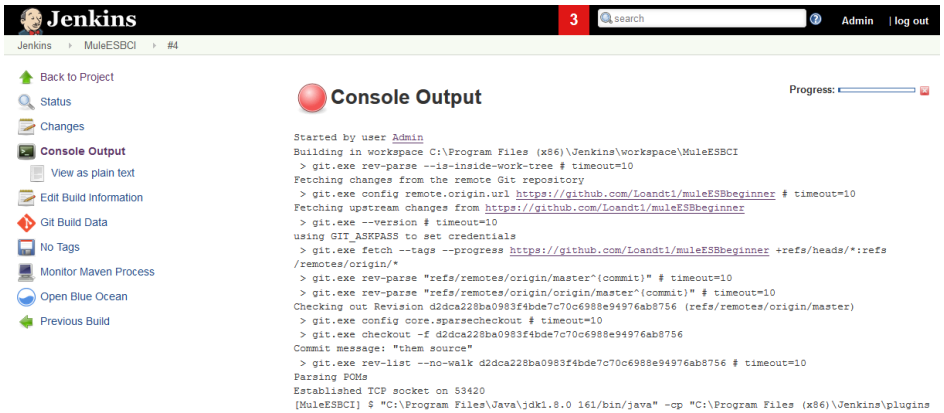
Hình 4.6: Tùy chọn tác vụ xử lý qua shell

Tại đây, ta chọn "window command shell" để tích hợp với quá trình sinh mã nguồn kiểm thử tự động từ thư viện AsenAPIDriver (Hình *4.7***Error! Reference source not found.**)



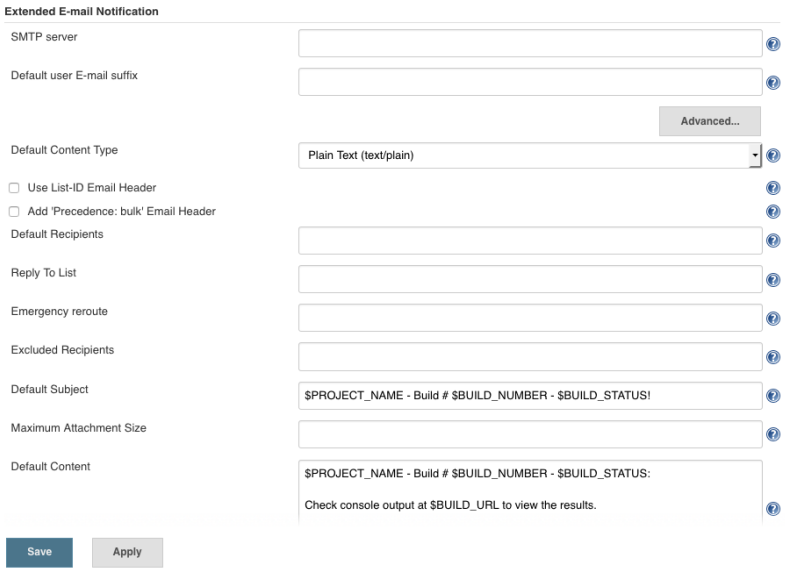
Hình 4.7:Thêm cấu hình gọi AsenAPIDriver

Bước 4: Sau khi thực hiện lưu các cấu hình, ta có thể chọn "Build now" để thực hiện chạy tác vụ (Hình 4.8**Error! Reference source not found.**)



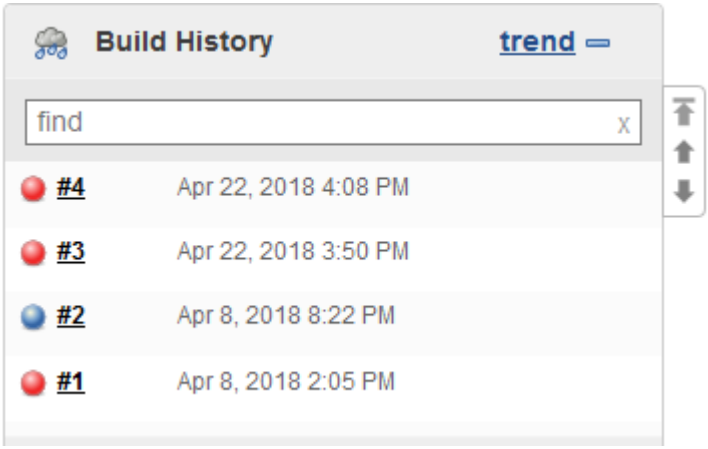
Hình 4.8: Quá trình chạy tác vụ

Trong quá trình thực hiện tác vụ, khi xảy ra lỗi, Jenkins tự động gửi mail cảnh báo về theo cấu hình định sẵn. Để có thể gửi mail thông báo lỗi, các cấu hình máy chủ mail phải được điền trong màn hình quản lý của Jenkins tại đường dẫn: "Manage Jenkins => Configure System => E-mail Notification" (Hình *4.9*). Jenkins cho phép cấu hình các thông tin cơ bản như máy chủ SMTP, hậu tố của email nhận cảnh báo , ví dụ như @gmail.com. Ngoài ra, Jekins còn cung cấp thêm các giải pháp xác thực sử dụng SMTP, SSL, cổng SMTP, địa chỉ để người nhận mail gửi lại phản hồi… Sau khi thêm một "E-mail notification Post-build Action", Jenkins sẽ gửi một mail tới địa chỉ người dùng đã cấu hình từ trước đối với các trường hợp như: biên dịch thất bại (build fail), biên dịch thành công sau khi có một lần biên dịch thất bại, một bản biên dịch không ổn định (với trường hợp hồi quy).



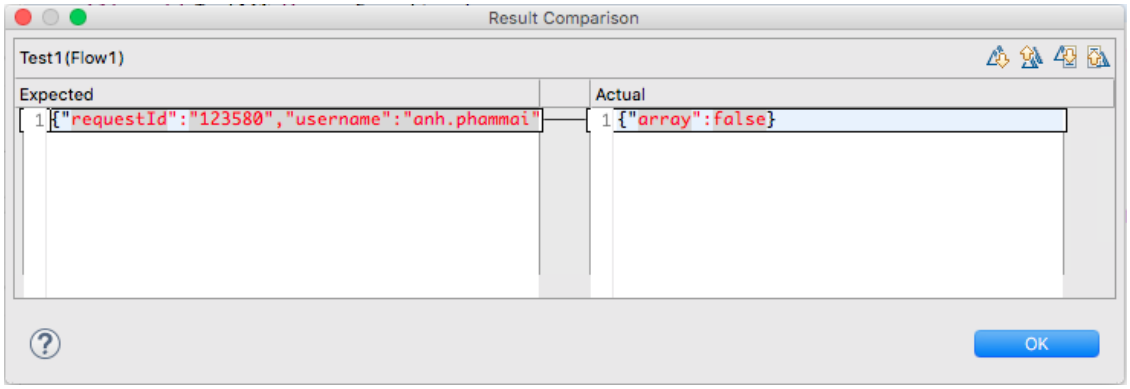
Hình 4.9: Cấu hình thông báo Email

Sau quá trình chạy tác vụ, kết quả được hiển thị tại màn hình ứng dụng (xem Hình 4.10**Error! Reference source not found.**)



Hình 4.10: Lịch sử chạy tác vụ

Ngoài các bước được trình bày ở trên, có rất nhiều cách để tạo công việc xây dựng, các tùy chọn có sẵn rất nhiều, điều khiển Jenkins trở thành một công cụ triển khai liên tục. Hình *4.11* là kết quả kiểm thử khi chạy mã nguồn kiểm thử tự động sinh



Hình 4.11: Kết quả thực hiện chạy mã nguồn kiểm thử tự sinh

4.3. Đánh giá kết quả

Qua việc áp dụng quy trình tích hợp liên tục trên công cụ Jenkins, kết hợp với công cụ AsenAPIDriver tự động sinh ca kiểm thử, ta có thể thấy rõ những ưu điểm mà nó mang lại.

Giảm thời gian kiểm thử mà vẫn đảm bảo được chất lượng của mã nguồn. Bằng cơ chế liên kết giữa jenkins và github, bất cứ một sự thay đổi nào về mã nguồn đều được kiểm tra lại với các ca kiểm thử ngay lập tức, sau đó thông báo đến đội phát triển thông qua email. Điều này giúp cho quy trình phát triển được tự động hóa và khép kín hơn.

Ngoài ra, hiệu năng làm việc của đội phát triển cũng sẽ được cải thiện đáng kể. Với việc các thay đổi đều được kiểm tra liên tục, các vấn đề xảy ra sẽ sớm được phát hiện, thông báo lại để sớm tìm giải pháp khắc phục, hạn chế được các lỗi tiềm tàng khi triển khai.

KẾT LUẬN

Kết quả đạt được

Luận văn tìm hiểu cơ sở lý thuyết về hệ thống xây dựng dựa trên kiến trúc trục tích hợp cũng như cách xây dựng và tích hợp công cụ sinh mã kiểm để giảm thiểu chi phí kiểm thử, đảm bảo tốt hơn chất lượng hệ thống ở pha lập trình so với việc kiểm thử bằng tay sau pha lập trình, đồng thời tăng hiệu năng làm việc của quá trình phát triển, tránh những lỗi tiềm tảng ở pha triển khai.

Mục tiêu chính của luận văn là xây dựng công cụ sinh mã kiểm thử chức năng tự động cho hệ thống xây dựng dựa trên kiến trúc trục tích hợp, kết hợp với các công cụ quản lý dự án tự động, từ đó tự động hóa toàn bộ quá trình kiểm thử và triển khai ứng dụng lên máy chủ.

Quy trình kiểm thử hệ thống được nêu trong luận văn hỗ trợ việc kiểm thử phần mềm, rút ngắn được thời gian và kiểm soát tốt các lỗi xảy ra đối với hệ thống. Quá trình kiểm thử diễn ra tự động liên tục giúp giảm thiểu thời gian lập trình cho lập trình viên, các lỗi được phát hiện ra sớm trong pha lập trình trước khi khi triển khai trên môi trường thật, đặc biệt là các khiếm khuyết do các phần chỉnh sửa liên đới tới nhau, các lỗi này đối với kiểm thử thông thường thủ công dễ bị bỏ qua.

Sau quá trình thực hiện, luận văn đã hoàn thiện quá trình kiểm thử tích hợp và tự động hệ thống xây dựng dựa trên nền tảng MuleESB bằng việc xây dựng công cụ sinh ca kiểm thử tự động, kết hợp với các công cụ mã nguồn mở như Jenkins, Github, Junit, Munit, Maven giúp giảm thiểu được thời gian phát triển ứng dụng, đảm bảo không xảy ra lỗi, thiếu sót trong quá trình phát triển ứng dụng.

Điểm hạn chế và hướng phát triển tiếp

Việc xây dựng và tích hợp công cụ kiểm thử ứng dụng AsenAPIDriver đã đáp ứng được những yêu cầu cơ bản nhất được đặt ra cho các hệ thống xây dựng trên nền tảng MuleESB. Tuy nhiên, do những giới hạn về thời gian và độ dài, một số vấn đề còn chưa được đề cập tới trong luận văn.

Đầu tiên, công cụ xây dựng mới chỉ thực hiện sinh các ca kiểm thử cho một số luồng cơ bản của nền tảng MuleESB và cũng mới chỉ dừng lại ở mức hỗ trợ nền tảng này. Ngoài ra, luận văn mới chỉ hỗ trợ sinh các ca kiểm thử chức năng, chưa bao quát được các ca kiểm thử phi chức năng về bảo mật cũng như hiệu năng của hệ thống.

Trong tương lai, để phục vụ được việc kiểm thử trên các hệ thống dựa trên các nền tảng kiến trúc trục tích hợp khác, tác giả cần xây dựng công cụ tích hợp được thành plugin trên IDE và hỗ trợ sinh các ca kiểm thử cho các nền tảng trục tích hợp khác như: ServiceMix, JbossESB... cũng như đáp ứng các yêu cầu kiểm thử về bảo mật, hiệu năng

của hệ thống. Đồng thời kết hợp nghiên cứu các quy trình phần mềm mới trong tương lai để đưa ra các chiến thuật kiểm thử tốt hơn.

TÀI LIỆU THAM KHẢO

[1] Dirk Slama, Dirk Krafzig and Karl Banke, Enterprise SOA: Service-Oriented Architecture Best Practices, Prentice Hall, 2004.

[2] Pulier, E., Taylor, Understanding Enterprise SOA, Manning, 2016, p. 2006.

[3] Dirk Krafzig, "Enterprise SOA: Service-Oriented Architecture Best Practices," 2005.

[4] Falko Menge, "Enterprise Service Bus," *Free and open source software conference,* 2007.

[5] Srinivas Shenoy, ""“Approach to ESB Testing” – An Experience Sharing," 2013.

[6] MuleSoft, "What is MuleESB," [Online]. Available: https://www.mulesoft.com/resources/esb/what-mule-esb.

[7] Mule, "Mule," [Online]. Available: https://www.mulesoft.com/.

[8] Gartner, "Gartner Magic quandrad leade," [Online]. Available: https://www.mulesoft.com/lp/reports/gartner-magic-quadrant-leader.

[9] Martin Fowler, "Continuous Integration," [Online]. Available: https://www.martinfowler.com/articles/continuousIntegration.html.

[10] Git, "Getting started - About version control," [Online]. Available: https://git-scm.com/book/en/v1/Getting-Started-About-Version-Control.

[11] Glenford J. Myers, Corey Sandler and Tom Badgett, The Art of Software Testing 3rd Edition, 26 September 2015.

[12] Phạm Ngọc Hùng, Trương Anh Hoàng and Đặng Văn Hưng, Giáo trình kiểm thử phần mềm, 2014.

[13] SoapUI, "SoapUI Getting started," [Online]. Available: https://www.soapui.org/soap-and-wsdl/getting-started.html.

[14] Gregor Hohpe and Wendy Istanick, "Test-Driven Development in Enterprise Integration Projects," 2002.

[15] Maven, "Maven," [Online]. Available: https://maven.apache.org/.

[16] JUnit, "JUnit," [Online]. Available: https://junit.org/junit5/ .

[17] Jenkins, "Jenkins," [Online]. Available: https://jenkins.io/.

[18] Gregor Hohpe and Bobby Woolf, Enterprise Integration Patterns, Pearson Education, 2004.

[19] David A. Chappell, Enterprise Service Bus, O’Reilly, 2004.

[20] Git, "Getting started - About version control," [Online]. Available: https://git-scm.com/book/en/v1/Getting-Started-About-Version-Control.