HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA VIỄN THÔNG I

ĐỒ ÁN

TỐT NGHIỆP ĐẠI HỌC

ĐỀ TÀI:

"KIẾN TRÚC RESTFUL API VÀ ỨNG DỤNG TRONG HỆ THỐNG IOT"

Giảng viên hướng dẫn	**:**	**TS. NGUYỄN TRỌNG TRUNG ANH**
Sinh viên thực hiện	**:**	**ĐỖ TRUNG ĐẠT**
Lớp	**:**	**D19VTHI02**
Mã sinh viên	**:**	**B19DCVT077**
Khóa	**:**	**2019 – 2024**
Hệ	**:**	**ĐẠI HỌC CHÍNH QUY**

HÀ NỘI – 12/2023

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

Điểm: (Bằng chữ:)

Hà Nội, ngày tháng 12 năm 2023

Giáo viên hướng dẫn

TS. Nguyễn Trọng Trung Anh

NHẬN XÉT CỦA GIẢNG VIÊN PHẢN BIỆN

.....

.....

.....

.....

.....

.....

.....

.....

Điểm: (Bằng chữ:)

Hà Nội, ngày tháng năm 2024

LỜI CẢM ƠN

Đồ án tốt nghiệp là kết quả của quá trình cố gắng không ngừng nghỉ của bản thân và nhận được sự hỗ trợ, giúp đỡ cũng như sự quan tâm, động viên từ nhiều cơ quan, tổ chức và cá nhân. Qua đây, em xin gửi lời cảm ơn chân thành tới những người đã giúp đỡ em hoàn thành đồ án này.

Trước tiên em xin cảm ơn thầy **TS. Nguyễn Trọng Trung Anh**, người thầy đã nhiệt tình, ân cần chỉ bảo, hướng dẫn giúp em hoàn thành đồ án này.

Em cũng xin cảm ơn các thầy cô trong ban lãnh đạo Học viện, các thầy cô giảng dạy và đặc biệt hơn là thầy cô trong Khoa Viễn Thông I đã tạo điều kiện cho em trong chặng đường dài học tập và rèn luyện.

Do kiến thức còn hạn hẹp nên không thể tránh được những thiếu sót về cách hiểu lẫn cách trong trình bày. Em rất mong nhận được sự đóng góp ý kiến của quý thầy, cô để hiểu rõ và đồ án của em được tốt hơn.

Em xin chân thành cảm ơn.

Hà Nội, ngày 27 tháng 12 năm 2023

Sinh viên thực hiện

Đạt

Đỗ Trung Đạt

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

THUẬT NGỮ VIẾT TẮT

Từ viết tắt	Tiếng Anh	Giải thích
RAM	Random Access Memory	Bộ nhớ truy cập tạm thời
SDLC	Software Development Life Cycle	Chu trình phát triển phần mềm
DSD	Design Structure Diagram	Sơ đồ kiến trúc thiết kế
RAD	Rapid Application Development	Phát triển ứng dụng nhanh chóng
URL	Uniform Resource Locator	Hệ thống định vị tài nguyên thống nhất
REST	Representational State Transfer	Chuyển đổi cấu trúc dữ liệu
WWW	World Wide Web	Mạng lưới toàn cầu
HTTP	HyperText Transfer Protocol	Giao thức truyền tải siêu văn bản
HTML	HyperText Markup Language	Ngôn ngữ đánh dấu siêu văn bản
URI	Uniform Resource Identifier	Định dạng tài nguyên thống nhất
TLS	Transport Layer Security	Bảo mật lớp truyền tải
API	Application Programming Interface	Giao diện lập trình ứng dụng
IDE	Integrated Development Environment	Môi trường tích hợp dùng để viết code để phát triển ứng dụng

LỜI MỞ ĐẦU

Ngày nay, trong bối cảnh cuộc cách mạng công nghệ 4.0 đang diễn ra toàn cầu, sự kết nối giữa con người và máy móc đã không còn là điều xa vời. Cùng với sự bùng nổ của Internet of Things (IoT), việc truyền thông và giao tiếp giữa các thiết bị đã trở nên không thể tránh khỏi. Trong bối cảnh này, kiến thức về kiến trúc RESTful API và ứng dụng trong hệ thống IoT đang trở thành một chủ đề nóng hổi và đầy tiềm năng.

Trong đồ án này em tập trung vào nghiên cứu về kiến trúc RESTful API và cách chúng được tích hợp vào hệ thống IoT. RESTful API, viết tắt của Representational State Transfer, đang trở thành một trong những tiêu chuẩn phổ biến cho việc truy cập và giao tiếp giữa các thiết bị và ứng dụng trên mạng. Sự linh hoạt và khả năng tương tác của RESTful API đã tạo ra những cơ hội đáng kể cho việc phát triển các ứng dụng IoT hiện đại. Trong đồ án này, em sẽ tập trung vào việc phân tích, thiết kế và triển khai các RESTful API trong ngữ cảnh của hệ thống IoT. Em cũng sẽ xem xét các thách thức và cơ hội mà việc tích hợp RESTful API vào IoT mang lại.

Mục tiêu chính của đồ án này là nghiên cứu sâu hơn về kiến trúc RESTful API, hiểu rõ về cách chúng tương tác với các thiết bị IoT và ứng dụng thực tế của chúng trong việc giải quyết các vấn đề trong hệ thống IoT. Đồng thời, em cũng mong muốn chia sẻ kiến thức và kinh nghiệm thu được từ quá trình nghiên cứu này với cộng đồng, giúp tăng cường nhận thức về sức mạnh và tiềm năng của việc sử dụng kiến trúc RESTful API trong lĩnh vực IoT.

Nội dung đồ án gồm 3 chương:

Chương 1: Tổng quan về phát triển phần mềm.

Chương 2: Kiến trúc REST và RESTful API.

Chương 3: Ứng dụng của RESTful API cho dữ liệu Iot.

CHƯƠNG 1 - TỔNG QUAN VỀ PHÁT TRIỂN PHẦN MỀM

1.1. Lịch sử phát triển phần mềm

1.1.1. Khái niệm phần mềm

Phần mềm là tập hợp dữ liệu hoặc các câu lệnh hướng dẫn, điều khiển cho máy tính để máy tính có thể làm việc theo cách mà bạn muốn. Và người đầu tiên đưa ra lý thuyết về phần mềm vào năm 1935 là Alan Turing (cha đẻ của ngành khoa học máy tính và công nghệ phần mềm).

Alan Turing sinh ngày 23/6/1912 tại Paddington, London, nước Anh. Từ những năm 1920, Hitler đã tập hợp nhiều chuyên gia để tạo ra loại máy trao đổi thông tin mật có tên là Enigma. Hitler ca ngợi mã Enigma là "Mật mã số một thế giới, cả đến thần thánh cũng không giải nổi". Năm 1939, Turing dùng trí tuệ của mình để giải mã Enigma góp phần vào cuộc kháng chiến của nhân dân Anh chống phát xít Đức xâm lược. Turing đã sáng chế ra một chiếc máy tính cơ-điện tử giúp giải mã Enigma và được đặt tên là Bombe chiếc máy tính hiện đại đầu tiên của nhân loại. Nhờ vậy mà mọi nội dung mật điện của Đức đều bị quân đội Anh nắm được, tạo ra thế chủ động cho phía nước Anh. Do vậy mà Thế Chiến thứ II đã được kết thúc sớm trước hai năm và nếu không có ông cục diện chiến tranh lúc đó sẽ khác hẳn. Alan Turing chính là người đặt nền móng cho ngành khoa học máy tính và công nghệ phần mềm nhờ vậy mà chúng mới có được những phát triển mạnh mẽ của công nghệ sau này.

1.1.2. Các giai đoạn quan trọng

Năm 1975, Tạp chí Popular Electronics vào số tháng một đã giới thiệu máy tính Altair 8080 dưới biệt danh "máy tính mini đầu tiên trên thế giới". Chiếc máy tính này không có chuột mà điều khiển thông qua các công tắc hoặc nhập liệu bằng bàn phím. Hai người là Paul Allen và Bill Gates đã đề nghị được viết phần mềm cho Altair bằng ngôn ngữ lập trình BASIC. Vào tháng 4, Paul Allen và Bill Gates đã thành lập nên công ty phần mềm **Microsoft** là hãng sản xuất phần mềm lớn nhất thế giới.

Năm 1977, Steve Jobs và Steve Wozniak thành lập **Apple** và giới thiệu máy tính Apple II tại triển lãm máy tính West Coast. Apple II mang tới trải nghiệm đồ hoạ màu sắc, âm thanh, có bộ nhớ RAM và hệ điều hành Apple DOS. Apple một trong

năm công ty lớn của ngành công nghệ thông tin Hoa Kỳ chuyên thiết kế, phát triển và bán thiết bị điện tử tiêu dùng, phần mềm máy tính và các dịch vụ trực tuyến. Có thể kể đến các sản phẩm nổi tiếng như iPod, iPhone, iPad, và các dịch vụ như iTunes, App Store và iCloud, định hình cách chúng ta tương tác với công nghệ trong thế giới hiện đại.

Năm 1998, Larry Page và Sergey Brin thành lập **Google** là một công ty công nghệ đa quốc gia của Mỹ, chuyên về các dịch vụ và sản phẩm liên quan đến Internet, bao gồm các công nghệ quảng cáo trực tuyến, điện toán đám mây, phần mềm, phần cứng và nổi tiếng nhất là công cụ tìm kiếm phổ biến nhất thế giới Google. Nó giúp tìm kiếm thông tin từ hàng ngàn trang web chỉ bằng cách nhập một vài từ khóa. Một số ứng dụng khác như Gmail, Google Maps, Youtube … những phần mềm giúp ích rất nhiều trong cuộc sống của con người ngày nay.

Năm 2004, **Facebook** là phương tiện truyền thông xã hội và dịch vụ mạng xã hội trực tuyến lớn nhất thế giới thành lập vào năm 2004 của Mỹ thuộc sở hữu của Meta Platforms có trụ sở tại Menlo Park, California. Nó được Mark Zuckerberg, cùng với các sinh viên Đại học Harvard và bạn cùng phòng là Eduardo Saverin, Andrew McCollum, Dustin Moskovitz, Chris Hughes sáng lập. Facebook ra đời nhằm kết nối tất cả mọi người trên toàn thế giới lại với nhau. Khi dùng ứng dụng này, bạn hoàn toàn có thể gặp gỡ hàng trăm, hàng nghìn bạn bè trên toàn thế giới thông qua mạng Internet. Giống như với mạng Internet thì Facebook sẽ giúp mọi người xóa bỏ mọi khoảng cách địa lý. Bạn có thể đăng và chia sẻ trạng thái, cập nhật hồ sơ cá nhân cũng như tương tác với người khác. Bên cạnh đó, trên mạng xã hội còn có rất nhiều tiện ích khác như bạn sử dụng Facebook để bán hàng và kiếm tiền một cách hiệu quả.

Đây là một trong những người sáng lập và định hình ngành công nghiệp phần mềm hiện đại, đã có vai trò quan trọng trong việc thay đổi cách tương tác với công nghệ. Họ đã đóng góp không nhỏ vào sự phát triển và thay đổi của lĩnh vực này, từ việc xây dựng những ứng dụng, phần mềm tiên tiến đến việc thúc đẩy sự đổi mới và tạo ra xu hướng mới.

Hình 1.1: Những người tiên phong của ngành công nghiệp phần mềm hiện đại

1.2. Quy trình phát triển phần mềm

Cũng như mọi ngành sản xuất khác, quy trình là một trong những yếu tố cực kỳ quan trọng đem lại sự thành công cho các nhà sản xuất phần mềm, nó giúp mọi thành viên trong dự án từ người cũ cho đến người mới, trong hay ngoài công ty đều có thể xử lý đồng bộ công việc tương ứng vị trí của mình qua cách thức chung của công ty, hay ít nhất ở cấp độ dự án.

Có thể nói quy trình phát triển/xây dựng phần mềm(Software Development Life Cycle) có tính chất quyết định để tạo ra sản phẩm chất lượng tốt với chi phí thấp và năng suất cao, điều này có ý nghĩa quan trọng đối với các công ty sản xuất hay gia công phần mềm củng cố và phát triển với nền công nghiệp phần mềm đầy cạnh tranh.

Hình 1.2: Quy trình phát triển phần mềm

1.2.1. Khái niệm quy trình phát triển phần mềm

Quy trình phát triển phần mềm là toàn bộ quá trình xây dựng lên sản phẩm đáp ứng các thông số kỹ thuật và yêu cầu của người dùng. SDLC cung cấp một tiêu chuẩn quốc tế mà các công ty sản xuất phần mềm có thể sử dụng để xây dựng và cải tiến sản phẩm công nghệ. Một quy trình tốt sẽ luôn tạo ra những sản phẩm đạt tiêu chuẩn.

Quy trình được chia thành 6 bước và mỗi giai đoạn đều có sự tham gia của đội ngũ phát triển phần mềm. Quy trình giúp tương tác hóa các hoạt động và yếu tố với nhau một cách nhịp nhàng, đem lại hiệu quả trong quá trình sản xuất phần mềm.

1.2.2. Các giai đoạn phát triển phần mềm

Giai đoạn 1: Needs identification (Xác định nhu cầu)

Needs identification là giai đoạn nghiên cứu thị trường và brainstorming (phương pháp động não) của quy trình. Trước khi xây dựng phần mềm, công ty cần thực hiện nghiên cứu sâu rộng thị trường để xác định khả năng tồn tại của sản phẩm. Developers phải xác định được các chức năng và dịch vụ mà phần mềm nên cung cấp được cho người tiêu dùng để họ cảm thấy sản phẩm cần thiết và hữu ích.

Ngoài ra, developers cũng nên thảo luận cùng với các bộ phận khác trong công ty về: Điểm mạnh, điểm yếu và cơ hội của sản phẩm. Quá trình phát triển phần mềm chỉ bắt đầu nếu sản phẩm thỏa mãn được mọi thông số nhất thiết để thành công.

Giai đoạn 2: Requirements Analytics (Phân tích yêu cầu)

Requirements Analytics là giai đoạn thực hiện khảo sát chi tiết yêu cầu, mong muốn của khách hàng. Sau đó, thông tin sẽ được tổng hợp vào tài liệu đặc tả yêu cầu. Tài liệu đặc tả phải đầy đủ các yêu cầu về chức năng, phi chức năng và giao diện. Ngoài ra, tài liệu còn cung cấp một bản phác thảo chi tiết về thành phần, phạm vi, nhiệm vụ của developers và các thông số thử nghiệm để tạo ra sản phẩm chất lượng.

Phân tích yêu cầu cũng là giai đoạn mà các developers lựa chọn cách tiếp cận phát triển phần mềm như: Mô hình chữ V (V Model) hay mô hình thác nước (Waterfall).

Giai đoạn 3: Design (Thiết kế)

Sau khi đã xác định và phân tích kỹ lưỡng về yêu cầu, sẽ chuyển sang giai đoạn nắm vai trò quan trọng thiết yếu của Quy trình phát triển phần mềm - thiết kế. Tại đây, các kiến trúc sư và nhà phát triển phần mềm sẽ đưa ra các thông số kỹ thuật tiên tiến mà họ cần để tạo ra sản phẩm theo yêu cầu. Vấn đề cần được thảo luận thêm giữa các bên bao gồm: Mức độ rủi ro, thành phần nhóm, công nghệ áp dụng, thời gian, ngân sách, giới hạn của dự án, phương pháp và thiết kế kiến trúc.

Tài liệu đặc điểm kỹ thuật thiết kế sẽ là kết quả cuối cùng của giai đoạn. DSD chỉ định thiết kế kiến trúc, thành phần, giao tiếp và luồng người dùng của sản phẩm.

Giai đoạn 4: Development (Lập trình)

Developers sẽ lập trình và triển khai thông số thiết kế. Lập trình viên sẽ coding dựa trên các thông số kỹ thuật và yêu cầu của sản phẩm đã được thống nhất trong các giai đoạn trước.

Sau khi coding hoàn tất, developers sẽ deploy sản phẩm trong môi trường phát triển (development environment). Lập trình viên sẽ thử nghiệm phiên bản đã tạo ra và điều chỉnh lại cho phù hợp với yêu cầu.

Giai đoạn 5: Testing (Kiểm thử)

Sau khi developers đã hoàn thành giai đoạn lập trình, tester sẽ tiếp nhận sản phẩm và tiến hành testing. Tester sẽ tạo test case (Kịch bản kiểm thử) dựa trên tài liệu giải pháp tạo ở giai đoạn 2 và tiến hành kiểm tra. Tester sẽ cập nhật kết quả test vào tool quản lý và thông báo bug (lỗi) đến developers. Tester và developers sẽ cùng nhau phối hợp xử lý các bug và cập nhật trên hệ thống quản lý lỗi. Trong thực tế, tùy theo mô hình phát triển phần mềm mà hoạt động, lập trình và kiểm Thử có thể diễn ra song song hoặc tiến hành lần lượt. Ví dụ như ở mô hình Waterfall, lập trình được thực hiện xong mới đến giai đoạn kiểm thử.

Giai đoạn 6: Deployment & Maintenance (Triển khai & bảo trì)

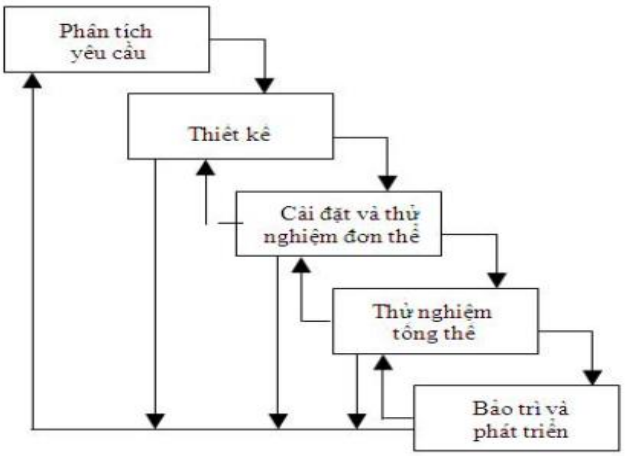
Tại giai đoạn này khi lỗi đã được xử lý xong, nhà phát triển phần mềm sẽ cung cấp sản phẩm hoàn chỉnh đến tay khách hàng. Testing vẫn được diễn ra ở giai đoạn triển khai để đảm bảo sản phẩm luôn có mức độ hoàn hảo cao. Sau khi phát hành, công ty sẽ tạo ra một nhóm bảo trì để quản lý các vấn đề mà khách hàng gặp phải khi sử

dụng sản phẩm. Bảo trì giúp khắc phục nhanh các vấn đề nhỏ xảy ra trong quá trình sử dụng sản phẩm.

1.2.3. Các mô hình phát triển phần mềm

1.2.3.1. Mô hình thác nước (Waterfall Model)

Mô hình thác nước hay còn gọi là mô hình tuyến tính hay mô hình kinh điển (classic model). Trong mô hình này, quy trình phát triển trông giống như một dòng chảy, với các giai đoạn được thực hiện theo trật tự nghiêm ngặt và không có sự quay lui hay nhảy vượt giai đoạn. Cụ thể, quá trình sẽ được thực hiện tuần tự qua các giai đoạn: phân tích yêu cầu, thiết kế, triển khai thực hiện, kiểm thử, phát triển và bảo trì. Tức là mô hình này sẽ xem quá trình xây dựng một sản phẩm phần mềm bao gồm nhiều giai đoạn tách biệt, sau khi hoàn tất một giai đoạn thì chuyển đến giai đoạn sau. Sau khi phân tích yêu cầu thì đặc tả yêu cầu, đưa cho khách hàng xem xét. Nếu khách hàng chấp nhận thì sẽ tiến hành tuần tự một loạt các giai đoạn như trên. Chỉ đưa sản phẩm cho khách hàng khi đã hoàn thiện.



Hình 1.3: Mô hình thác nước là mô hình phát triển phần mềm đầu tiên

Ưu điểm của mô hình Waterfall :

1. Dễ quản lý: Mô hình Waterfall có cấu trúc rõ ràng và dễ quản lý. Mỗi giai đoạn của quy trình phát triển được thực hiện một cách tuần tự, điều này giúp dễ dàng theo dõi tiến độ và quản lý dự án.

2. Tài liệu đầy đủ: Vì mỗi giai đoạn được hoàn thành trước khi chuyển sang giai đoạn tiếp theo, có nhiều tài liệu được tạo ra trong mỗi giai đoạn. Điều này giúp việc hiểu và bảo trì dự án ở các giai đoạn sau.

3. Kiểm soát tốt: Do mỗi giai đoạn được kiểm tra kỹ lưỡng trước khi tiến xa hơn, việc xác định và sửa lỗi ở giai đoạn trước đó trở nên dễ dàng hơn.

4. Dễ dàng đo lường tiến độ: Bởi vì mối giai đoạn được đặt ra trước với các mục tiêu rõ ràng, việc đo lường tiến độ so với các mục tiêu này trở nên đơn giản.

Nhược điểm của mô hình Waterfall:

1. Khó thích ứng với thay đổi: Mô hình Waterfall không linh hoạt. Nếu có yêu cầu mới hoặc thay đổi, việc thực hiện chúng sau khi đã bắt đầu giai đoạn triển khai có thể rất khó khăn.

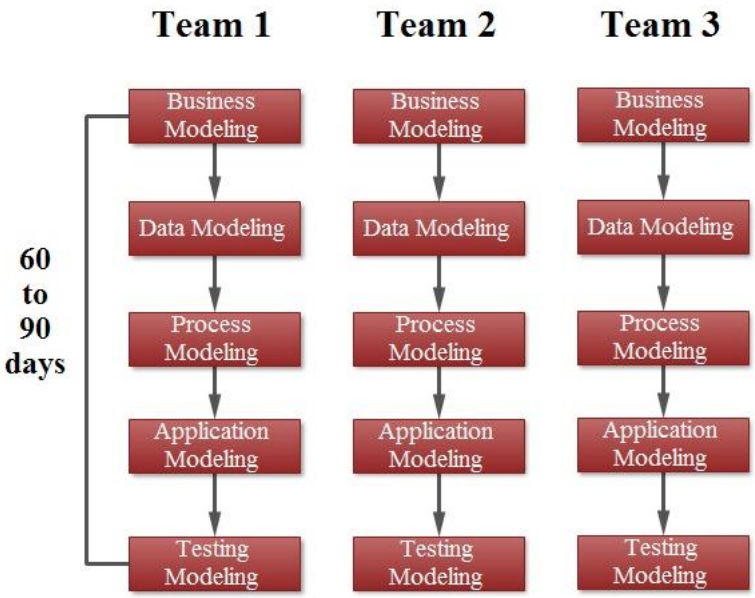
2. Không phản hồi linh hoạt: Do tính tuần tự của Waterfall, không có phản hồi ngay lập tức từ khách hàng hoặc người dùng cho đến khi sản phẩm hoàn thành. Điều này có thể dẫn đến việc phát hiện lỗi hoặc sự không hải lòng của khách hàng ở giai đoạn cuối cùng, khi việc sửa lỗi có thể trở nên đắt đỏ và khó khăn.

3. Khó quản lý dự án lớn: Trong các dự án lớn, việc chia nhỏ dự án thành các giai đoạn có thể trở nên phức tạp và khó duy trì quản lý, đặc biệt nếu có nhiều nhóm làm việc cùng một lúc.

4. Thời gian và chi phí: Nếu có lỗi phát sinh ở giai đoạn cuối, việc sửa chúng có thể tốn kém hơn nếu lỗi được phát hiện và sửa ngay từ giai đoạn đầu. Điều này có thể dẫn đến vượt quá ngân sách và thời gian dự định của dự án.

1.2.3.2. Mô hình RAD (Rapid Application Development)

Mô hình này đưa ra bởi IBM vào những năm 1980, qua sách của James Martin. Là mô hình phát triển gia tăng, tăng dần từng bước với mỗi chu trình phát triển rất ngắn(60-90 ngày). Xây dựng dựa trên hướng thành phần với khả năng tái sử dụng và sử dụng các ứng dụng tạo mã tự động. Gồm một số nhóm, mỗi nhóm làm một RAD theo các pha:



Hình 1.4: Mô hình RAD

Mô hình nghiệp vụ (Business Modeling): là cơ sở quan trọng để hiểu rõ nhu cầu và mục tiêu kinh doanh, từ đó hỗ trợ quá trình phát triển sản phẩm hoặc dịch vụ trong các bước tiếp theo của quá trình phát triển.

Mô hình dữ liệu (Data Modeling): Các đối tượng dữ liệu cần để hỗ trợ nghiệp vụ. Định nghĩa các thuộc tính của từng đối tượng và xác lập quan hệ giữa các đối tượng

Mô hình tiến trình (Process Modeling): Các đối tượng dữ liệu được chuyển sang luồng thông tin được thực hiện chức năng nghiệp vụ. Tạo mô tả xử lý để cập nhật(thêm, sửa, xóa, khôi phục) từng đối tượng dữ liệu.

Tự sinh ứng dụng (Application Modeling): Đây là công đoạn coding giống như các quy trình khác, khi dữ liệu và tiến trình thực hiện đã được vạch ra thì công đoạn này là sử dụng các công cụ lập trình, phần mềm biên dịch và các ngôn ngữ lập trình để tiến hành code lên các chức năng phần mềm.

Kiểm thử và chuyển giao (Testing Modeling): Là công đoạn test sản phẩm được tạo, sau khi việc coding đã hoàn thành, và cuối cùng đưa vào thử nghiệm trong thực tế, giao bán sản phẩm, tính toán doanh thu.

Ưu điểm của mô hình RAD :

1.Giảm thời gian phát triển phần mềm

2.Giảm rủi ro và dễ hiểu nhờ có sự tham gia của khách hàng

3.Tăng khả năng sử dụng lại các thành phần và giảm thiểu lỗi trong quá trình phát triển hệ thống.

4.Được chứng minh là tốt nhất cho các dự án quy mô vừa và nhỏ.

Nhược điểm của mô hình RAD :

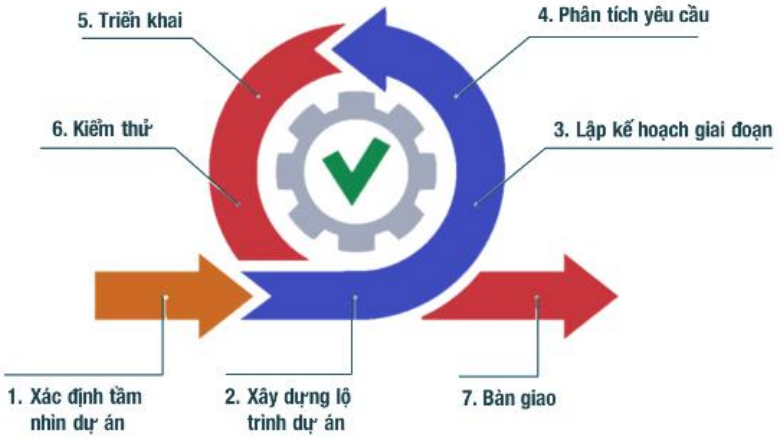
1.Yêu cầu các nhà phát triển phải có kỹ năng cao

2.Không thích hợp cho các dự án kỹ thuật phần mềm lớn.

3.Chỉ được sử dụng khi mô hình thiết kế là sẵn có và bên cạnh đó là ngân sách phải có nhiều để có thể xây dựng được nhiều team cùng phát triển song song.

1.2.3.3. Mô hình Agile

Khái niệm Agile (Agile Software Development) là một triết lý hoặc một khung tư duy để nhanh chóng thích ứng và phản hồi với sự thay đổi từ đó đạt được thành công trong một môi trường liên tục biến động, không chắc chắn. Phương pháp này là sự phát triển phần mềm linh hoạt thay vì phải dùng phương pháp truyền thống Waterfall. Được ứng dụng trong quy trình phát triển phần mềm với mục tiêu là đưa sản phẩm đến tay người dùng càng nhanh càng tốt, với chất lượng ngày càng được đảm bảo.



Hình 1.5: Quy trình Agile

Trên thực tế, phương pháp Agile được xây dựng trên nguyên tắc phân đoạn vòng lặp (iterative) và tăng trưởng (incremental). Đây là cách để quản lý một dự án bằng

cách chia nó thành nhiều giai đoạn. Nó liên quan đến sự hợp tác liên tục với các bên liên quan và cải tiến liên tục ở mọi giai đoạn. Khi công việc bắt đầu, các nhóm luân chuyển qua một quá trình lập kế hoạch, thực hiện và đánh giá. Sự hợp tác liên tục là rất quan trọng, với cả các thành viên trong nhóm và các bên liên quan của dự án.

12 nguyên tắc Agile và 4 giá trị cốt lõi của tuyên ngôn Agile

Hình 1.6: Bốn giá trị cốt lõi của Agile

Cá nhân và sự tương tác quan trọng hơn quy trình và công cụ

Đánh giá cao con người hơn các quy trình hoặc công cụ là điều dễ hiểu bởi vì chính những người đáp ứng nhu cầu kinh doanh và thúc đẩy quá trình phát triển. Nếu quy trình hoặc công cụ thúc đẩy sự phát triển, thì nhóm sẽ ít phản ứng hơn với sự thay đổi và ít có khả năng đáp ứng nhu cầu của khách hàng hơn. Giao tiếp là một ví dụ về sự khác biệt giữa đánh giá cá nhân và quá trình. Trong trường hợp cá nhân, giao tiếp là linh hoạt và xảy ra khi có nhu cầu. Trong trường hợp của quá trình, giao tiếp được lên lịch và yêu cầu nội dung cụ thể.(Sẽ không thấy manager, leader) mà các cá nhân sẽ tự tương tác với nhau chủ động trao đổi và làm việc với nhau.

Tập trung phát triển sản phẩm hơn là viết tài liệu

Trong lịch sử, rất nhiều thời gian được dành cho việc ghi chép sản phẩm để phát triển và phân phối cuối cùng. Đặc điểm kỹ thuật, yêu cầu kỹ thuật, bản cáo bạch kỹ

thuật, tài liệu thiết kế giao diện, kế hoạch thử nghiệm, kế hoạch tài liệu và phê duyệt cần thiết cho từng loại. Danh sách này rất rộng và là nguyên nhân dẫn đến sự chậm trễ kéo dài trong quá trình phát triển. Agile không loại bỏ tài liệu, nhưng nó sắp xếp hợp lý nó theo một hình thức cung cấp cho nhà phát triển những gì cần thiết để thực hiện công việc mà không bị sa lầy vào những chi tiết vụn vặt. Các yêu cầu về tài liệu Agile dưới dạng câu chuyện của người dùng, đủ để một nhà phát triển phần mềm bắt đầu nhiệm vụ xây dựng một chức năng mới. Tuyên ngôn Agile coi trọng tài liệu, nhưng nó coi trọng phần mềm hoạt động hơn.

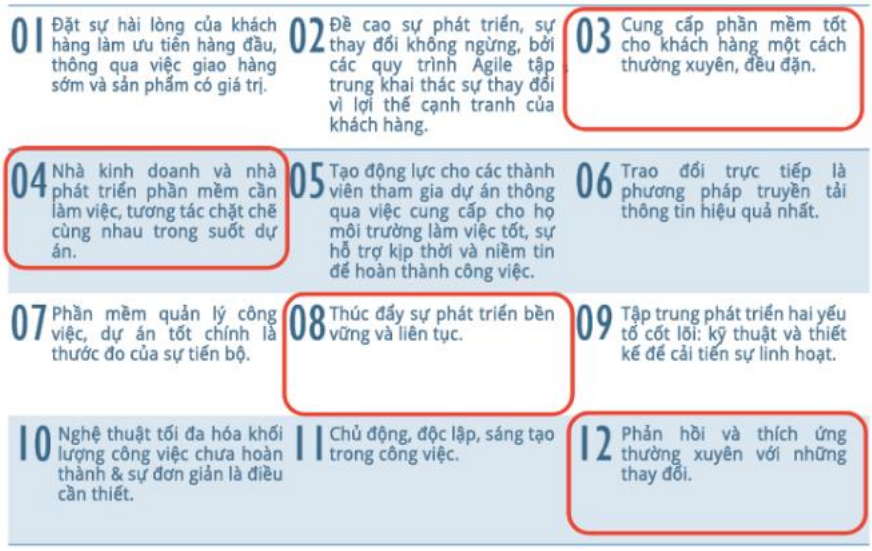
Hợp tác với khách hàng trong quá trình đàm phán hợp đồng

Đàm phán là khoảng thời gian khách hàng và người quản lý sản phẩm tìm ra các chi tiết của việc giao hàng, với các điểm mà các chi tiết có thể được thương lượng lại. Với các mô hình phát triển như Waterfall, khách hàng thương lượng các yêu cầu đối với sản phẩm, thường rất chi tiết, trước khi bắt đầu bất kỳ công việc nào. Điều này có nghĩa là khách hàng đã tham gia vào quá trình phát triển trước khi quá trình phát triển bắt đầu và sau khi hoàn thành, nhưng không phải trong quá trình này. Tuyên ngôn Agile mô tả một khách hàng đã tham gia và cộng tác trong suốt quá trình phát triển. Điều này giúp cho việc phát triển đáp ứng nhu cầu của khách hàng trở nên dễ dàng hơn rất nhiều. Các phương pháp Agile có thể bao gồm khách hàng trong các khoảng thời gian cho các bản trình diễn định kỳ.

Ứng phó với sự thay đổi hơn là theo kế hoạch

Phát triển phần mềm truyền thống coi thay đổi là một khoản chi phí, vì vậy nó phải được tránh. Mục đích là phát triển các kế hoạch chi tiết, phức tạp, với một tập hợp các tính năng được xác định và với mọi thứ, nói chung, có mức độ ưu tiên cao như mọi thứ khác và với một số lượng lớn nhiều phụ thuộc vào việc phân phối theo một thứ tự nhất định để nhóm có thể làm việc trên mảnh tiếp theo của câu đố.Với Agile, độ ngắn của một lần lặp có nghĩa là các ưu tiên có thể được chuyển từ lần lặp này sang lần lặp khác và các tính năng mới có thể được thêm vào lần lặp tiếp theo. Quan điểm của Agile là những thay đổi luôn cải thiện một dự án; thay đổi cung cấp giá trị bổ sung. Có lẽ không có gì minh họa cách tiếp cận tích cực của Agile để thay đổi tốt hơn khái niệm điều chỉnh phương pháp, được định nghĩa trong phương pháp phát triển hệ thống

thông tin Agile đang được sử dụng như: "Một quy trình hoặc khả năng trong đó các tác nhân của con người xác định cách tiếp cận phát triển hệ thống cho một tình huống dự án cụ thể thông qua các thay đổi đáp ứng và xen kẽ động giữa các bối cảnh, ý định và các đoạn phương pháp". Các phương pháp Agile cho phép nhóm Agile sửa đổi quy trình và làm cho nó phù hợp với nhóm hơn là ngược lại.



Hình 1.7: 12 nguyên tắc quản lý dự án linh hoạt của phương pháp Agile

Khách hàng có nhu cầu chuyển giao sản phẩm liên tục, nhanh chóng nên đây được xem là ưu tiên hàng đầu khi áp dụng mô hình Agile. Đặt sự hài lòng của khách hàng làm ưu tiên hàng đầu, thông qua việc giao hàng sớm và sản phẩm có giá trị.

Chấp nhận sự thay đổi mặc dù đang ở giai đoạn cuối của việc thực hiện dự án. Đề cao sự phát triển, sự thay đổi không ngừng, bởi các quy trình Agile tập trung khai thác sự thay đổi vì lợi thế cạnh tranh của khách hàng.

Ưu tiên những khung thời gian ngắn (từ vài tuần đến vài tháng) trong việc chuyển giao công việc. Cung cấp phần mềm tốt cho khách hàng một cách thường xuyên, đều đặn.

Sự tương tác giữa chuyên gia kinh doanh và nhóm phát triển dự án nên được diễn ra hằng ngày. Nhà kinh doanh và nhà phát triển phần mềm cần làm việc, tương tác chặt chẽ cùng nhau trong suốt dự án.

Những cá nhân có động lực được làm việc trong môi trường có sự hỗ trợ tốt, giúp mang lại hiệu quả công việc cao hơn. Tạo động lực cho các thành viên tham gia dự án

thông qua việc cung cấp cho họ môi trường làm việc tốt, sự hỗ trợ kịp thời và niềm tin để hoàn thành công việc.

Tương tác trực diện (face to face) là phương pháp hiệu quả nhất để truyền thông thông tin trong nhóm dự án. Trao đổi trực tiếp là phương pháp truyền tải thông tin hiệu quả nhất.

Chắc chắn công việc phân công được thực hiện để đảm bảo tiến độ. Phần mềm quản lý công việc, dự án tốt chính là thước đo của sự tiến bộ.

Mô hình Agile giúp thúc đẩy sự phát triển bền vững, có tính ổn định liên tục vô thời hạn. Thúc đẩy sự phát triển bền vững và liên tục.

Tính nhanh nhẹn được quyết định bởi quá trình theo dõi những kỹ thuật xuất sắc và các thiết kế tốt. Tập trung phát triển hai yếu tố cốt lõi: kỹ thuật và thiết kế để cải tiến sự linh hoạt.

Tính đơn giản là yếu tố quan trọng và cần thiết trong việc tối đa hóa lượng công hiện có. Nghệ thuật tối đa hóa khối lượng công việc chưa hoàn thành và sự đơn giản là điều cần thiết.

Kiến trúc, yêu cầu và thiết kế tốt nhất thường được tạo ra từ nhóm tự tổ chức (self-organizing teams). Chủ động, độc lập, sáng tạo trong công việc.

Định kỳ sau một khoảng thời gian, nhóm sẽ đánh giá để đưa ra những cách làm hiệu quả và điều chỉnh những hành vi chưa phù hợp. Phản hồi và thích ứng thường xuyên với những thay đổi.

Ưu điểm của mô hình Agile :

1.Dễ dàng thay đổi: Dự án được chia thành những phần nhỏ riêng biệt nên có thể dễ dàng thay đổi mà không ảnh hưởng đến tổng quan. Việc thay đổi có thể thực hiện ở bất kỳ giai đoạn nào của dự án.

2.Không cần nắm thông tin ban đầu: Mỗi khâu có thể được thực hiện mà không cần nắm quá nhiều thông tin từ những phần khác của dự án. Do đó, Agile sẽ phù hợp với nhiều loại dự án khác nhau, đặc biệt dự án không xác định mục tiêu cuối cùng.

3.Bàn giao nhanh chóng: Việc chia nhỏ dự án cho phép việc kiểm tra dễ dàng hơn. Qua đó, quá trình xác định, chỉnh sửa và bàn giao trở nên thuận tiện hơn.

4.Chú ý đến phản hồi của khách hàng: Tạo cơ hội để khách hàng và người dùng cuối có thể đóng góp ý kiến, phản hồi giúp điều chỉnh sản phẩm cuối cùng.

5.Cải tiến liên tục: Những ý kiến của nhân sự và khách hàng được ghi nhận để cải thiện chất lượng dự án.

Nhược điểm của mô hình Agile :

1.Vì dự án được chia nhỏ nên khó khăn trong việc lên kế hoạch chi tiết dự án.

2.Phương pháp Agile khá phức tạp nên bạn cần được đào tạo, hướng dẫn cụ thể để có thể thực hiện tốt.

3.Mô hình này có tính biến đổi cao nên không có quá nhiều tài liệu hướng dẫn phù hợp cho thời điểm hiện tại.

4. Bắt buộc phải có sự tương hỗ giữa các phòng ban, bên liên quan để đảm bảo thời gian hoàn thành và hiệu quả của dự án.

5.Chi phí để thực hiện theo phương pháp này thường cao hơn so với các mô hình khác.

1.3. Mẫu thiết kế phần mềm (Design Pattern)

1.3.1. Tổng quan về Design Pattern

Design Pattern, hay còn gọi là mẫu thiết kế sẵn có, là một giải pháp tổng hợp cho các vấn đề thường gặp trong thiết kế phần mềm. Nói cách khác, Design Pattern là một khuôn mẫu được thiết kế sẵn giúp giải quyết các vấn đề thiết kế phần mềm một cách hiệu quả và linh hoạt.

Design Pattern không phải là một mã nguồn hoặc một thành phần phần mềm cụ thể, mà là một cách viết code hoặc một mô hình thiết kế có thể được áp dụng trong nhiều tình huống khác nhau. Quá trình phát triển phần mềm sẽ luôn gắn liền cùng với sự thay đổi về các yêu cầu cụ thể. Chính vào lúc này thì hệ thống sẽ có thể phình to, một số tính năng mới khác đều sẽ được thêm vào trong nhiều hơn, trong khi đó hiệu suất(performance) cần được tối ưu hơn cả.

Design Pattern có khả năng cung cấp cho người sử dụng một số giải pháp đã được tối ưu hóa cũng như kiểm chứng để có thể giải quyết được các vấn đề xảy ra trong công nghệ phần mềm. Những giải pháp tổng quát này sẽ giúp bạn gia tăng được cho tốc độ phát triển của phần mềm nhờ cách đưa ra được các mô hình test cũng như các mô hình phát triển đều đã qua kiểm nghiệm.

Nếu như gặp phải các khó khăn không đáng có với chính những vấn đề đã giải quyết, thì design patterns là một trong những hướng đi vô cùng đúng đắn. Bởi vì nó có thể giúp giải quyết triệt để được các vấn đề cũng như đưa ra được giải pháp đỡ tốn kém thời gian. Design Pattern có thể hỗ trợ cho Developer hiểu hơn về code của người khác nhanh chóng. Nhờ vậy, mọi hoạt động của các thành viên trong team đều có thể chia sẻ với nhau dễ dàng hơn. Từ đó, dự án sẽ được xây dựng cùng nhau nhanh chóng hơn mà không tốn nhiều thời gian.

1.3.2. MVC Design Pattern

1.3.2.1. Khái niệm và các thành phần của MVC

MVC (MVC Design Pattern) là viết tắt của Model - View - Controller. Đó là một mẫu thiết kế, mô hình lập trình phổ biến được sử dụng để tạo cấu trúc cho nhiều trang web, ứng dụng tiên tiến.

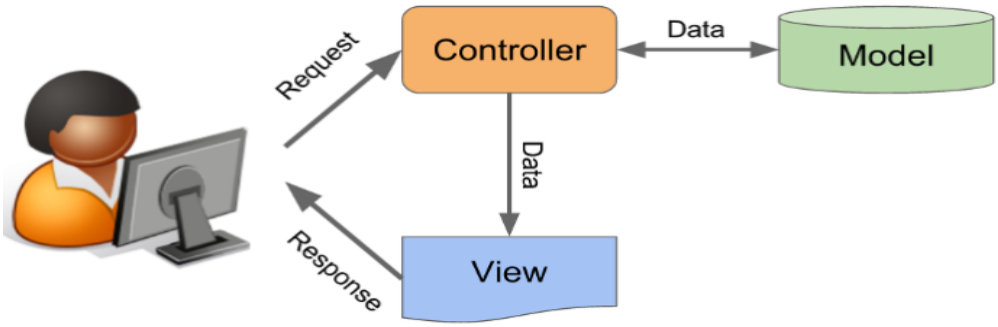
Mô hình MVC chia các components của ứng dụng thành 3 loại khác nhau. Các components của mô hình MVC đảm nhiệm một trách nhiệm nhất định và mỗi components đều độc lập với components khác. Việc thay đổi một components sẽ không ảnh hưởng hoặc ảnh hưởng rất ít đến components khác.

Model: Model đảm nhận nhiệm vụ cung cấp dữ liệu từ cơ sở dữ liệu và lưu trữ các thông tin đó ở nơi chứa dữ liệu. Tất cả các Business Logic(Logic nghiệp vụ) đều được thực thi ở Model. Dữ liệu được nhập vào bởi người sử dụng qua View sẽ được kiểm tra ở Model trước khi được lưu vào cơ sở dữ liệu. Truy xuất dữ liệu, xử lý dữ liệu là các thành phần của Model.

View: Là thành phần hỗ trợ việc hiển thị dữ liệu hay kết quả ra màn hình, hỗ trợ nhập thông tin từ phía người dùng.

Controller: Là thành phần hỗ trợ kết nối người dùng với máy chủ, đón nhận yêu cầu người dùng, thực hiện chuyển xử lý, lựa chọn và cập nhật Model và View tương ứng để hiển thị thông tin về phía người dùng. Hỗ trợ kết nối giữa Model và View, giúp Model xác định được View cần hiển thị.

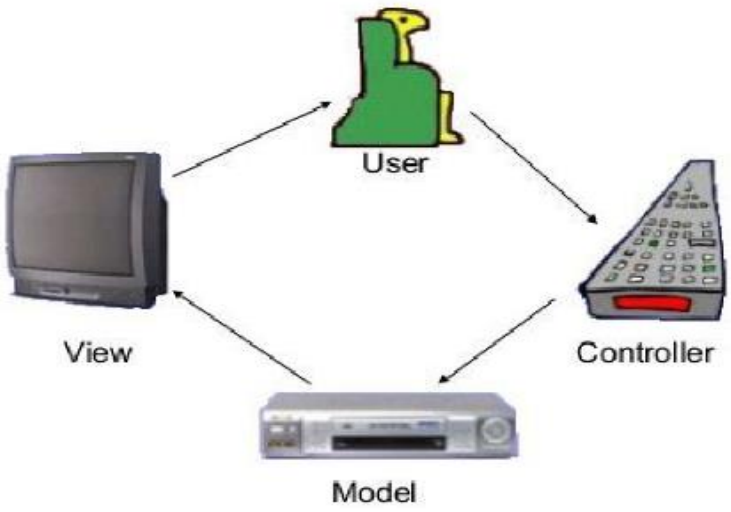
1.3.2.2. Luồng xử lý trong MVC



Hình 1.8: Luồng xử lý của MVC

Khi một yêu cầu từ máy khách(Client) gửi đến máy chủ(Server) thì sẽ bị Controller chặn lại để xem đó là URL request hay là một sự kiện. Sau đó, Controller sẽ xử lý yêu cầu của máy khách rồi giao tiếp với Model. Model sẽ chuẩn bị dữ liệu rồi gửi lại cho Controller. Cuối cùng xử lý xong yêu cầu thì Controller gửi dữ liệu trở lại View và hiển thị cho phía người dùng.

Ví dụ một mô hình thực tế :



Hình 1.9: Mô hình MVC khi A xem phim

A sẽ xem phim như sau :

Khi A muốn xem phim yêu thích A sẽ sử dụng điều khiển(đây chính là Controller) để chọn đến phim đó đến đầu đĩa.

Đầu đĩa là nơi xử lý dữ liệu, chọn lựa cách xử lý, nội dung cần thiết, nghĩa là đầu đĩa đóng vai trò là Model.

Sau khi đã lấy được nội dung phù hợp, đĩa sẽ được đọc từ máy và trình chiếu lên màn hình. Khi đó A sẽ thấy phim mà muốn xem và màn hình đóng vai trò là View.

Và cách thức này sẽ được lặp đi lặp lại khi A chọn một phim khác.

1.3.2.3. Ưu và nhược điểm của MVC

Ưu điểm của MVC :

Tính linh hoạt và uyển chuyển cao: Cho phép người lập trình viên có thể tách biệt công việc trong qua trình xây dựng chức năng cho ứng dụng và quá trình xây dựng giao diện cho người dùng, thể hiện tính chuyên nghiệp trong lập trình, phân tích thiết kế giúp phát triển ứng dụng nhanh hơn.

Cho phép việc thay đổi thành phần của dữ liệu(Model) sẽ không ảnh hưởng nhiều đến giao diện của người dùng. Vì mô hình đưa ra Model để không cho người dùng thao tác trực tiếp vào dữ liệu vật lý(cơ sở dữ liệu hay tập tin) mà phải thông qua Model, do vậy cho dù dữ liệu vật lý có bị thay đổi thì cấu trúc Model cho việc truy cập, xử lý, lưu trữ dữ liệu sẽ không bị ảnh hưởng.

Tính tin cậy cao: việc chia từng phần riêng biệt giúp chúng ra sửa đổi từng phần riêng biệt, không ảnh hưởng, có thể thay thế thành từng phần tương đương, có thể chia công việc theo nhóm, biên dịch độc lập, tăng cường khả năng tích hợp với khả năng đứng đắn cao.

Tính tái sử dụng: có thể sử dụng các thành phần chia cắt lại trong các ứng dụng khác hay dùng lại nhiều lần trong cùng một ứng dụng, tăng tính hiệu quả trong lập trình.

Khả năng triển khai bảo trì nhanh chóng: vì các thành phần độc lập với nhau.

Giúp dễ dàng trong quá trình làm việc với ứng dụng, bảo trì, nâng cấp ứng dụng.

Nhược điểm của MVC :

Phân chia công việc và nghiệp vụ giữa các thành phần không đồng đều, trong đó Model phải xử lý rất nhiều tác vụ.

Sự hỗ trợ cho quá trình kiểm thử không quá tốt bởi lớp View phải phụ thuộc vào cả Controller và Model. View sẽ không thể xử lý được vấn đề gì bởi View không thể nhận yêu cầu và cũng không có dữ liệu để hiển thị. Để tiến hành kiểm thử trên View, cần giả lập cả Controller và Model.

Đối với các mô hình, ứng dụng nhỏ thì việc triển khai sử dụng MVC có vẻ quá cồng kềnh.

Đối với các ứng dụng quy mô lớn, quy trình xử lý nghiệp vụ có tính phức tạp cao, lượng dữ liệu lớn thì mô hình MVC trở nên không còn khả dụng.

Tóm lại, hiện tại mô hình MVC đang được ứng dụng rất nhiều trong các mô hình lập trình ứng dụng web. Để việc lập trình web trở nên đơn giản, chuyên nghiệp hơn và có thể ứng dụng cho nhóm làm việc nhiều người thì việc áp dụng mô hình MVC là rất tốt và khả quan.

1.4. Kết luận chương 1

Chương 1 giới thiệu tổng quan về phát triển phần mềm bao gồm lịch sử phát triển từ khi nào đến ngày nay và quy trình phát triển. Sau đó, tập trung vào các giai đoạn trong quy trình phát triển phần mềm và phân tích các quy trình phổ biến đang được sử dụng ngày nay. Khi đã chọn quy trình phát triển phần mềm phù hợp, bước tiếp theo là xây dựng phần mềm để đạt hiệu quả cao. Phần tiếp theo là về mẫu thiết kế phần mềm (Design Pattern). Nói về lợi ích của việc sử dụng mẫu thiết kế sẵn có và giới thiệu một Design Pattern phổ biến và nổi tiếng là MVC (Model-View-Controller). Đây là một mô hình thiết kế giúp tổ chức cách các thành phần trong phần mềm tương tác với nhau. Chương tiếp theo sẽ đi sâu hơn vào cách các thành phần trong MVC hoạt động cùng nhau, từ việc truyền dữ liệu, xử lý, đến hiển thị dữ liệu.

CHƯƠNG 2 - Kiến trúc REST và RESTful API

2.1. Nguồn gốc của REST

Lý do thiết kế kiến trúc Web có thể được mô tả bằng một kiến trúc phong cách bao gồm các ràng buộc áp dụng cho các yếu tố trong kiến trúc. Bằng cách xem xét tác động của mỗi ràng buộc khi nó được thêm vào phong cách tiến hóa, có thể xác định các thuộc tính mà các ràng buộc của Web tạo ra. Các ràng buộc bổ sung sau đó có thể được áp dụng để tạo thành một phong cách kiến trúc mới phản ánh tốt hơn các thuộc tính mong muốn của một kiến trúc Web hiện đại. Phần này cung cấp một tổng quan chung về REST thông qua việc đi qua quá trình suy ra nó như một kiến trúc phong cách. Các phần sau sẽ mô tả chi tiết hơn về các ràng buộc cụ thể tạo nên phong cách REST.

2.1.1. Null Style

Đối với quá trình thiết kế kiến trúc, có hai quan điểm phổ biến. Quan điểm đầu tiên cho rằng một người thiết kế bắt đầu từ không có gì(một bảng trắng) hoặc bảng vẽ và xây dựng một kiến trúc từ các thành phần quen thuộc cho đến khi nó đáp ứng được các yêu cầu của hệ thống dự định. Quan điểm thứ hai cho rằng một người thiết kế bắt đầu với nhu cầu của hệ thống như một tổng thể, không có ràng buộc cụ thể, sau đó từ từ xác định và áp dụng các ràng buộc vào các thành phần của hệ thống để phân biệt không gian thiết kế và cho phép các yếu tố ảnh hưởng đến hành vi của hệ thống chảy mạch tự nhiên, hòa hợp với hệ thống. Trong đó, quan điểm đầu tiên nhấn mạnh sự sáng tạo và tầm nhìn không giới hạn, trong khi quan điểm thứ hai nhấn mạnh sự kiềm chế và hiểu biết về bối cảnh của hệ thống. REST đã được phát triển bằng cách sử dụng quy trình sau. Các hình 2.1 đến 2.8 minh họa điều này một cách đồ họa theo cách các ràng buộc được áp dụng sẽ phân biệt quan điểm quá trình của một kiến trúc khi bộ ràng buộc tăng dần được áp dụng.

Kiểu Null đơn giản chỉ là một tập hợp rỗng của các ràng buộc. Từ góc độ kiến trúc, kiểu null mô tả một hệ thống trong đó không có ranh giới đặc biệt giữa các thành phần. Đó là điểm khởi đầu cho việc mô tả về REST.

2.1.2. Client-Server Style

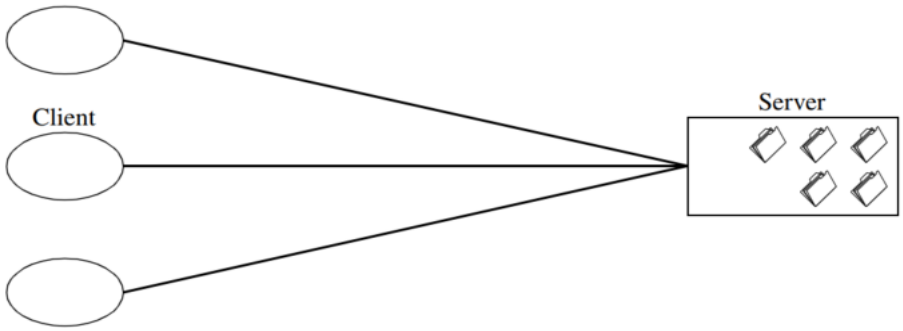
Hình 2.1: Client-Server

Phong cách client-server là một trong những kiểu kiến trúc phổ biến nhất cho các ứng dụng dựa trên mạng. Một thành phần máy chủ, cung cấp một tập hợp các dịch vụ, lắng nghe các yêu cầu về những dịch vụ đó. Một thành phần khách hàng, muốn thực hiện một dịch vụ nào đó, gửi một yêu cầu đến máy chủ thông qua một kết nối. Máy chủ có thể từ chối hoặc thực hiện yêu cầu và gửi một phản hồi lại cho khách hàng.

Nguyên tắc "tách biệt các quan tâm" là cơ sở của các ràng buộc client-server. Một sự tách biệt chính xác về chức năng sẽ làm đơn giản hóa thành phần máy chủ để cải thiện khả năng mở rộng. Điều này thường thể hiện dưới dạng việc di chuyển tất cả chức năng giao diện người dùng vào thành phần khách hàng. Sự tách rời cũng cho phép hai loại thành phần này tiến hóa độc lập, miễn là giao diện không thay đổi.

2.1.3. Client-Stateless-Server Style

Kiểu Client-Stateless-Server phát sinh từ Client-Server với ràng buộc bổ sung là không cho phép trạng thái phiên trên thành phần máy chủ. Mỗi yêu cầu từ khách hàng đến máy chủ phải chứa đủ thông tin cần thiết để hiểu yêu cầu và không thể tận dụng bất kỳ ngữ cảnh được lưu trữ trên máy chủ. Trạng thái phiên được duy trì hoàn toàn trên khách hàng.



Hình 2.2: Client-Stateless-Server

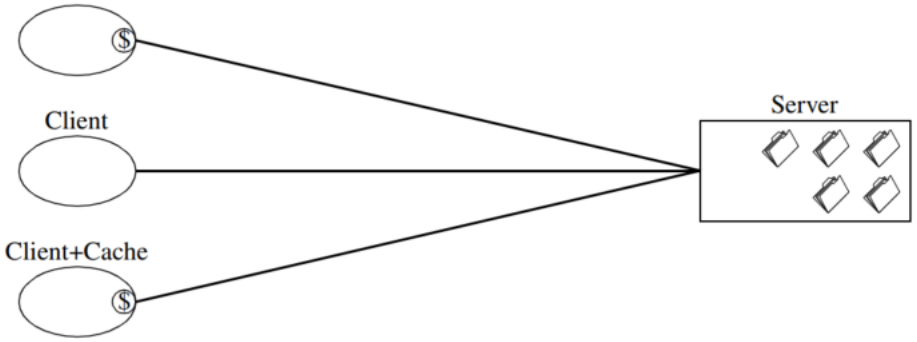
Ràng buộc này tạo ra các đặc tính về khả năng quan sát, đáng tin cậy và khả năng mở rộng. Khả năng quan sát được cải thiện vì một hệ thống giám sát không cần phải xem xét nhiều hơn một dữ liệu yêu cầu để xác định đầy đủ bản chất của yêu cầu đó. Đáng tin cậy được cải thiện vì nó làm giảm khó khăn khi phục hồi từ những lỗi một phần. Khả năng mở rộng được cải thiện vì không cần lưu trữ trạng thái giữa các yêu cầu, cho phép thành phần máy chủ giải phóng tài nguyên nhanh chóng và đơn giản hóa việc triển khai vì máy chủ không cần quản lý việc sử dụng tài nguyên qua các yêu cầu.

Tương tự như hầu hết các lựa chọn kiến trúc, ràng buộc không lưu trạng thái phản ánh một sự đánh đổi thiết kế. Hạn chế là có thể giảm hiệu suất mạng bằng cách tăng dữ liệu lặp đi lặp lại (chi phí trên mỗi tương tác) được gửi trong một loạt các yêu cầu, vì dữ liệu đó không thể được lưu trữ trên máy chủ trong ngữ cảnh chia sẻ. Ngoài ra, việc đặt trạng thái ứng dụng ở phía khách hàng giảm khả năng kiểm soát của máy chủ đối với hành vi ứng dụng nhất quán, vì ứng dụng trở nên phụ thuộc vào việc triển khai đúng đắn của ngữ nghĩa trên nhiều phiên bản khách hàng khác nhau.

2.1.4. Client-Cache-Stateless-Server

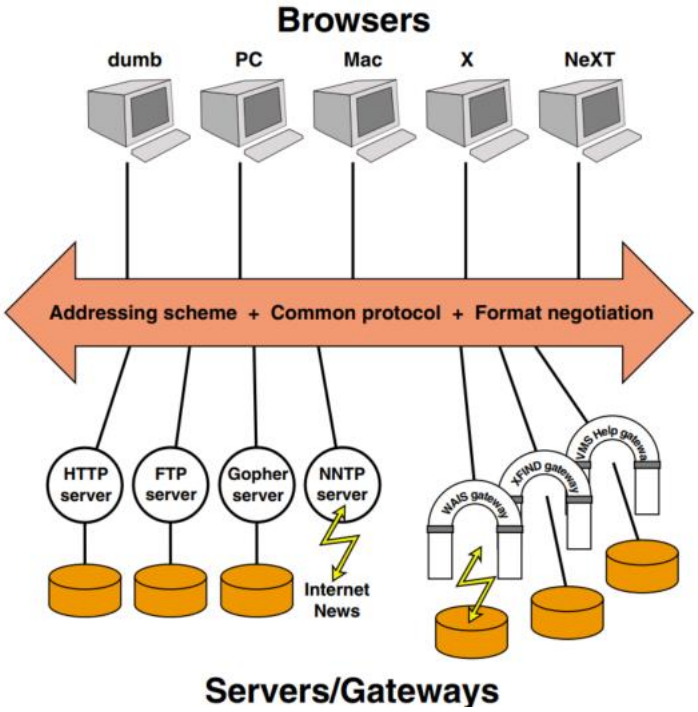
Bộ nhớ cache là một thành phần trong hệ thống, thường là một phần của máy khách (client) hoặc một phần mềm trung gian, được sử dụng để lưu trữ những dữ liệu đã được truy cập trước đó. Chức năng chính của bộ nhớ cache là giữ lại các dữ liệu từ các phản hồi trước đó từ máy chủ hoặc từ các nguồn dữ liệu khác, như vậy khi có yêu cầu tương tự, dữ liệu có thể được trả về từ bộ nhớ cache mà không cần truy cập lại máy chủ. Điều này giúp cải thiện hiệu suất mạng bằng cách giảm thiểu thời gian và tài nguyên cần thiết để truy cập dữ liệu từ nguồn gốc, đồng thời giảm tải cho máy chủ bằng cách giảm số lượng yêu cầu mới.

Để tăng cường hiệu suất mạng, áp dụng ràng buộc bộ nhớ cache để tạo ra kiểu kiến trúc client-cache-stateless-server. Ràng buộc cache đòi hỏi rằng dữ liệu trong phản hồi cho một yêu cầu phải được đánh dấu một cách rõ ràng hoặc ngầm định là có thể được lưu vào bộ nhớ cache hoặc không thể lưu vào. Nếu một phản hồi có thể lưu vào cache, thì bộ nhớ cache của client được cấp quyền tái sử dụng dữ liệu phản hồi đó cho các yêu cầu tương tự sau này.



Hình 2.3: Client-Cache-Stateless-Server

Thêm ràng buộc cache mang lại lợi thế tiềm ẩn là có khả năng giảm hoặc loại bỏ một số tương tác cụ thể, từ đó nâng cao hiệu suất, khả năng mở rộng, và hiệu suất được người dùng cảm nhận thông qua việc giảm độ trễ trung bình của một loạt các tương tác. Tuy nhiên, điều đổi lấy là cache có thể làm giảm đáng kể tính đáng tin cậy nếu dữ liệu trong cache trở nên cũ và khác biệt đáng kể so với dữ liệu mà sẽ được lấy nếu yêu cầu được gửi trực tiếp đến máy chủ.



Hình 2.4: Sơ đồ kiến trúc WWW ban đầu

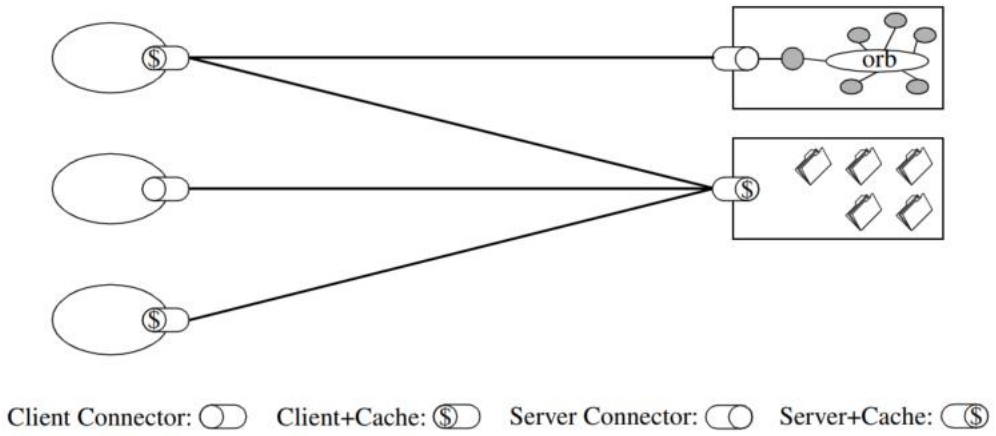
Kiến trúc Web ban đầu, được minh họa trong Hình 2.5, được định hình bởi bộ ràng buộc client-cache-stateless-server. Trước năm 1994, nguyên tắc thiết kế của kiến

trúc Web tập trung vào việc tương tác không lưu trạng thái giữa máy khách và máy chủ để trao đổi các tài liệu tĩnh trên Internet. Các giao thức giao tiếp hỗ trợ cache không chia sẻ cơ bản nhưng không ép buộc một bộ quy ước nhất quán cho tất cả các tài nguyên. Thay vào đó, Web dựa vào việc sử dụng một thư viện triển khai client-server chung để duy trì tính nhất quán trên các ứng dụng Web khác nhau.

Những nhà phát triển Web đã vượt xa ngoài thiết kế ban đầu. Ngoài tài liệu tĩnh, các yêu cầu có thể xác định các dịch vụ tạo ra phản hồi một cách động, như bản đồ hình ảnh được phát triển bởi Kevin Hughes và các kịch bản phía máy chủ bởi Rob McCool.

Công việc cũng đã bắt đầu trên các thành phần trung gian dưới dạng các proxy và cache chia sẻ, nhưng cần có các mở rộng cho các giao thức để chúng có thể giao tiếp một cách đáng tin cậy. Các phần tiếp theo mô tả các ràng buộc được thêm vào kiến trúc của Web để hướng dẫn cho các mở rộng tạo nên kiến trúc Web hiện đại.

2.1.5. Giao diện đồng nhất(Uniform Interface)



Hình 2.5: Uniform-Client-Cache-Stateless-Server

Kiến trúc REST có điểm trung tâm phân biệt nó so với các kiểu kiến trúc mạng khác là sự tập trung vào một giao diện đồng nhất giữa các thành phần (Hình 2.6). Bằng cách áp dụng nguyên tắc của công nghệ phần mềm vào giao diện thành phần, kiến trúc hệ thống tổng thể được đơn giản hóa và khả năng quan sát các tương tác được cải thiện. Các triển khai được tách rời khỏi các dịch vụ mà chúng cung cấp, điều này khuyến khích khả năng tiến hóa độc lập. Tuy nhiên, điều đổi lấy là giao diện đồng nhất

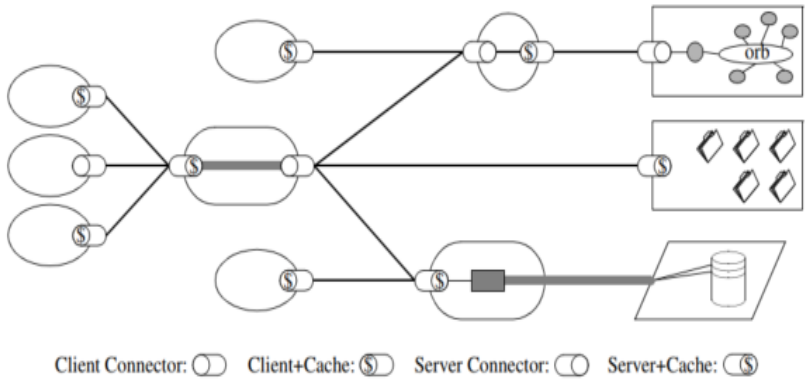
làm giảm hiệu suất vì thông tin được truyền theo một dạng chuẩn hóa thay vì dựa vào nhu cầu cụ thể của ứng dụng. Giao diện REST được thiết kế để hiệu quả trong việc truyền dữ liệu hypermedia cỡ lớn, tối ưu hóa cho trường hợp thông thường trên Web, nhưng dẫn đến một giao diện không tối ưu cho các hình thức tương tác kiến trúc khác.

Để có được một giao diện đồng nhất, cần có nhiều ràng buộc kiến trúc để hướng dẫn hành vi của các thành phần. REST được định nghĩa bởi bốn ràng buộc giao diện:

- Xác định tài nguyên (identification of resources).
- Thao tác với tài nguyên thông qua các biểu diễn (manipulation of resources through representations).
- Tin nhắn tự mô tả (self-descriptive messages).
- Hypermedia làm động cơ của trạng thái ứng dụng (hypermedia as the engine of application state).

Những ràng buộc này sẽ được thảo luận chi tiết trong phần 2.2.

2.1.6. Hệ thống phân lớp(Layered System)

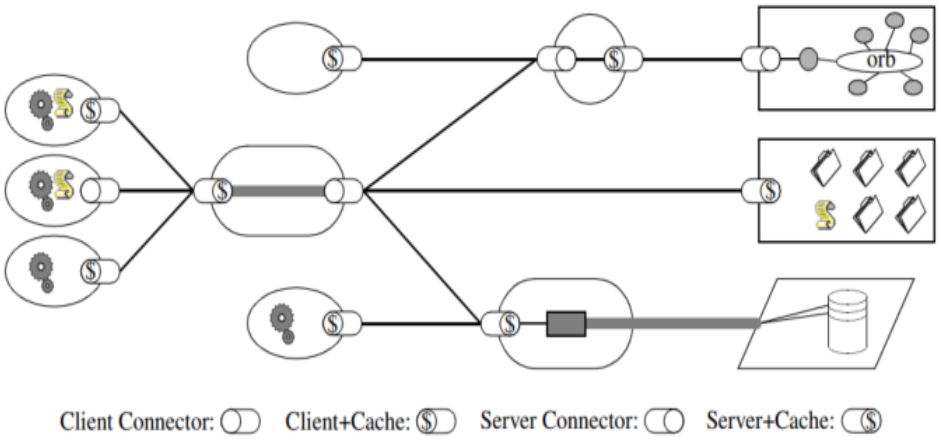


Hình 2.6: Uniform-Layered-Client-Cache-Stateless-Server

Để cải thiện hành vi cho yêu cầu quy mô Internet, thêm các ràng buộc hệ thống lớp (Hình 2.7). Kiểu kiến trúc lớp hệ thống cho phép một kiến trúc được tạo thành từ các lớp phân cấp bằng cách ràng buộc hành vi của thành phần sao cho mỗi thành phần không thể "nhìn thấy" xa hơn lớp trực tiếp mà họ đang tương tác. Bằng cách hạn chế kiến thức về hệ thống chỉ trong một lớp duy nhất, đặt một giới hạn về tổng thể phức

tạp của hệ thống và thúc đẩy tính độc lập với cơ sở. Các lớp có thể được sử dụng để đóng gói các dịch vụ cổ điển và bảo vệ các dịch vụ mới khỏi các client cổ điển, đơn giản hóa các thành phần bằng cách chuyển chức năng ít được sử dụng đến một trung gian chia sẻ. Trung gian cũng có thể được sử dụng để cải thiện khả năng mở rộng của hệ thống bằng cách cho phép cân bằng tải các dịch vụ trên nhiều mạng và bộ xử lý.

Nhược điểm chính của các hệ thống lớp là chúng tăng overhead và độ trễ trong việc xử lý dữ liệu, làm giảm hiệu suất được cảm nhận bởi người dùng. Đối với một hệ thống dựa trên mạng hỗ trợ các ràng buộc cache, điều này có thể được bù đắp bằng các lợi ích của việc cache chia sẻ tại các trung gian. Việc đặt các cache chia sẻ tại ranh giới của một lĩnh vực tổ chức có thể mang lại lợi ích hiệu suất đáng kể. Các lớp như vậy cũng cho phép các chính sách bảo mật được áp dụng vào dữ liệu vượt qua ranh giới tổ chức, như yêu cầu của các tường lửa .

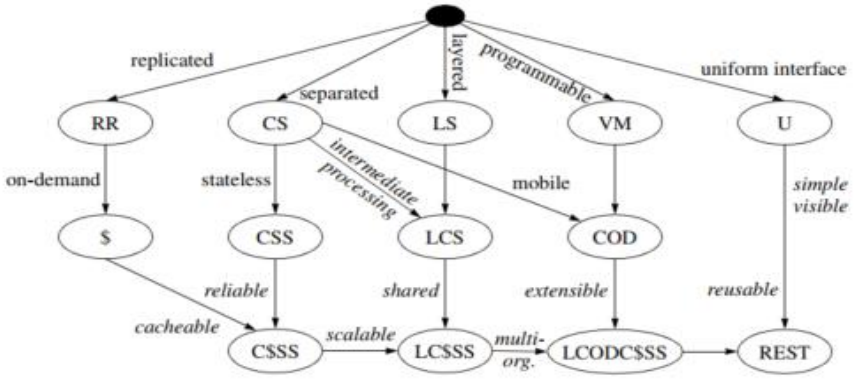


Hình 2.7: REST

Sự kết hợp giữa hệ thống lớp và ràng buộc giao diện đồng nhất tạo ra các tính chất kiến trúc tương tự như kiểu kiến trúc ống và bộ lọc đồng nhất. Mặc dù tương tác REST là hai chiều, các luồng dữ liệu hypermedia cỡ lớn có thể được xử lý như một mạng luồng dữ liệu, với các thành phần bộ lọc được áp dụng một cách chọn lọc vào luồng dữ liệu để biến đổi nội dung khi nó đi qua. Trong REST, các thành phần trung gian có thể chuyển đổi nội dung của các tin nhắn một cách tích cực vì các tin nhắn là tự mô tả và ý nghĩa của chúng có thể được nhìn thấy bởi các trung gian.

2.1.7. Code-On-Demand

REST cho phép mở rộng chức năng của client bằng cách tải xuống và thực thi mã dưới dạng applets hoặc scripts. Điều này giúp đơn giản hóa client bằng cách giảm số lượng tính năng cần được triển khai trước. Việc cho phép tải xuống tính năng sau khi triển khai cải thiện khả năng mở rộng của hệ thống. Tuy nhiên, nó cũng làm giảm khả năng nhìn thấy, do đó chỉ là một ràng buộc tùy chọn trong REST.



Hình 2.8: Xây dựng REST thông qua các ràng buộc phong cách

Khái niệm về ràng buộc tùy chọn có thể là một mâu thuẫn. Tuy nhiên, nó lại có mục đích trong thiết kế kiến trúc của một hệ thống bao gồm nhiều ranh giới tổ chức. Điều này có nghĩa là kiến trúc chỉ thu được lợi ích (và phải chịu nhược điểm) của các ràng buộc tùy chọn khi chúng được biết là có hiệu lực trong một phạm vi cụ thể của hệ thống tổng thể. Ví dụ, nếu tất cả phần mềm client trong một tổ chức được biết là hỗ trợ Java applets, các dịch vụ trong tổ chức đó có thể được xây dựng sao cho họ tận dụng được tính năng nâng cao thông qua việc tải xuống các lớp Java. Tuy nhiên, đồng thời, tường lửa của tổ chức có thể ngăn chặn việc chuyển Java applets từ các nguồn bên ngoài, và vì vậy đối với toàn bộ Web, sẽ có vẻ như những client đó không hỗ trợ mã theo yêu cầu. Ràng buộc tùy chọn cho phép thiết kế một kiến trúc hỗ trợ hành vi mong muốn trong trường hợp chung, nhưng với sự hiểu biết rằng nó có thể bị vô hiệu hóa trong một số ngữ cảnh cụ thể.

2.1.8. Tóm tắt nguồn gốc

REST được hình thành từ một tập hợp các ràng buộc kiến trúc được lựa chọn vì các đặc tính mà chúng tạo ra trên các kiến trúc tiềm năng. Mặc dù mỗi ràng buộc này

có thể được xem xét độc lập, việc mô tả chúng dựa trên nguồn gốc từ các kiểu kiến trúc phổ biến làm cho việc hiểu lí do sau lựa chọn trở nên dễ dàng hơn. Hình 2-8 minh họa sự phát triển của các ràng buộc của REST dựa trên các kiểu kiến trúc mạng.

2.2. Các phần tử trong kiến trúc REST

REST(Representational State Transfer) là một sự trừu tượng về các yếu tố kiến trúc bên trong một hệ thống siêu phương tiện phân tán. REST không quan tâm đến các chi tiết về cài đặt thành phần và cú pháp giao thức để tập trung vào vai trò của các thành phần, các ràng buộc trong việc tương tác với các thành phần khác, và cách chúng hiểu và xử lý các yếu tố dữ liệu quan trọng. Nó bao gồm các ràng buộc cơ bản đối với các thành phần, kết nối và dữ liệu xác định cơ sở của kiến trúc Web, và do đó là bản chất của hành vi của nó như một ứng dụng dựa trên mạng.

2.2.1. Yếu tố dữ liệu

REST tập trung vào việc hiểu biết chia sẻ về các loại dữ liệu với siêu dữ liệu, nhưng hạn chế phạm vi của những gì được tiết lộ thành một giao diện chuẩn hóa. Các thành phần của REST giao tiếp bằng cách truyền một biểu diễn của một tài nguyên trong một định dạng phù hợp với một trong số các loại dữ liệu tiêu chuẩn đang phát triển, được chọn động dựa trên khả năng hoặc mong muốn của người nhận và bản chất của tài nguyên. Cho dù biểu diễn có cùng định dạng với nguồn gốc thô, hay được xuất phát từ nguồn gốc, vẫn được giấu sau giao diện. Các lợi ích của kiểu đối tượng di động được xấp xỉ thông qua việc gửi một biểu diễn bao gồm các chỉ thị trong định dạng dữ liệu tiêu chuẩn của một bộ máy đóng gói. Do đó, REST đạt được sự tách biệt về quan tâm của kiểu client-server mà không gặp vấn đề về khả năng mở rộng của máy chủ, cho phép ẩn thông tin thông qua một giao diện tổng quát để kích hoạt việc bao gói và tiến hóa của các dịch vụ, và cung cấp một loạt các chức năng đa dạng thông qua các công cụ tính năng có thể tải xuống.

Bảng 1: Phần tử dữ liệu REST

Phần tử dữ liệu	**Ví dụ trên Web**
Nguồn tài nguyên	Mục tiêu đích đến dự kiến của một liên kết siêu văn bản
Nhận diện tài nguyên	URL, URN
Sự biểu diễn	Tài liệu HTML, ảnh JPEG
Siêu dữ liệu biểu diễn	Loại phương tiện, thời gian chỉnh sửa lần cuối
Siêu dữ liệu tài nguyên	Liên kết nguồn, tùy chọn, thay đổi
Dữ liệu điều khiển	Điều khiển bộ nhớ đệm

2.2.1.1. Tài nguyên và mã định danh tài nguyên

Trong REST, trừu tượng chính của thông tin là một tài nguyên. Bất kỳ thông tin nào có thể được đặt tên có thể trở thành một tài nguyên: một tài liệu hoặc hình ảnh, một dịch vụ thời gian, một bộ sưu tập các tài nguyên khác, một đối tượng không ảo. Nói cách khác, bất kỳ khái niệm nào có thể là mục tiêu của một tham chiếu siêu văn bản phù hợp với định nghĩa của một tài nguyên. Một tài nguyên là một ánh xạ khái niệm đến một tập hợp các thực thể, không phải là thực thể tương ứng với ánh xạ tại bất kỳ thời điểm cụ thể nào.

Một cách chính xác hơn, một tài nguyên R là một hàm thành viên thay đổi theo thời gian MR(t), mà cho thời điểm t tương ứng với một tập hợp các thực thể hoặc giá trị có giá trị tương đương. Các giá trị trong tập hợp có thể là biểu diễn của tài nguyên hoặc định danh của tài nguyên. Một tài nguyên có thể ánh xạ đến tập hợp rỗng, điều này cho phép tham chiếu đến một khái niệm trước khi bất kỳ thực hiện nào của khái niệm đó tồn tại - một khái niệm mà trước đây là lạ lẫm với hầu hết các hệ thống siêu

văn bản trước Web. Một số tài nguyên là tĩnh trong ý nghĩa rằng khi kiểm tra vào bất kỳ thời điểm sau khi chúng được tạo, chúng luôn tương ứng với cùng một tập hợp giá trị. Những tài nguyên khác có mức độ biến thiên cao về giá trị của chúng theo thời gian. Điều duy nhất cần phải là tĩnh cho một tài nguyên là ngữ nghĩa của ánh xạ, vì ngữ nghĩa chính là điều làm cho một tài nguyên khác biệt so với tài nguyên khác.

Định nghĩa trừu tượng này của một tài nguyên cho phép các đặc điểm chính của kiến trúc Web. Thứ nhất, nó cung cấp tính chung chung bằng cách bao quát nhiều nguồn thông tin mà không phân biệt chúng một cách nhân tạo theo loại hoặc cách thức triển khai. Thứ hai, nó cho phép việc ràng buộc trễ của tham chiếu đến một biểu diễn, cho phép đàm phán nội dung diễn ra dựa trên các đặc điểm của yêu cầu. Cuối cùng, nó cho phép tham chiếu đến khái niệm thay vì một biểu diễn duy nhất của khái niệm đó, do đó loại bỏ nhu cầu thay đổi tất cả các liên kết hiện có mỗi khi biểu diễn thay đổi

REST sử dụng một định danh tài nguyên để xác định tài nguyên cụ thể tham gia trong tương tác giữa các thành phần. Các kết nối REST cung cấp một giao diện tổng quát để truy cập và thao tác tập hợp giá trị của một tài nguyên, bất kể cách thức ánh xạ thành viên được định nghĩa như thế nào hoặc loại phần mềm nào đang xử lý yêu cầu. Cơ quan đặt tên đã gán định danh tài nguyên, tạo điều kiện để tham chiếu đến tài nguyên, chịu trách nhiệm duy trì tính hợp lệ ngữ nghĩa của ánh xạ theo thời gian (tức là đảm bảo rằng chức năng thành viên không thay đổi).

Các hệ thống siêu văn bản truyền thống thường hoạt động trong môi trường đóng, sử dụng các định danh nút hoặc tài liệu duy nhất thay đổi mỗi khi thông tin thay đổi, phụ thuộc vào máy chủ liên kết để duy trì các tham chiếu một cách riêng biệt khỏi nội dung. Vì các máy chủ liên kết tập trung là điều không mong muốn đối với quy mô khổng lồ và yêu cầu miền tổ chức đa cơ sở của Web, REST thay vào đó dựa vào tác giả chọn một định danh tài nguyên phù hợp nhất với bản chất của khái niệm được xác định. Tự nhiên, chất lượng của một định danh thường tương ứng với số tiền được chi để duy trì tính hợp lệ của nó, điều này dẫn đến việc các liên kết bị hỏng khi thông tin được hỗ trợ kém) di chuyển hoặc biến mất theo thời gian.

2.2.1.2. Sự biểu diễn

Các thành phần REST thực hiện các hành động trên một tài nguyên bằng cách sử dụng một biểu diễn để ghi lại trạng thái hiện tại hoặc dự định của tài nguyên đó và chuyển đổi biểu diễn đó giữa các thành phần. Một biểu diễn là một chuỗi các byte, cộng với siêu dữ liệu biểu diễn để mô tả những byte đó. Các tên thông thường khác được sử dụng nhưng không chính xác hơn cho một biểu diễn bao gồm: tài liệu, tập tin, và thực thể, phiên bản hoặc trường hợp của thông điệp HTTP.

Một biểu diễn bao gồm dữ liệu, siêu dữ liệu mô tả dữ liệu và đôi khi cũng bao gồm siêu dữ liệu để mô tả siêu dữ liệu (thường để xác minh tính toàn vẹn của thông điệp). Siêu dữ liệu có dạng các cặp tên-giá trị, trong đó tên tương ứng với một tiêu chuẩn xác định cấu trúc và ngữ nghĩa của giá trị. Các thông điệp phản hồi có thể bao gồm cả siêu dữ liệu biểu diễn và siêu dữ liệu tài nguyên: thông tin về tài nguyên không đặc thù cho biểu diễn được cung cấp.

Dữ liệu điều khiển xác định mục đích của một thông điệp giữa các thành phần, như hành động đang được yêu cầu hoặc ý nghĩa của một phản hồi. Nó cũng được sử dụng để tham số hóa các yêu cầu và ghi đè lên hành vi mặc định của một số yếu tố kết nối. Ví dụ, hành vi bộ nhớ cache có thể được sửa đổi bằng dữ liệu điều khiển đi kèm trong thông điệp yêu cầu hoặc phản hồi.

Tùy thuộc vào dữ liệu điều khiển của thông điệp, một biểu diễn cụ thể có thể chỉ định trạng thái hiện tại của tài nguyên được yêu cầu, trạng thái mong muốn cho tài nguyên được yêu cầu, hoặc giá trị của một số tài nguyên khác, như là biểu diễn của dữ liệu đầu vào trong mẫu truy vấn của một khách hàng, hoặc biểu diễn của một số điều kiện lỗi cho một phản hồi. Ví dụ, việc tác giả từ xa của một tài nguyên đòi hỏi tác giả gửi một biểu diễn đến máy chủ, từ đó thiết lập một giá trị cho tài nguyên đó có thể được lấy bằng các yêu cầu sau này. Nếu tập hợp giá trị của một tài nguyên tại một thời điểm nhất định bao gồm nhiều biểu diễn, thương lượng nội dung có thể được sử dụng để chọn biểu diễn tốt nhất để bao gồm trong một thông điệp cụ thể.

Định dạng dữ liệu của một biểu diễn được biết đến như là một loại phương tiện. Một biểu diễn có thể được bao gồm trong một thông điệp và được xử lý bởi người nhận dựa trên dữ liệu điều khiển của thông điệp và bản chất của loại phương tiện. Một số loại phương tiện được thiết kế để xử lý tự động, một số được thiết kế để hiển thị cho

người dùng, và một số ít có khả năng cả hai. Các loại phương tiện hỗn hợp có thể được sử dụng để bao gồm nhiều biểu diễn trong một thông điệp duy nhất.

Thiết kế của một loại phương tiện có thể ảnh hưởng trực tiếp đến hiệu suất mà người dùng cảm nhận được của một hệ thống siêu phương tiện phân tán. Mọi dữ liệu cần phải được nhận trước khi người nhận có thể bắt đầu hiển thị biểu diễn sẽ làm tăng thêm thời gian trễ trong một tương tác. Một định dạng dữ liệu đặt thông tin hiển thị quan trọng nhất ở đầu trang, sao cho thông tin ban đầu có thể được hiển thị một cách từ từ trong khi phần còn lại của thông tin đang được nhận, sẽ mang lại hiệu suất cảm nhận của người dùng tốt hơn nhiều so với một định dạng dữ liệu cần phải nhận toàn bộ trước khi bắt đầu hiển thị.

2.2.2. Kết nối

Bảng 2: Các loại kết nối trong REST

Loại kết nối	**Mô Tả**
Khách hàng	Khởi tạo yêu cầu tài nguyên và nhận phản hồi từ máy chủ
Máy chủ	Nghe yêu cầu từ khách hàng, xử lý và phản hồi các yêu cầu đó
Bộ nhớ Cache	Lưu trữ phản hồi của yêu cầu để sử dụng lại cho các yêu cầu tương tự sau này
Trình giải quyết	Giải quyết định danh tài nguyên đến vị trí mạng thực tế
Đường hầm	Thiết lập một liên kết truyền thông cho tài nguyên thông qua các trung gian, thường được sử dụng cho proxy hoặc cổng
Chia sẻ	Quản lý tài nguyên được chia sẻ qua nhiều tương tác, tăng cường hiệu suất và khả năng phản hồi

REST dựa vào các loại kết nối khác nhau được tóm tắt trong *Bảng 2* để đóng gói các hoạt động truy cập tài nguyên và truyền tải biểu diễn tài nguyên. Các kết nối này

trình bày giao diện trừu tượng cho giao tiếp thành phần, cho phép tách biệt rõ ràng giữa các mối quan tâm và ẩn đi các triển khai cơ bản của tài nguyên và cơ chế giao tiếp. Trừu tượng hóa này cũng cho phép thay thế: nếu người dùng tương tác chỉ qua một giao diện trừu tượng, triển khai có thể được thay thế mà không ảnh hưởng đến người dùng. Các kết nối, thông qua việc quản lý giao tiếp mạng, cho phép chia sẻ thông tin qua nhiều tương tác, tăng cường hiệu suất và khả năng phản hồi.

Tất cả các tương tác trong REST đều không có trạng thái (stateless). Nghĩa là, mỗi yêu cầu chứa đựng đầy đủ thông tin cần thiết để một kết nối hiểu được yêu cầu, độc lập với bất kỳ yêu cầu nào có thể trước đó. Hạn chế này thực hiện bốn chức năng:

1) Loại bỏ bất kỳ nhu cầu nào để các kết nối giữ lại trạng thái ứng dụng giữa các yêu cầu, từ đó giảm sự tiêu thụ của tài nguyên vật lý và cải thiện khả năng mở rộng;

2) Cho phép các tương tác được xử lý song song mà không cần yêu cầu rằng cơ chế xử lý hiểu ngữ nghĩa của tương tác;

3) Cho phép một trung gian xem và hiểu một yêu cầu một cách độc lập, điều này có thể cần thiết khi các dịch vụ được sắp xếp động;

4) Buộc mọi thông tin có thể ảnh hưởng đến khả năng tái sử dụng của một phản hồi được lưu cache phải có mặt trong mỗi yêu cầu.

Khách hàng tải xuống tệp kê khai (file index) mô tả các phân đoạn luồng có sẵn và tốc độ bit tương ứng của chúng. Trong quá trình khởi động luồng, khách hàng thường yêu cầu các chunk file từ luồng tốc độ bit thấp nhất. Nếu máy khách thấy rằng thông lượng mạng lớn hơn tốc độ bit của phân đoạn đã tải xuống, thì nó sẽ yêu cầu phân đoạn tốc độ bit cao hơn. Sau đó, nếu máy khách nhận thấy thông lượng mạng bị giảm sút, nó sẽ yêu cầu phân đoạn tốc độ bit thấp hơn.

Giao diện của kết nối tương tự như việc gọi thủ tục, nhưng có những khác biệt quan trọng trong việc truyền các tham số và kết quả. Các tham số đầu vào bao gồm dữ liệu điều khiển yêu cầu, một định danh tài nguyên chỉ ra đích của yêu cầu, và một biểu diễn tùy chọn. Các tham số đầu ra bao gồm dữ liệu điều khiển phản hồi, siêu dữ liệu tài nguyên tùy chọn và một biểu diễn tùy chọn. Từ một quan điểm trừu tượng, việc gọi là đồng bộ, nhưng cả tham số đầu vào và đầu ra có thể được truyền như dòng dữ liệu.

Nói cách khác, việc xử lý có thể được gọi trước khi giá trị của các tham số được biết hoàn toàn, do đó tránh được sự trễ trên quá trình xử lý dữ liệu lớn trong lô.

Các loại kết nối chính là khách hàng và máy chủ. Sự khác biệt cốt lõi giữa hai loại này là khách hàng khởi tạo giao tiếp bằng cách gửi yêu cầu, trong khi máy chủ lắng nghe các kết nối và phản hồi các yêu cầu để cung cấp quyền truy cập vào dịch vụ của nó. Một thành phần có thể bao gồm cả kết nối khách hàng và máy chủ.

Một loại kết nối thứ ba, kết nối cache, có thể được đặt trên giao diện của kết nối khách hàng hoặc máy chủ để lưu trữ các phản hồi có thể lưu cache cho các tương tác hiện tại để có thể sử dụng lại cho các tương tác được yêu cầu sau. Cache có thể được sử dụng bởi một khách hàng để tránh lặp lại giao tiếp mạng, hoặc bởi một máy chủ để tránh lặp lại quá trình tạo ra một phản hồi, cả hai trường hợp đều giúp giảm thời gian trễ trong tương tác. Cache thường được triển khai trong không gian địa chỉ của kết nối sử dụng nó.

Một số kết nối cache là chia sẻ, có nghĩa là các phản hồi được lưu cache của nó có thể được sử dụng để trả lời cho một khách hàng khác ngoài khách hàng mà phản hồi ban đầu được nhận. Việc chia sẻ cache có thể hiệu quả trong việc giảm tác động của đám đông đột ngột lên tải của một máy chủ phổ biến, đặc biệt khi việc lưu cache được tổ chức theo cấu trúc phân cấp để bao gồm nhóm lớn người dùng, như những người trong mạng nội bộ của một công ty, khách hàng của một nhà cung cấp dịch vụ Internet, hoặc các trường đại học chia sẻ một mạng lưới quốc gia. Tuy nhiên, việc chia sẻ cache cũng có thể dẫn đến lỗi nếu phản hồi được lưu cache không khớp với điều gì đã được... REST cố gắng cân nhắc giữa mong muốn về hành vi bộ nhớ cache minh bạch và việc sử dụng mạng một cách hiệu quả, thay vì giả định rằng việc minh bạch tuyệt đối luôn là cần thiết.

Khả năng của một bộ nhớ cache để xác định khả năng lưu cache của một phản hồi xuất phát từ giao diện của nó, không phụ thuộc vào các chi tiết cụ thể liên kết với mỗi tài nguyên. Thông thường, phản hồi cho một yêu cầu truy xuất được xem xét là có thể lưu cache theo mặc định, trong khi các phản hồi cho các loại yêu cầu khác thì không. Tuy nhiên, nếu xác thực người dùng là một phần của yêu cầu hoặc nếu phản hồi chỉ ra rằng nó không nên được chia sẻ, thì phản hồi chỉ có thể được lưu cache bởi

một bộ nhớ cache không chia sẻ. Các thành phần có thể ghi đè lên các giá trị mặc định này bằng cách bao gồm dữ liệu điều khiển, chỉ định tương tác có thể lưu cache, không thể lưu cache hoặc có thể lưu cache trong một khoảng thời gian hạn chế.

Một bộ giải quyết (resolver) dịch các định danh tài nguyên, phần hoặc toàn bộ, thành thông tin địa chỉ mạng cần thiết để thiết lập kết nối giữa các thành phần. Ví dụ, hầu hết các URI chứa tên máy chủ DNS như cơ chế để xác định cơ quan đặt tên cho tài nguyên. Để khởi tạo một yêu cầu, trình duyệt web sẽ trích xuất tên máy chủ từ URI và sử dụng một bộ giải quyết DNS để có được địa chỉ giao thức Internet (IP) cho cơ quan đó. Trong một tình huống khác, một số lược đồ nhận dạng, như URN, có thể yêu cầu một bên trung gian chuyển đổi một định danh cố định thành một địa chỉ tạm thời để truy cập tài nguyên. Sử dụng một hoặc nhiều bộ giải quyết trung gian có thể cải thiện tính bền vững của các tham chiếu tài nguyên thông qua sự chuyển hướng, tuy nhiên cách tiếp cận này đóng góp vào thời gian trễ của yêu cầu.

Loại kết nối cuối cùng là một "đường hầm" (tunnel), đơn giản là truyền thông qua ranh giới kết nối, như một tường lửa hoặc cổng mạng cấp thấp hơn. Lý do duy nhất mà nó được mô hình hóa như một phần của REST và không được trừu tượng hóa như một phần của cơ sở hạ tầng mạng là vì một số thành phần REST có thể chuyển động động từ hành vi của thành phần hoạt động sang hành vi của một đường hầm. Ví dụ chính là một proxy HTTP chuyển đổi sang một đường hầm trong phản hồi cho một yêu cầu phương thức CONNECT, từ đó cho phép khách hàng của nó trực tiếp giao tiếp với một máy chủ từ xa bằng cách sử dụng một giao thức khác, như TLS mà không cho phép các proxy. Đường hầm biến mất khi cả hai đầu kết thúc việc truyền thông của họ.

2.2.3. Các thành phần

Các thành phần REST, được tóm tắt trong *Bảng 3*, được phân loại theo vai trò của chúng trong một tổng thể.

Bảng 3: Các thành phần REST

Thành phần	Ví dụ trên Web
Máy chủ nguồn	Apache httpd, Microsoft IIS
Gateway(đầu vào)	Squid, CGI, Reverse Proxy
Proxy(ủy quyền)	CERN Proxy, Netscape Proxy, Gauntlet
User agent(tương tác người dùng)	Netscape Navigator, Lynx, MOMspider

Một user agent sử dụng một kết nối khách hàng để khởi tạo một yêu cầu và trở thành người nhận cuối cùng của phản hồi. Ví dụ phổ biến nhất là trình duyệt web, mà cung cấp truy cập vào các dịch vụ thông tin và hiển thị các phản hồi dịch vụ theo nhu cầu của ứng dụng.

Một máy chủ nguồn (origin server) sử dụng một kết nối máy chủ để quản lý không gian tên cho một tài nguyên được yêu cầu. Nó là nguồn thông tin cơ bản cho các biểu diễn của tài nguyên của nó và phải là người nhận cuối cùng của bất kỳ yêu cầu nào cố ý sửa đổi giá trị của tài nguyên của nó. Mỗi Máy chủ nguồn cung cấp một giao diện chung cho các dịch vụ của mình dưới dạng một cấu trúc tài nguyên phân cấp. Các chi tiết thực hiện tài nguyên được che giấu sau giao diện.

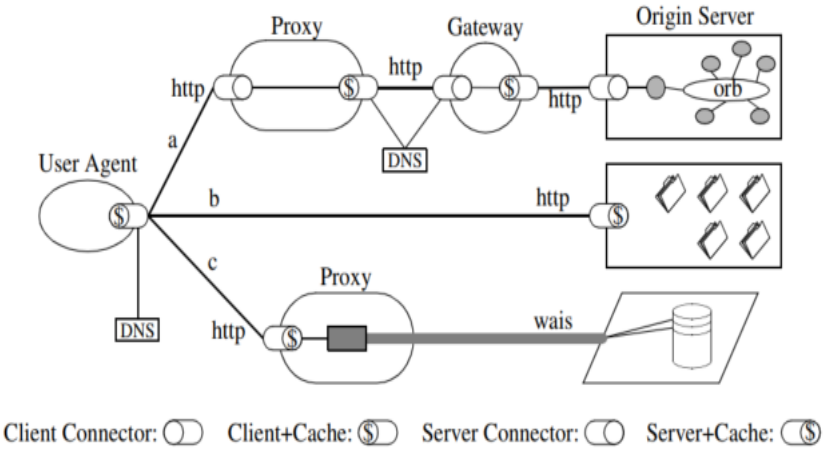
Các thành phần trung gian hoạt động như một khách hàng và một máy chủ để chuyển tiếp, có thể có sự dịch chuyển, yêu cầu và phản hồi. Một thành phần proxy là một trung gian được lựa chọn bởi một khách hàng để cung cấp bao gói giao diện của các dịch vụ khác, dịch chuyển dữ liệu, cải thiện hiệu suất hoặc bảo vệ bảo mật. Một cổng (còn gọi là proxy ngược) là một thành phần trung gian được mạng hoặc máy chủ nguồn áp đặt để cung cấp bao gói giao diện của các dịch vụ khác, để dịch chuyển dữ liệu, cải thiện hiệu suất hoặc thực hiện bảo mật. Lưu ý rằng sự khác biệt giữa một proxy và một cổng là khách hàng quyết định khi nào sẽ sử dụng một proxy.

2.3. Góc nhìn kiến trúc REST

Bây giờ khi đã hiểu được các yếu tố kiến trúc REST độc lập, có thể sử dụng các quan điểm(góc nhìn) của kiến trúc để mô tả cách các yếu tố hoạt động cùng nhau để tạo thành một kiến trúc. Ba loại góc nhìn - góc nhìn quy trình, kết nối và dữ liệu - rất hữu ích để làm sáng tỏ các nguyên tắc thiết kế của REST.

2.3.1. Góc nhìn về quy trình

Trong một kiến trúc chủ yếu hiệu quả trong việc đưa ra các mối quan hệ tương tác giữa các thành phần bằng cách tiết lộ đường dẫn của dữ liệu khi nó chảy qua hệ thống. Thật không may, mối tương tác của một hệ thống thực tế thường liên quan đến một số lượng lớn các thành phần, dẫn đến một cái nhìn tổng thể bị che khuất bởi các chi tiết.



Hình 2.9: Quy trình của kiến trúc dựa trên REST

Một đại diện người dùng được mô tả trong ba tương tác song song: a, b và c. Các tương tác không được đáp ứng bởi bộ nhớ cache của kết nối khách hàng của đại diện người dùng, vì vậy mỗi yêu cầu đã được định tuyến đến nguồn tài nguyên theo các thuộc tính của mỗi định danh tài nguyên và cấu hình của kết nối khách hàng. Yêu cầu (a) đã được gửi đến một proxy cục bộ, sau đó truy cập vào một cổng bộ nhớ cache được tìm thấy thông qua tìm kiếm DNS, mà tiếp tục chuyển tiếp yêu cầu để được đáp ứng bởi một máy chủ nguồn mà tài nguyên nội bộ của nó được định nghĩa bởi một kiến trúc trung gian yêu cầu đóng gói. Yêu cầu (b) được gửi trực tiếp đến một máy chủ nguồn, có khả năng đáp ứng yêu cầu từ bộ nhớ cache của chính nó. Yêu cầu (c) được gửi đến một proxy có khả năng truy cập trực tiếp vào WAIS, một dịch vụ thông tin

khác biệt với kiến trúc Web, và dịch phản hồi từ WAIS thành định dạng được nhận diện bởi giao diện kết nối chung. Mỗi thành phần chỉ nhận thức về tương tác với các kết nối khách hàng hoặc máy chủ của riêng họ.

Hình 2.9 cung cấp một mẫu góc nhìn quy trình từ một kiến trúc dựa trên REST tại một thời điểm cụ thể trong quá trình xử lý ba yêu cầu song song.

Sự tách biệt của REST giữa khách hàng và máy chủ đơn giản hóa việc triển khai thành phần, giảm độ phức tạp của ngữ nghĩa kết nối, cải thiện hiệu suất điều chỉnh và tăng cường khả năng mở rộng của các thành phần máy chủ thuần túy. Ràng buộc hệ thống theo lớp cho phép các trung gian - proxy, cổng và tường lửa - được giới thiệu tại các điểm khác nhau trong giao tiếp mà không thay đổi các giao diện giữa các thành phần, do đó cho phép chúng hỗ trợ trong việc dịch chuyển giao tiếp hoặc cải thiện hiệu suất thông qua việc bộ nhớ cache chia sẻ quy mô lớn. REST cho phép xử lý trung gian bằng cách ràng buộc các thông điệp phải mô tả chính nó: tương tác không có trạng thái giữa các yêu cầu, các phương thức tiêu chuẩn và các loại phương tiện được sử dụng để chỉ ra ngữ nghĩa và trao đổi thông tin, và các phản hồi chỉ rõ khả năng lưu trữ vào bộ nhớ cache.

Do các thành phần được kết nối một cách động, sự sắp xếp và chức năng của chúng cho một hành động ứng dụng cụ thể có đặc tính tương tự như kiểu pipe-and-filter. Mặc dù các thành phần REST giao tiếp qua luồng hai chiều, việc xử lý của mỗi hướng là độc lập và do đó dễ bị ảnh hưởng bởi các bộ biến đổi dòng (bộ lọc). Giao diện kết nối chung cho phép các thành phần được đặt trên luồng dựa trên các thuộc tính của mỗi yêu cầu hoặc phản hồi.

Các dịch vụ có thể được triển khai bằng cách sử dụng một cấu trúc phân cấp phức tạp của các trung gian và nhiều máy chủ nguồn phân tán. Tính không có trạng thái của REST cho phép mỗi tương tác độc lập với các tương tác khác, loại bỏ nhu cầu nhận thức về cấu trúc tổng thể của các thành phần, một nhiệm vụ không thể với một kiến trúc quy mô Internet, và cho phép các thành phần hoạt động như điểm đến hoặc trung gian, được xác định động bởi mục tiêu của mỗi yêu cầu. Các kết nối chỉ cần nhận thức về sự tồn tại của nhau trong phạm vi giao tiếp của họ, mặc dù họ có thể lưu bộ nhớ về sự tồn tại và khả năng của các thành phần khác vì lý do hiệu suất.

2.3.2. Góc nhìn về kết nối

Tập trung vào cơ chế của việc giao tiếp giữa các thành phần. Đối với một kiến trúc dựa trên REST, đặc biệt quan tâm đến các ràng buộc xác định giao diện tài nguyên chung.

Các kết nối khách hàng xem xét định danh tài nguyên để lựa chọn cơ chế giao tiếp phù hợp cho mỗi yêu cầu. Ví dụ, một khách hàng có thể được cấu hình để kết nối với một thành phần proxy cụ thể, có thể là một thành phần hoạt động như một bộ lọc chú thích, khi định danh chỉ ra rằng đó là một tài nguyên cục bộ. Tương tự, một khách hàng có thể được cấu hình để từ chối các yêu cầu cho một số phần nhỏ của các định danh.

REST không hạn chế giao tiếp đến một giao thức cụ thể, nhưng nó ràng buộc giao diện giữa các thành phần và do đó phạm vi tương tác và các giả định triển khai mà có thể được thực hiện giữa các thành phần khác nhau. Ví dụ, giao thức truyền chính của Web là HTTP, nhưng kiến trúc cũng bao gồm việc truy cập liền mạch đến các tài nguyên có nguồn từ các máy chủ mạng hiện có trước đây, bao gồm FTP, Gopher và WAIS. Tương tác với những dịch vụ đó bị hạn chế theo ngữ nghĩa của một kết nối REST. Ràng buộc này hy sinh một số lợi ích của các kiến trúc khác, như tương tác có trạng thái của một giao thức phản hồi sự liên quan như WAIS, để giữ lại lợi ích của một giao diện chung duy nhất cho ngữ nghĩa kết nối. Đổi lại, giao diện chung cho phép truy cập vào nhiều dịch vụ thông qua một proxy duy nhất. Nếu một ứng dụng cần các khả năng bổ sung của một kiến trúc khác, nó có thể triển khai và gọi các khả năng đó như một hệ thống riêng biệt chạy song song, tương tự như cách kiến trúc Web tương tác với các tài nguyên "telnet" và "mailto".

2.3.3. Góc nhìn về dữ liệu

Một kiến trúc tiết lộ trạng thái ứng dụng khi thông tin chảy qua các thành phần. Với REST được đặc biệt hướng tới các hệ thống thông tin phân tán, nó xem xét một ứng dụng như một cấu trúc liên kết của thông tin và các lựa chọn điều khiển thông qua đó người dùng có thể thực hiện một nhiệm vụ mong muốn. Ví dụ, tra từ trong từ điển trực tuyến là một ứng dụng, như cũng như việc tham quan một bảo tàng ảo, hoặc xem

lại một bộ ghi chú để học cho một kỳ thi. Mỗi ứng dụng xác định mục tiêu cho hệ thống cơ sở của nó, thông qua đó hiệu suất của hệ thống có thể được đánh giá.

Tương tác giữa các thành phần diễn ra dưới dạng các thông điệp có kích thước linh hoạt. Các thông điệp nhỏ hoặc trung bình được sử dụng cho ngữ nghĩa điều khiển, nhưng phần lớn công việc của ứng dụng được thực hiện thông qua các thông điệp lớn chứa một biểu diễn tài nguyên hoàn chỉnh. Dạng ngữ nghĩa yêu cầu phổ biến nhất là việc lấy một biểu diễn của một tài nguyên (ví dụ, phương thức "GET" trong HTTP), mà thường có thể được lưu vào bộ nhớ cache để sử dụng lại sau này.

REST tập trung tất cả trạng thái điều khiển vào các biểu diễn nhận được như là phản hồi cho các tương tác. Mục tiêu là cải thiện khả năng mở rộng của máy chủ bằng cách loại bỏ bất kỳ nhu cầu nào của máy chủ để duy trì sự nhận thức về trạng thái của khách hàng vượt quá yêu cầu hiện tại. Do đó, trạng thái của một ứng dụng được xác định bởi các yêu cầu đang chờ xử lý, cấu trúc liên kết của các thành phần kết nối (một số trong số đó có thể là bộ lọc dữ liệu đệm), các yêu cầu hoạt động trên các kết nối đó, luồng dữ liệu của các biểu diễn phản hồi cho các yêu cầu đó và xử lý của các biểu diễn đó khi chúng được nhận bởi trình duyệt.

Một ứng dụng đạt trạng thái ổn định khi nó không có yêu cầu nào đang chờ xử lý, tức là nó không có yêu cầu nào đang chờ xử lý và tất cả các phản hồi cho bộ yêu cầu hiện tại của nó đã được nhận hoặc đã được nhận đến mức có thể xem xét chúng như một luồng dữ liệu biểu diễn. Đối với ứng dụng trình duyệt, trạng thái này tương ứng với "trang web," bao gồm biểu diễn chính và các biểu diễn phụ, như hình ảnh nằm trong nội dung, các ứng dụng nhúng và bảng kiểu. Tầm quan trọng của các trạng thái ổn định của ứng dụng thể hiện trong ảnh hưởng của chúng đến cả hiệu suất được cảm nhận bởi người dùng và tính không đều của lưu lượng yêu cầu mạng.

Hiện tượng hiệu suất được cảm nhận bởi người dùng của ứng dụng trình duyệt được xác định bởi độ trễ giữa các trạng thái ổn định: khoảng thời gian giữa việc chọn một liên kết siêu phương tiện trên một trang web và thời điểm mà thông tin có thể sử dụng được đã được hiển thị cho trang web tiếp theo. Tối ưu hiệu suất của trình duyệt do đó tập trung vào việc giảm thiểu độ trễ giao tiếp này.

Do các kiến trúc dựa trên REST giao tiếp chủ yếu thông qua việc truyền các biểu diễn của tài nguyên, độ trễ có thể bị ảnh hưởng bởi cả thiết kế của các giao thức truyền thông và thiết kế của các định dạng dữ liệu biểu diễn. Khả năng hiển thị dữ liệu phản hồi một cách tăng dần khi nhận được phụ thuộc vào thiết kế của loại phương tiện và sự có sẵn của thông tin bố trí(kích thước hình ảnh nằm trong nội dung)trong mỗi biểu diễn.

Một quan sát thú vị là yêu cầu mạng hiệu suất nhất là yêu cầu không sử dụng mạng. Nói cách khác, khả năng tái sử dụng một phản hồi được lưu vào bộ nhớ cache dẫn đến cải thiện đáng kể về hiệu suất ứng dụng. Mặc dù việc sử dụng bộ nhớ cache làm tăng độ trễ cho mỗi yêu cầu cụ thể do chi phí tìm kiếm, thì độ trễ trung bình của yêu cầu được giảm đáng kể khi ngay cả một phần nhỏ yêu cầu dẫn đến kết quả chính xác từ bộ nhớ cache.

Trạng thái điều khiển tiếp theo của một ứng dụng đặt ở biểu diễn của tài nguyên đầu tiên được yêu cầu, do đó việc nhận biểu diễn đầu tiên này được ưu tiên. Tương tác REST được cải thiện thông qua các giao thức "phản hồi trước và suy nghĩ sau". Nói cách khác, một giao thức đòi hỏi nhiều tương tác mỗi hành động của người dùng, để thực hiện các việc như đàm phán khả năng tính năng trước khi gửi phản hồi nội dung, sẽ được cảm nhận chậm hơn so với một giao thức gửi cái có khả năng tốt nhất trước tiên và sau đó cung cấp một danh sách các lựa chọn cho khách hàng lấy nếu phản hồi đầu tiên không đáp ứng được mong đợi.

Trạng thái ứng dụng được điều khiển và lưu trữ bởi trình duyệt người dùng và có thể bao gồm các biểu diễn từ nhiều máy chủ. Ngoài việc giải phóng máy chủ khỏi vấn đề mở rộng của việc lưu trữ trạng thái, điều này cho phép người dùng can thiệp trực tiếp vào trạng thái (ví dụ: lịch sử của trình duyệt web), dự đoán các thay đổi trong trạng thái đó (ví dụ: bản đồ liên kết và tải trước các biểu diễn) và chuyển đổi từ một ứng dụng sang ứng dụng khác (ví dụ: các đánh dấu và hộp thoại nhập URI).

Ứng dụng mẫu do đó là một hệ thống di chuyển từ trạng thái này sang trạng thái khác bằng cách xem xét và lựa chọn từ các chuyển đổi trạng thái thay thế trong bộ biểu diễn hiện tại. Không ngạc nhiên khi điều này chính xác phù hợp với giao diện người dùng của trình duyệt siêu phương tiện. Tuy nhiên, kiểu mẫu này không giả định

rằng tất cả các ứng dụng đều là trình duyệt. Trên thực tế, chi tiết ứng dụng được ẩn khỏi máy chủ bởi giao diện kết nối thông dụng của các kết nối, và do đó một trình duyệt người dùng có thể hoàn toàn là một robot tự động thực hiện truy xuất thông tin cho một dịch vụ chỉ mục, một trợ lý cá nhân tìm kiếm dữ liệu phù hợp với một số tiêu chí nhất định, hoặc một con nhện duy trì bận rộn kiểm tra thông tin để tìm kiếm các liên kết hỏng hoặc nội dung đã được sửa đổi.

Tóm tắt về REST:

REST là một kiến trúc phần mềm, mà trong đó định nghĩa một tập các ràng buộc, các quy tắc, tập trung vào việc sử dụng các giao thức web tiêu chuẩn như HTTP để tương tác giữa các thành phần của hệ thống. Khi mà phát triển các dịch vụ Web(Web Service) cần phải tuân theo những quy tắc và ràng buộc của REST.

REST đã trở thành một trong những phong cách kiến trúc phổ biến trong việc xây dựng các dịch vụ web linh hoạt và mở rộng, cho phép các ứng dụng tương tác với nhau một cách hiệu quả và linh hoạt thông qua giao thức HTTP và các nguyên tắc thiết kế cơ bản. Khi mà một Web Service mà có những API tuân thủ các quy tắc của REST định nghĩa thì nó được gọi là RESTful API. Và để hiểu rõ hơn và RESTful API thì chúng sẽ biết ở phần 2.4.

2.4. Kiến trúc RESTful API

Trước khi khám phá RESTful API, hãy nắm vững một số khái niệm quan trọng trong lĩnh vực công nghệ phần mềm.

2.4.1. Khái niệm API

API được viết tắt của Application Programming Interface là một tập hợp các câu lệnh(commands), các hàm(functions), các giao thức(protocols), … giúp hai ứng dụng(application) có thể tương tác và trao đổi dữ liệu qua lại được với nhau.

API định rõ các phương thức, cấu trúc dữ liệu và quy tắc mà các phần mềm khác có thể sử dụng để truy cập và thực hiện các chức năng cụ thể của ứng dụng hoặc hệ thống. Cụ thể, các API có thể cung cấp các giao diện cho việc truy xuất dữ liệu từ cơ

sở dữ liệu, thực hiện các chức năng cụ thể như xử lý thông tin, gửi và nhận dữ liệu qua mạng, quản lý tài nguyên, và nhiều chức năng khác.

Một số loại API :

Private API còn được gọi là API nội bộ, được sử dụng để kết nối các thành phần phần mềm và ứng dụng khác nhau trong một doanh nghiệp và chúng không được sử dụng bởi bên thứ ba.

Partner API đây là những API chỉ dành cho các nhà phát triển thuộc bên thứ ba, được ủy quyền để hỗ trợ mỗi quan hệ hợp tác giữa doanh nghiệp với doanh nghiệp(ví dụ là một của hàng bán hàng muốn tích hợp thanh toán online, thì các cổng thanh toán sẽ cung cấp API để tích hợp vào).

Public API đây là những API dành chung cho mọi người, bất kỳ ai cũng có thể sử dụng. Chúng có thể có hoặc không yêu cầu về ủy quyền và chi phí(ví dụ là Google Maps API).

Và trong nội dung đồ án này sẽ tìm hiểu về **RESTful API** được xây dựng dựa trên các nguyên tắc và tiêu chuẩn của của kiến trúc REST cho phép frontend và backend giao tiếp với nhau.

2.4.2. Front end và Back end

Front end là tất cả những gì bạn thấy khi điều hướng trên Internet, từ các font chữ, màu sắc cho tới các menu xổ xuống và các thanh trượt,…nói tóm lại là giao diện người dùng(User Interface). Khác với API ở chỗ UI là giữa người dùng với Application.

Back end là phần của ứng dụng hoặc hệ thống không được người dùng trực tiếp tương tác mà thay vào đó nó thực hiện các nhiệm vụ xử lý logic, lưu trữ và quản lý dữ liệu. Backend thường là nơi xử lý các yêu cầu từ phía frontend, truy xuất cơ sở dữ liệu, xử lý logic kinh doanh, và chuẩn bị dữ liệu trước khi gửi về phía frontend. Đây là phần của hệ thống mà người dùng không nhìn thấy nhưng quan trọng để ứng dụng hoạt động.

Khi phần frontend của một ứng dụng muốn gửi yêu cầu hoặc nhận dữ liệu từ phía backend, nó sẽ sử dụng các RESTful API được cung cấp bởi phần backend. RESTful API định rõ các điểm cuối (endpoints) mà phần frontend có thể truy cập và gửi yêu cầu thông qua giao thức HTTP. Nó thường đóng vai trò là cầu nối giữa phần frontend và backend trong mô hình phát triển phần mềm.

2.5. RESTful API

RESTful API là một tiêu chuẩn dùng trong việc thiết kế API cho các ứng dụng web (thiết kế Web services) được sử dụng để truyền tải và trao đổi dữ liệu giữa các ứng dụng web. RESTful là một trong những kiểu thiết kế API được sử dụng phổ biến ngày nay để cho các ứng dụng (web, mobile…) khác nhau giao tiếp với nhau. **RESTful API** buộc phải tuân theo những quy định về cách sử dụng HTTP method(như GET, POST, PUT, DELETE…) và cách định dạng các URL cho ứng dụng web để quản lý các resource mà **REST** định nghĩa ra. Các phương thức này được sử dụng để định nghĩa các hành động như lấy thông tin, tạo mới, cập nhật và xóa các tài nguyên trên server.

2.5.1. Các nguyên tắc của RESTful API

Resource:

Một tài nguyên là bất cứ thứ gì có thể được xác định thông qua một URI. Trong bước đầu tiên của quy trình trước đó, URI được người dùng nhập vào là địa chỉ của một tài nguyên tương ứng với một trang web. Trong một trang web tĩnh điển hình, mỗi trang web đều là một tài nguyên.

URI (Uniform Resource Identifier)

Khi người dùng nhập vào trình duyệt để bắt đầu tương tác trước đó là một Uniform Resource Identifier (URI). Một tên phổ biến khác cho URI là Uniform Resource Locator (URL). URI là thuật ngữ tổng quát hơn bạn có thể sử dụng để chỉ đến một vị trí (URL) hoặc một tên. Một URI là một định danh của một tài nguyên. Trong hầu hết các trường hợp, URIs là không rõ ràng đối với các client. Điều này có nghĩa là thường thì người dùng không cần phải hiểu hoặc quan tâm đến cấu trúc nội dung chính xác của URI, mà chỉ cần sử dụng nó để xác định một tài nguyên cụ thể trên

internet. Sau đó máy chủ thực hiện cập nhật trạng thái của người dùng là một tài nguyên khác. Biểu mẫu HTML được sử dụng để gửi thông tin có chứa địa chỉ (URI) của tài nguyên này được mã hóa như giá trị của thuộc tính "action" của phần tử "<form>" .

Representations

Tài liệu HTML mà máy chủ trả về cho client là một biểu diễn của tài nguyên được yêu cầu. Một biểu diễn là một bao gồm thông tin (trạng thái, dữ liệu, hoặc đánh dấu) của tài nguyên, được mã hóa bằng các định dạng như XML, JSON, hoặc HTML. Một tài nguyên có thể có một hoặc nhiều biểu diễn khác nhau. Client và server sử dụng các loại phương tiện (media types) để chỉ định loại biểu diễn đang được gửi đến đối tác nhận (client hoặc server). Hầu hết các trang web và ứng dụng thường sử dụng định dạng HTML với loại phương tiện text/html. Tương tự, khi người dùng gửi biểu mẫu, trình duyệt gửi một biểu diễn bằng định dạng mã hóa URI sử dụng loại phương tiện application/x-www-form-urlencoded.

Uniform Interface

Các khách hàng sử dụng giao thức truyền tải siêu văn bản (HTTP) để gửi các yêu cầu đến các nguồn tài nguyên và nhận phản hồi. Trong bước đầu tiên, khách hàng gửi một yêu cầu GET để lấy một tài liệu HTML. Sau đó, khách hàng gửi một yêu cầu POST để cập nhật trạng thái người dùng. Hai phương thức này là một phần của giao diện đồng nhất của HTTP. Việc sử dụng giao diện đồng nhất này làm cho các yêu cầu và phản hồi tự mô tả và rõ ràng. Ngoài hai phương thức này, giao diện này bao gồm các phương thức khác như:

- HEAD: giống với GET nhưng response trả về không có body, chỉ có header.
- PUT: ghi đè tất cả thông tin của đối tượng với những gì được gửi lên.
- PATCH: ghi đè các thông tin được thay đổi của đối tượng.
- DELETE: xóa tài nguyên trên server.
- CONNECT: thiết lập một kết nối tới server theo URI.
- OPTIONS: mô tả các tùy chọn giao tiếp cho resource.

- TRACE: thực hiện một bài test loop – back theo đường dẫn đến resource.

Trong số các phương thức này, ngoại trừ phương thức CONNECT, mà HTTP/1.1 dành cho việc tạo ra các giao thức dựa trên TCP như TLS, HTTP xác định ngữ nghĩa của mỗi phương thức.

HTTP là một giao thức giữa các khách hàng và nguồn tài nguyên. Trong giao thức này, trừ khi bạn định nghĩa các phương thức mới để mở rộng HTTP, danh sách các phương thức và ngữ nghĩa của chúng là cố định. Các ngữ nghĩa đó độc lập với các nguồn tài nguyên. Đó là lý do tại sao HTTP được gọi là một giao diện đồng nhất(Uniform Interface). Điều này khác biệt so với các cuộc gọi thủ tục từ xa (RPC) hoặc các dịch vụ web dựa trên SOAP, nơi ngữ nghĩa của mỗi yêu cầu là cụ thể cho ứng dụng.

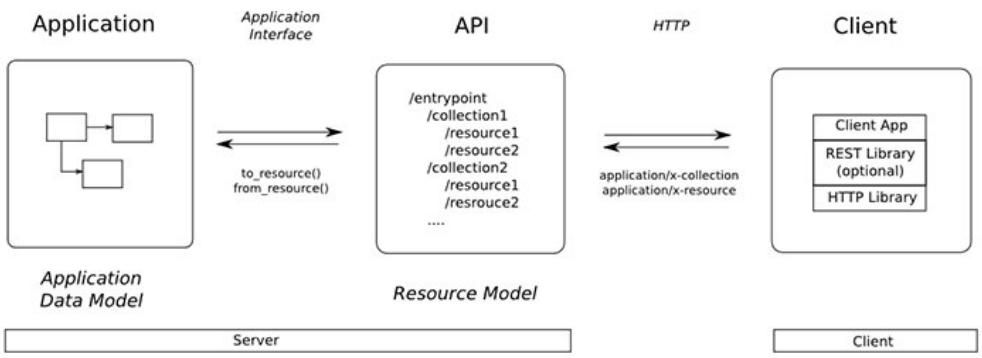
Hypermedia and Application State

Cuối cùng, mỗi biểu diễn mà khách hàng nhận từ máy chủ đều đại diện cho trạng thái tương tác của người dùng trong ứng dụng. Ví dụ, khi người dùng gửi biểu mẫu để nhận trang web khác, người dùng thay đổi trạng thái của ứng dụng từ quan điểm cá nhân của họ. Khi một người dùng chỉ duyệt một trang web, người dùng thay đổi trạng thái của ứng dụng với mỗi lần nhấp vào một liên kết để tải trang web khác.

Trong ví dụ này, để thay đổi trạng thái của ứng dụng, người dùng dựa vào các biểu mẫu và liên kết được tìm thấy trong HTML. HTML là một định dạng siêu văn bản, cho phép điều khiển liên kết và biểu mẫu để bạn dễ dàng di chuyển qua ứng dụng và từ đó thay đổi trạng thái của ứng dụng.

Cách sử dụng siêu văn bản của biểu diễn (như HTML) để chỉ định và quản lý trạng thái của ứng dụng được gọi là siêu văn bản là động cơ của trạng thái ứng dụng, hoặc viết tắt là ràng buộc siêu văn bản.

2.5.2. Cách hoạt động của RESTful API



Hình 2.10: Mô hình cách hoạt động của RESTful API

Client gửi yêu cầu: Client (có thể là một ứng dụng web, thiết bị di động, hoặc bất kỳ máy tính nào) tạo yêu cầu gửi đến máy chủ thông qua HTTP, bao gồm các phương thức như GET, POST, PUT hoặc DELETE và các thông tin cần thiết như headers và body (nếu cần).

Máy chủ xử lý yêu cầu: Máy chủ nhận yêu cầu, xác định và xử lý yêu cầu dựa trên URI và phương thức được gửi từ client.

Trả về phản hồi: Máy chủ sau đó trả về phản hồi cho client, thông qua HTTP status code để chỉ ra kết quả của yêu cầu (ví dụ: 200 OK, 404 Not Found), cùng với dữ liệu (nếu có) được trả về trong định dạng đã được xác định trước.

Client xử lý phản hồi: Client nhận phản hồi từ máy chủ và xử lý dữ liệu nhận được để hiển thị hoặc thực hiện các hành động tiếp theo.

- **Ví dụ cụ thể** về cách một RESTful API hoạt động để lấy thông tin về người dùng từ hệ thống như sau:

Tài nguyên (Resource): Trong ví dụ này, sử dụng tài nguyên là "users". Mỗi người dùng sẽ có một định danh duy nhất, ví dụ: "/users/123".

Phương thức HTTP: Để lấy thông tin về người dùng, tại đây sử dụng phương thức **GET**.

Yêu cầu từ Client:

GET /users/123 **HTTP**/1.1
Host: www.example.com

Hình 2.11: Ví dụ về một yêu cầu từ Client

- GET là phương thức HTTP được sử dụng để yêu cầu dữ liệu từ địa chỉ "/users/123".
- Host chỉ định tên miền của máy chủ mà yêu cầu được gửi đến.

Xử lý yêu cầu từ Server:

- Máy chủ nhận yêu cầu và phân tích địa chỉ "/users/123".
- Máy chủ tìm thông tin về người dùng có ID là 123 trong cơ sở dữ liệu hoặc hệ thống lưu trữ.

Phản hồi từ Server:

HTTP/1.1 **200 OK**
Content-Type: application/json
{ “id”:123, “username”: “trungdat”, “email”: “dotrungdat.mda@gmail.com” **}**

Hình 2.12: Ví dụ về phản hồi của Server

- "200 OK" là mã trạng thái HTTP cho biết đã thành công.
- "Content-Type" xác định định dạng của dữ liệu trả về, trong trường hợp này là JSON.
- Dữ liệu về người dùng (ID, username, email) được trả về trong định dạng JSON.

Client xử lý phản hồi:

- Client nhận được phản hồi từ máy chủ.
- Nếu thành công, client có thể hiển thị thông tin về người dùng trong giao diện người dùng hoặc thực hiện các thao tác tiếp theo với dữ liệu này.

2.5.3. Ứng dụng phổ biến của RESTful API

RESTful API được sử dụng rộng rãi trong các ứng dụng web và di động để trao đổi dữ liệu giữa client và server. Dưới đây là một số ứng dụng phổ biến:

Ứng dụng di động: RESTful API được sử dụng để lấy và gửi dữ liệu giữa các ứng dụng di động và server. Các ứng dụng như Facebook, Twitter, Instagram, Airbnb, Uber, ... đều sử dụng RESTful API để trao đổi dữ liệu.

Các hệ thống IoT: RESTful API được sử dụng trong các hệ thống IoT để lấy và gửi dữ liệu giữa các thiết bị và server. Ví dụ như các hệ thống giám sát và điều khiển thông minh trong nhà, hệ thống giám sát năng lượng,...

Các hệ thống bán hàng: RESTful API được sử dụng trong các hệ thống bán hàng để truy xuất thông tin sản phẩm, quản lý đơn hàng và thanh toán trực tuyến.

2.6. Kết luận chương 2

Trong nội dung chương 2, đã phân tích rõ về kiến trúc phần mềm REST, những khái niệm quan trọng trong ngành công nghệ phần mềm. Từ đó khi mà thiết kế giao diện lập trình ứng dụng(API - Application Programming Interface) mà tuân theo những quy tắc mà kiến trúc REST định nghĩa thì sẽ có một kiểu kiến trúc hay một tiêu chuẩn cho việc thiết kế API là RESTful API. Tiếp theo đã làm sáng tỏ được các nguyên tắc, cách hoạt động và ứng dụng của RESTful API. Để hiểu rõ hơn về việc áp dụng RESTful API vào trong những ứng dụng thực tế thì sẽ thấy ở chương 3.

CHƯƠNG 3 - ỨNG DỤNG RESTFUL API CHO DỮ LIỆU IOT

3.1. Phần mềm quản lý kho lạnh

3.1.1. Giới thiệu về phần mềm

Phần mềm gồm 3 phần :

- Phía giao diện(front-end)
- Phía máy chủ(back-end)
- API để kết nối 2 phần trên

Phần mềm là một giải pháp quản lý kho lạnh hiệu quả. Từ những dữ liệu của các cảm biến thu thập được từ môi trường, phần mềm sẽ lưu trữ dữ liệu vào cơ sở dữ liệu, nếu nhiệt độ, độ ẩm, tốc độ gió vượt ngưỡng cho phép thì sẽ gửi thông báo về email cho khách hàng để khách hàng biết và xử lý kịp thời. Phần mềm cũng giúp xem lại thông tin kho lạnh theo ngày giúp cho việc báo cáo về dữ liệu trở lên dễ dàng hơn.

3.1.2. Luồng hoạt động của phần mềm

Tạo dữ liệu: Đầu tiên viết một API để tạo ra dữ liệu nhiệt độ, độ ẩm, tốc độ gió, các dữ liệu này thu được cách nhau một giờ. Mô phỏng một cảm biến cứ một giờ thì gửi nhiệt độ về hệ thống.

Xử lý dữ liệu: Sau khi đã có dữ liệu từ các cảm biến, tổng hợp các dữ liệu cảm biến theo giờ rồi lưu trữ vào cơ sở dữ liệu. Bước này sẽ viết các API để xử lý công việc trên.

Xem nội dung: Sau khi đã viết xong các API sẽ dùng để liên kết giữa người dùng và máy chủ. Từ đó những dữ liệu sẽ được hiển thị lên phía người dùng.

3.2. Các bước thực hiện

3.2.1. Xác định nhu cầu

Nhiệt độ kho lạnh là yếu tố quan trọng trong quản lý hàng hóa, đặc biệt là các sản phẩm như thực phẩm, dược phẩm hay các vật phẩm y tế. Một sự cố nhỏ về nhiệt độ có thể dẫn đến hậu quả lớn đối với chất lượng của sản phẩm và an toàn cho người tiêu dùng. Vì vậy, quản lý nhiệt độ kho lạnh đóng vai trò quan trọng trong việc đảm bảo

chất lượng sản phẩm và nâng cao uy tín của doanh nghiệp.

3.2.2. Phân tích yêu cầu

Giám sát nhiệt độ, độ ẩm, sức gió: Phần mềm cần cung cấp giao diện để giám sát thông tin hiện tại của kho lạnh.

Cảnh báo và thông báo: Có khả năng cảnh báo khi các thông tin kho lạnh vượt quá hoặc dưới ngưỡng an toàn. Thông báo sự cố hoặc thay đổi nhiệt độ đến người quản lý.

Ghi nhật ký và báo cáo: Ghi lại dữ liệu về kho lạnh theo thời gian để phân tích và tạo báo cáo. Báo cáo này có thể bao gồm thông tin về biến động nhiệt độ, độ ẩm , sức gió thời gian xảy ra, và các sự cố nhiệt độ.

Hỗ trợ kỹ thuật và cập nhật: Cung cấp hỗ trợ kỹ thuật liên tục và cập nhật phần mềm thường xuyên để nâng cao tính ổn định và các tính năng mới.

3.2.3. Thiết kế

Lựa chọn công nghệ phù hợp:

3.2.3.1. Cơ sở dữ liệu

SQL (Structured Query Language hay ngôn ngữ truy vấn có cấu trúc) là một loại ngôn ngữ máy tính phổ biến để tạo, sửa, và lấy dữ liệu từ một hệ quản trị cơ sở dữ liệu quan hệ. Ngôn ngữ này phát triển vượt xa so với mục đích ban đầu là để phục vụ các hệ quản trị cơ sở dữ liệu đối tượng-quan hệ.

Hình 3.1: Hệ quản trị cơ sở dữ liệu(MySQL)

Là mã nguồn mở phổ biến nhất thế giới và được các nhà phát triển rất ưa chuộng trong quá trình phát triển ứng dụng. Vì MySQL là hệ quản trị cơ sở dữ liệu tốc độ cao, ổn định và dễ sử dụng, có tính khả chuyển, hoạt động trên nhiều hệ điều hành cung cấp một hệ thống lớn các hàm tiện ích rất mạnh. Với tốc độ và tính bảo mật cao, MySQL rất thích hợp cho các ứng dụng có truy cập CSDL trên internet. Người dùng có thể tải về MySQL miễn phí từ trang chủ. MySQL có nhiều phiên bản cho các hệ điều hành khác nhau: phiên bản Win32 cho các hệ điều hành dòng Windows, Linux, Mac OS X, Unix, FreeBSD, NetBSD, Novell NetWare, SGI Irix, Solaris, SunOS,..

3.2.3.2. Ngôn ngữ lập trình và Framework

Ngôn ngữ Java

Java là một trong những ngôn ngữ lập trình hướng đối tượng. Nó được sử dụngtrong phát triển phần mềm, trang web, game hay ứng dụng trên các thiết bị di động. Java được khởi đầu bởi James Gosling và bạn đồng nghiệp ở Sun MicroSystem năm 1991. Ban đầu Java được tạo ra nhằm mục đích viết phần mềm cho các sản phẩm gia dụng, và có tên là Oak.

Java được phát hành năm 1994, đến năm 2010 được Oracle mua lại từ Sun MicroSystem. Java được tạo ra với tiêu chí "Viết (code) một lần, thực thi khắp nơi" (Write Once, Run Anywhere – WORA). Chương trình phần mềm viết bằng Java có thể chạy trên mọi nền tảng (platform) khác nhau thông qua một môi trường thực thi với điều kiện có môi trường thực thi thích hợp hỗ trợ nền tảng đó.

Spring Boot Framework

Là một dự án phát triển bởi JAV (ngôn ngữ java) trong hệ sinh thái Spring framework. Nó giúp cho các lập trình viên đơn giản hóa quá trình lập trình một ứng dụng với Spring.

Hình 3.2: Spring Boot Framework

Ở thời điểm hiện tại Spring Boot có lẽ đang là framework nổi tiếng và được sử dụng nhiều bậc nhất trong phát triển các ứng dụng sử dụng ngôn ngữ Java. Được xây dựng trên nền tảng Spring framework, nó có tất cả các tính năng của Spring cộng thêm những tiện ích mà nó mang lại như giảm thiểu các bước cấu hình phức tạp, nhúng server container (Tomcat, Jetty hoặc Undertow) tự động vào ứng dụng giúp khởi chạy một ứng dụng ngay lập tức, quản lý dependence thông minh,…

Tất cả điều này đã tạo nên một sức lôi cuốn vô cùng lớn đối với cộng đồng developer trên khắp thế giới, các công ty cũng đang dần chuyển sang dùng Spring Boot cho các dự án tiếp theo. Lựa chọn học Spring Boot ở thời điểm hiện tại là một quyết định đúng đắn vì thị trường việc làm của nó đang rất nhiều. Không chỉ vậy, hiện tại có nhiều công ty cũng đang dần chuyển từ Spring qua Spring Boot cho các dự án cũ và các dự án mới sẽ dùng hẳn Spring Boot.

Hibernate Framework

Là Java Framework giúp đơn giản hóa việc phát triển ứng dụng Java để tương tác với cơ sở dữ liệu. Nó là một công cụ ORM mã nguồn mở giúp lập trình viên có thể ánh xạ các đối tượng trong Java với hệ quản trị cơ sở dữ liệu. Hibernate triển khai các thông số kỹ thuật của JPA (API Persistence Java) để duy trì dữ liệu.

Hiểu ngắn gọn thì Hibernate sẽ là một lớp đứng trung gian giữa ứng dụng và database, và sẽ giao tiếp với Hibernate thay vì giao tiếp với database.

Để giao tiếp với Hibernate, sẽ tạo ra một Class đại diện cho một Table. Và mọi dữ liệu từ Table trong database sẽ được Hibernate ánh xạ vào Class đó cho lập trình viên.

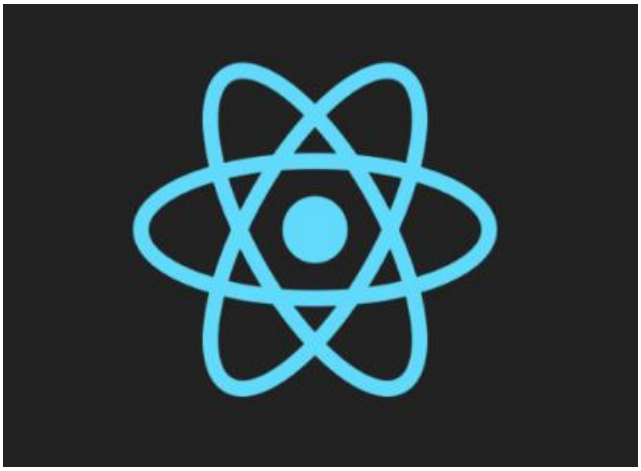
Ngôn ngữ JavaScript

Là ngôn ngữ lập trình được nhà phát triển sử dụng để tạo trang web tương tác. Từ làm mới bảng tin trên trang mạng xã hội đến hiển thị hình ảnh động và bản đồ tương tác, các chức năng của JavaScript có thể cải thiện trải nghiệm người dùng của trang web. Là ngôn ngữ kịch bản phía máy khách, JavaScript là một trong những công nghệ cốt lõi của World Wide Web. Ví dụ: khi duyệt internet, bất cứ khi nào bạn thấy quảng

cáo quay vòng dạng hình ảnh, menu thả xuống nhấp để hiển thị hoặc màu sắc phần tử thay đổi động trên trang web cũng chính là lúc bạn thấy các hiệu ứng của JavaScript.

ReactJS Framework

Là một opensource được phát triển bởi Facebook, ra mắt vào năm 2013, bản thân nó là một thư viện Javascript được dùng để để xây dựng các tương tác với các thành phần trên website. Một trong những điểm nổi bật nhất của ReactJS đó là việc render dữ liệu không chỉ thực hiện được trên tầng Server mà còn ở dưới Client nữa.



Hình 3.3: ReactJS Framework

Là một thư viện JavaScript chuyên giúp các nhà phát triển xây dựng giao diện người dùng hay UI. Trong lập trình ứng dụng front-end, lập trình viên thường sẽ phải làm việc chính trên 2 thành phần sau: UI và xử lý tương tác của người dùng. UI là tập hợp những thành phần mà bạn nhìn thấy được trên bất kỳ một ứng dụng nào, ví dụ có thể kể đến bao gồm: menu, thanh tìm kiếm, những nút nhấn, card,… Giả sử bạn đang lập trình một website thương mại điện tử, sau khi người dùng chọn được sản phẩm ưng ý rồi và nhấn vào nút "Thêm vào giỏ hàng", thì việc tiếp theo mà bạn phải làm đó là thêm sản phẩm được chọn vào giỏ hàng và hiển thị lại sản phẩm đó khi user vào xem => xử lý tương tác.

3.2.3.3. IDE

IntelliJ IDEA

Là một IDE dùng để lập trình rất nhiều các ngôn ngữ(đặc biệt là Java), sản phẩm

nổi tiếng của JetBrains đã nhận được rất nhiều giải thưởng. Phần mềm được thiết kế để cải tiến năng suất cho các nhà phát triển. IntelliJ IDEA cung cấp trình soạn thảo thông minh, trình phân tích mã và tập hợp mạnh mẽ của refactorings hỗ trợ một loạt các ngôn ngữ lập trình, các khuôn khổ và công nghệ, và đã sẵn sàng để sử dụng.

Visual Studio Code

Là trình soạn thảo, biên tập lập trình mã nguồn miễn phí được sử dụng trên 3 nền tảng đó là: Windows, macOS và Linux được xây dựng, phát triển bởi Microsoft. Visual Studio Code được các chuyên gia công nghệ thông tin đánh giá cao, nó là sự kết hợp hoàn hảo giữa IDE và CODE Editor.

Visual Studio Code còn có nhiệm vụ hỗ trợ các nền tảng như: JavaScript, TypeScript và Node.js. Bạn có thể hiểu cụ thể công việc của nó là mang đến một hệ sinh thái mới vô cùng phong phú cho các ngôn ngữ lập trình.

3.2.3.4. Môi trường kiểm tra API

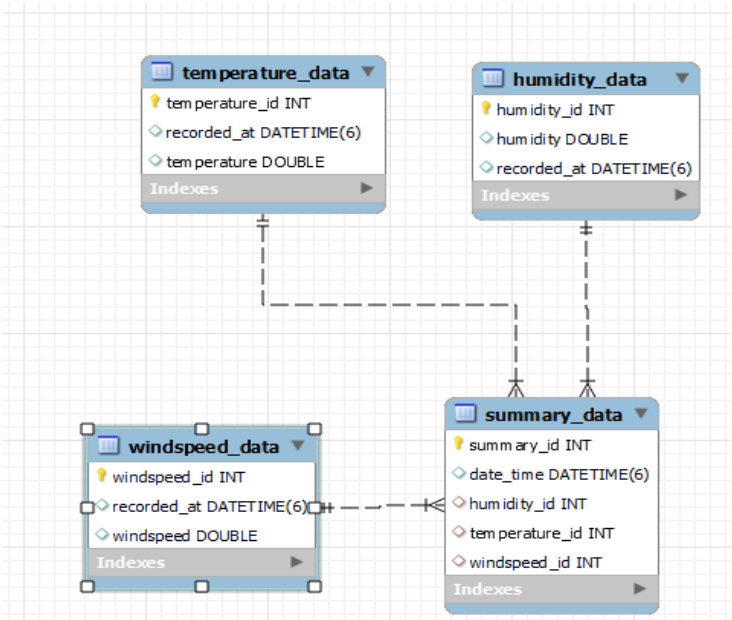
Postman là một ứng dụng cho phép người dùng có thể tương tác với API. API là tên viết tắt của Application Programming Interface – giao diện lập trình ứng dụng. API chịu trách nhiệm tạo ra các kết nối giữa những ứng dụng khác nhau. Còn Postman sẽ chịu trách nhiệm giúp cho thao tác của người dùng với API trở nên dễ dàng hơn. Thông thường, Postman sẽ được dùng cho API REST. Postman có thể dễ dàng gọi REST mà không cần tạo code như cách truyền thống.

Hình 3.4: Ứng dụng Postman

Postman còn chịu trách nhiệm cho rất nhiều thao tác phát triển phần mềm như test, chạy thử, rà lỗi, cung cấp các đoạn code tự động cần thiết… Nhìn chung, lợi ích mà Postman mang lại là rất nhiều.

Đặc biệt, Postman tương thích với hầu hết các giao thức HTTP, bao gồm: Get, Put, Delete, Post, Patch… Cấu tạo của Postman hướng đến sự linh động. Postman có sẵn nhiều phiên bản cho các hệ điều hành và môi trường khác nhau. Đó cùng là điểm khiến nó trở nên phổ biến hơn.

3.2.3.5. Thiết kế cơ sở dữ liệu



Hình 3.5: Biểu đồ mối quan hệ giữa các bảng(ER Diagram)

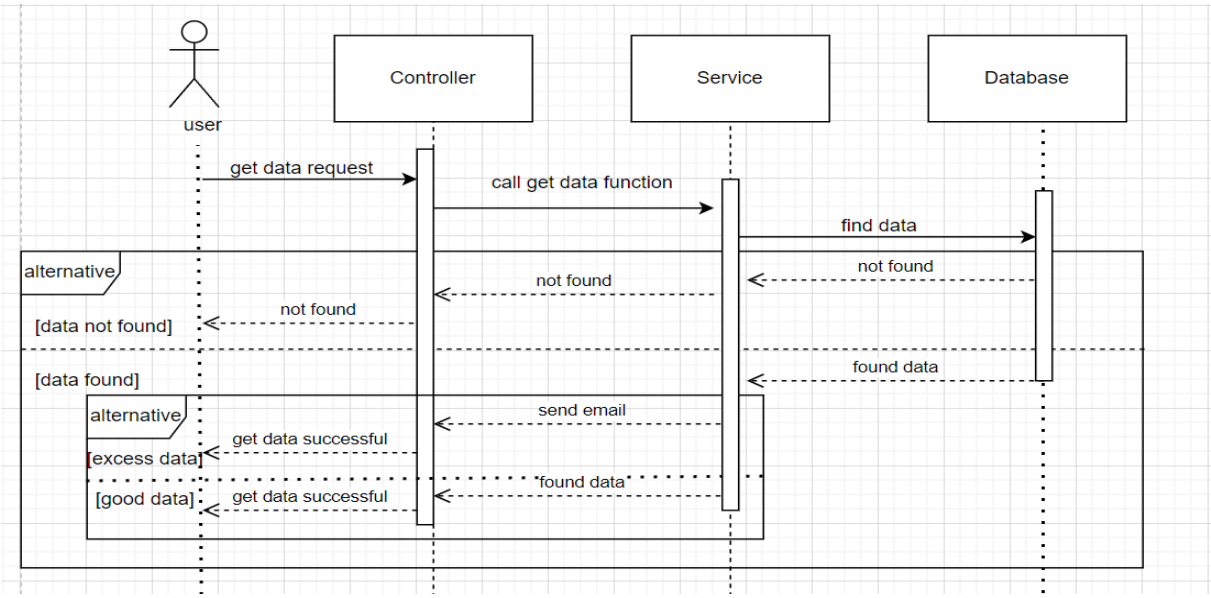
Cơ sở sẽ có 4 bảng gồm temperature_data, humidity_data, windspeed_data để lưu dữ liệu nhiệt độ, độ ẩm, tốc độ gió mà cảm biến thu được và bảng summary_data nơi tổng hợp dữ liệu của các cảm biến theo giờ. Các bảng điều có khóa chính và liên kết với nhau bằng khóa phụ.

3.2.3.6. Thiết kế giao diện

Giao diện sẽ gồm thông tin kho hàng hiện tại, một nút tìm kiếm và một bảng bảng để hiển thị thông tin kho hàng theo ngày tìm kiếm.

Hình 3.6: Minh họa giao diện của phần mềm

3.2.3.7. Luồng hoạt động của phần mềm



Hình 3.7: Luồng hoạt động của phần mềm

Yêu cầu lấy dữ liệu: Khi một yêu cầu được gửi đi, phía controller nhận được sẽ gọi đến service để xử lý yêu cầu. Trong hàm xử lý đó sẽ gọi đến database để tìm kiếm dữ liệu phù hợp.

Xem nội dung: nếu có dữ liệu phù hợp, thông tin sẽ được hiển thị trên giao diện người dùng. Nếu dữ liệu vượt quá giới hạn cho phép, người dùng sẽ nhận được email cảnh báo về tình trạng kho hàng. Trong trường hợp không có dữ liệu phù hợp, không có thông tin nào được hiển thị trên giao diện.

3.2.4. Lập trình

3.2.4.1. Tạo dự án

Sau khi đã cài đặt tất cả các công nghệ trên đầu tiên mở IntelliJ IDEA để tạo dự án Spring chọn File → New Project → Spring Initializr. Rồi đặt tên dự án chọn ngôn ngữ Java kiểu Maven rồi nhấn Next.

Tiếp theo thêm các thư viện cần thiết để kết nối với MySQL, ánh xạ với database và triển khai thành web. Rồi chọn create để tạo dự án Spring boot để xây dựng API phía máy chủ.

Tiếp tục mở Visual Studio Code chọn Terminal → New Terminal. Sau đó sẽ hiện ra 1 cửa sổ Terminal.

Sau đó gõ lệnh npx create-react-app để tạo dự án ReactJS. Và gõ lệnh npm install axios để gọi API.

3.2.4.2. Xây dựng API theo chuẩn REST

Đầu tiên vào file application.properties để kết nối IntelliJ với MySQL và cấu hình JavaMailSender:

Ở đây kết nối với database có tên là doan đã tạo trước đó ở MySQL và máy chủ của chúng hoạt động ở cổng 8088.

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/doan
spring.datasource.username=root
spring.datasource.password=trungdat35
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
server.port=8088
#spring.jpa.show-sql: true
logging.level.org.springframework.security=TRACE spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=dotrungdat.mda@gmail.com spring.mail.password=vhccxypsctltuhgv
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Hình 3.8: File application.properties

Tiếp theo tạo các thư mục như sau:

Tại thư mục model thêm bốn thực thể. Các thực thể này sẽ ánh xạ với database sẽ tạo ra các bảng và cột tương ứng.

HumidityData	TemperatureData

```java
@Entity
@Table(name = "humidity_data")
@Data
public class HumidityData {
    @Id
    @GeneratedValue(strategy =
            GenerationType.IDENTITY)
    @Column(name = "humidity_id")
    private int temperatureID;
    @Column(name = "humidity")
    private Double humidity;
    @Column(name = "recorded_at")
    private LocalDateTime recordedAt;
    @OneToMany(mappedBy = "humidity",
            cascade = CascadeType.ALL,
            fetch = FetchType.LAZY)
    @JsonIgnore
    private Set<SummaryData>summaryDataSet;
}
```

```java
@Entity
@Table(name = "temperature_data")
@Data
public class TemperatureData {
    @Id
    @GeneratedValue(strategy =
            GenerationType.IDENTITY)
    @Column(name = "temperature_id")
    private int temperatureID;
    @Column(name = "temperature")
    private Double temperature;
    @Column(name = "recorded_at")
    private LocalDateTime recordedAt;
    @OneToMany(mappedBy = "temperature",
            cascade = CascadeType.ALL,
            fetch = FetchType.LAZY)
    @JsonIgnore
    private Set<SummaryData>summaryDataSet;
}
```

Hình 3.9: Thực thể HumidityData và TemperatureData

WindSpeedData	SummaryData

```java
@Entity
@Table(name = "windspeed_data")
@Data
public class WindSpeedData {
    @Id
    @GeneratedValue(strategy =
            GenerationType.IDENTITY)
    @Column(name = "windspeed_id")
    private int temperatureID;
    @Column(name = "windspeed")
    private Double windspeed;
    @Column(name = "recorded_at")
    private LocalDateTime recordedAt;
    @OneToMany(mappedBy = "windspeed",
            cascade = CascadeType.ALL,
            fetch = FetchType.LAZY)
    @JsonIgnore
    private Set<SummaryData>summaryDataSet;
}
```

```java
@Entity
@Table(name = "summary_data")
@Data
public class SummaryData {
    @Id
    @GeneratedValue(strategy =
            GenerationType.IDENTITY)
    @Column(name = "summary_id")
    private int summaryID;
    @Column(name = "date_time")
    private LocalDateTime dateTime;
    @ManyToOne()
    @JoinColumn(name = "humidity_id",
            foreignKey = @ForeignKey(
                    name = "fk_sum_hum"))
    private HumidityData humidity;
    @ManyToOne()
    @JoinColumn(name = "temperature_id",
            foreignKey = @ForeignKey(
                    name = "fk_sum_temp"))
    private TemperatureData temperature;
    @ManyToOne()
    @JoinColumn(name = "windspeed_id",
            foreignKey = @ForeignKey(
                    name = "fk_sum_wind"))
    private WindSpeedData windspeed;
}
```

Hình 3.10: Thực thể WindSpeedData và SummaryData

Tiếp theo vào thư mục repository tạo các file:

WindSpeedDataRepository	TemperatureDataRepository

```java
@Repository
public interface WindSpeedDataRepository
        extends JpaRepository<
        WindSpeedData,Integer> {
    1 usage
    WindSpeedData findByRecordedAt(
            LocalDateTime dateTime);
}
```

```java
@Repository
public interface TemperatureDataRepository
        extends JpaRepository<
        TemperatureData,Integer> {
    1 usage
    TemperatureData findByRecordedAt(
            LocalDateTime dateTime);
}
```

Hình 3.11: Hai file (WindSpeedDataRepository và TemperatureDataRepository)

HumidityDataRepository	SummaryDataRepository

```java
@Repository
public interface HumidityDataRepository
        extends JpaRepository
        <HumidityData,Integer> {
    1 usage
    HumidityData findByRecordedAt
            (LocalDateTime dateTime);
}
```

```java
@Repository
public interface SummaryDataRepository
        extends JpaRepository<SummaryData,Integer> {
    1 usage
    SummaryData findFirstByOrderByDateTimeDesc();
    1 usage
    List<SummaryData> findByDateTimeBetween(
            LocalDateTime startOfDay,
            LocalDateTime endOfDay);
}
```

Hình 3.12: Hai file(HumidityDataRepository và TemperatureDataRepository)

Chức năng các file trên để thao tác như tìm kiếm, thêm, sửa, xóa và truy vấn dữ liệu ứng với từng bảng trong database .

Tiếp theo tại thư mục dto nơi chứa các trường muốn trả về cho client tại file: Response để về trạng thái và thông điệp của server. File ViewData chứa toàn bộ thông tin của kho hàng. Còn file EmailUtil chứa giao diện sẽ gửi về email của client.

Tiếp tục tại thư mục service tạo các file:

HumidityDataService, TemperatureDataService, WindSpeedDataService

Mỗi file này có một hàm để tạo dữ liệu để lưu vào database như là cảm biến thu được. Cuối cùng là file SummaryDataService chứa các hàm xử lý chính của phần mềm. File này gồm ba hàm xử lý :

Bảng 4: Cách hàm trong SummaryDataService

Hàm	Cách hoạt động và chức năng
Public void createSummaryData(String datetime) {…};	Hàm này sẽ được gọi mỗi khi thời gian chuyển sang giờ khác. Tham số truyền vào là tại thời điểm đó, sau đó hàm sẽ xử lý và lưu toàn bộ giữ liệu kho vào bảng summary_data
Public ViewData getSummaryDataNow() {…};	Sau khi hàm createSummaryData() được gọi, thì hàm này sẽ được gọi để lấy dữ liệu gần nhất trong bảng summary_data
Public List<ViewData> getSummaryDataByDate(String datetime) {…};	Hàm này để lọc ra dữ liệu mà người dùng tìm kiếm.

Cuối cùng tại thư mục controller tạo file SummaryDataController để xử lý yêu cầu từ người dùng và trả về một "view". View ở đây là dữ liệu hoặc trạng thái mà máy chủ trả về khi ta gọi API.

Dưới đây một số hàm trong SummaryDataController:

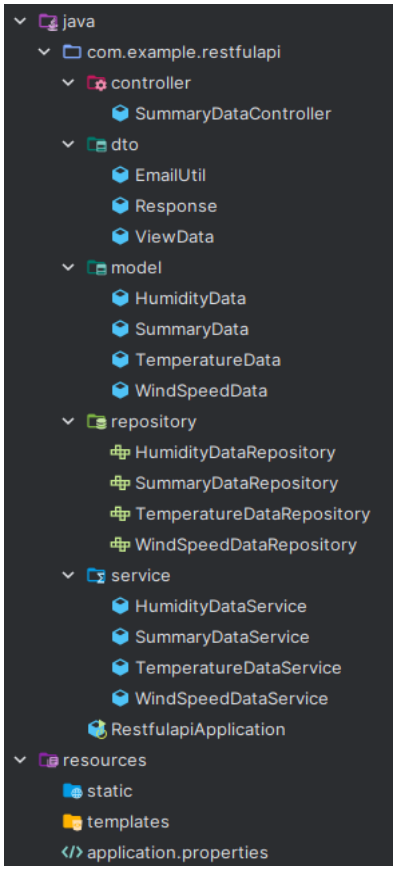
```
@CrossOrigin(origins = "http://localhost:3000/")
@RestController
@RequestMapping("/api/v1")
public class SummaryDataController {
    @Autowired
    private TemperatureDataService temperatureDataService;
    @Autowired
    private HumidityDataService humidityDataService;
    @Autowired
    private WindSpeedDataService windSpeedDataService;
    @Autowired
    private SummaryDataService summaryDataService;
    @PostMapping(value = "/createtemperaturedata", produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Response> insertTemperatureData(@RequestParam("year") int year,
                                                          @RequestParam("month") int month,
                                                          @RequestParam("day") int day) {
        temperatureDataService.createTemperatureData(year, month, day);
        return ResponseEntity.ok(new Response(HttpStatus.OK, message: "Thành công !"));
    }
    @GetMapping(value = "/getsummarydatabydate")
    public List<ViewData> getSummaryByDate(@RequestParam("date") LocalDate date) {
        return summaryDataService.getSummaryDataByDate(date);
    }
```

Hình 3.13: File SummaryDataController.class

@CrossOrigin(origins = "http://localhost:3000/"): đây là đường dẫn mà người dùng truy cập để xem thông tin của kho hàng. Còn khi đặt đường dẫn này tại controller thì đây chính là đường dẫn có thể truy cập toàn bộ tài nguyên của server có.

Khi một class được đánh dấu bằng **@RestController**, nó chỉ định rằng class này sẽ xử lý các yêu cầu HTTP và trả về kết quả dưới dạng dữ liệu RESTful thay vì trả về trang HTML hoặc các trang web được render, trong Spring Boot được sử dụng để xây dựng các RESTful API.

Dưới đây là tổ chức code của phần viết API:



Hình 3.14: Tổ chức code phía back-end

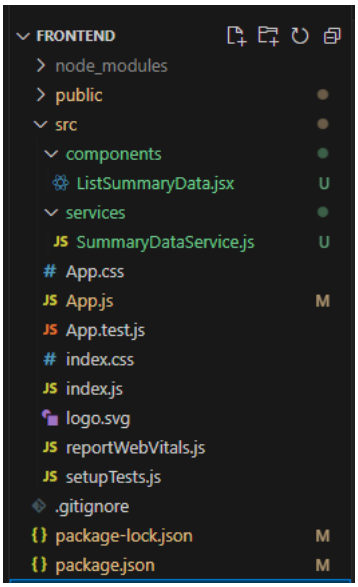
Cuối cùng sẽ thu được các API như sau:

Bảng 5: Các API được viết

STT	Đường dẫn(URL)
1	http://localhost:8088/api/v1/createwindspeeddata
2	http://localhost:8088/api/v1/createtemperaturedata
3	http://localhost:8088/api/v1/createsummarydata
4	http://localhost:8088/api/v1/createhumiditydata
5	http://localhost:8088/api/v1/getsummarydatanow
6	http://localhost:8088/api/v1/getsummarydatabydate

3.2.4.3 Xây dựng giao diện người dùng

Vừa rồi đã tạo xong project ReactJS, tiếp theo sẽ xây dựng giao diện tại đây. Tạo các thư mục components và services:



Hình 3.15: Tổ chức code phía front-end

Thư mục services chưa file TempService.js để kết nối với phía máy chủ(backend). Tại đây dùng thư viện axios để gọi đến API.

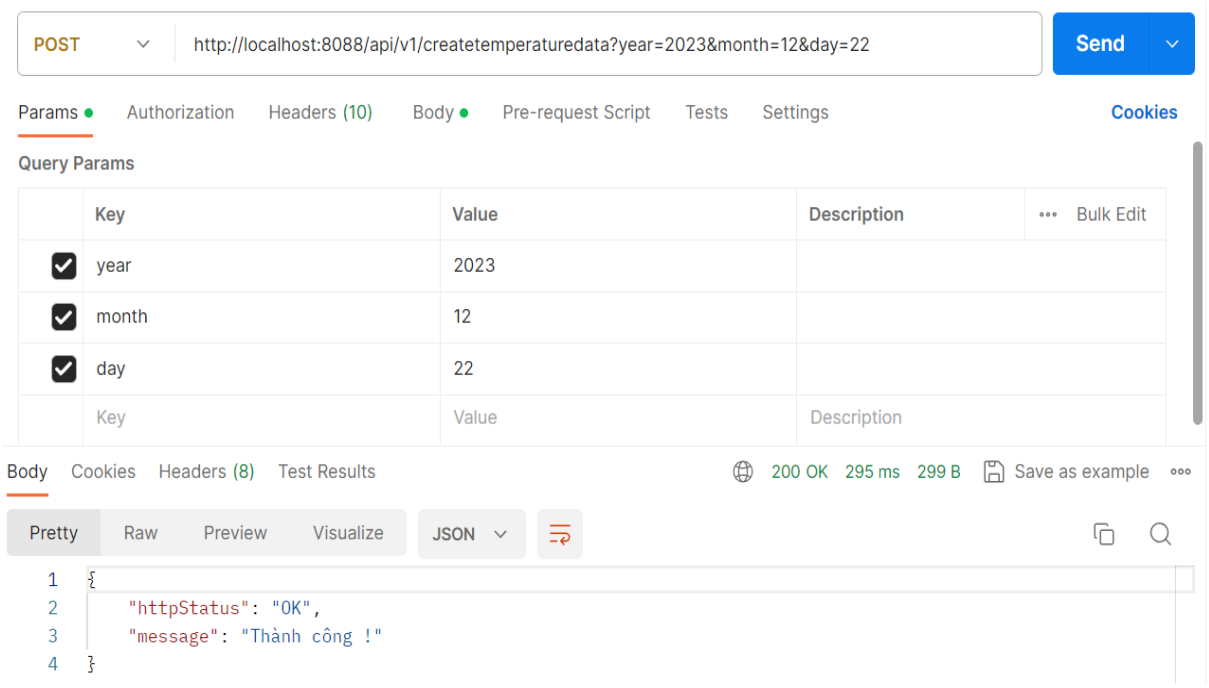
```
1  import axios from "axios";
2
3  const TEMP_API_BASE_URL = "http://localhost:8088/api/v1/"
4  class SummaryDataService {
5      getSummaryNow() {
6          debugger
7          return axios.get(`${TEMP_API_BASE_URL}getsummarydatanow`);
8      }
9
10     getSummaryDataByDate(date) {
11         debugger
12         return axios.get(`${TEMP_API_BASE_URL}getsummarydatabydate?date=${date}`);
13     }
14     saveSummaryData(date){
15         debugger
16         return axios.post(`${TEMP_API_BASE_URL}createsummarydata?datetime=${date}`);
17     }
18 }
19 export default new SummaryDataService()
```

Hình 3.16: File TempService.js

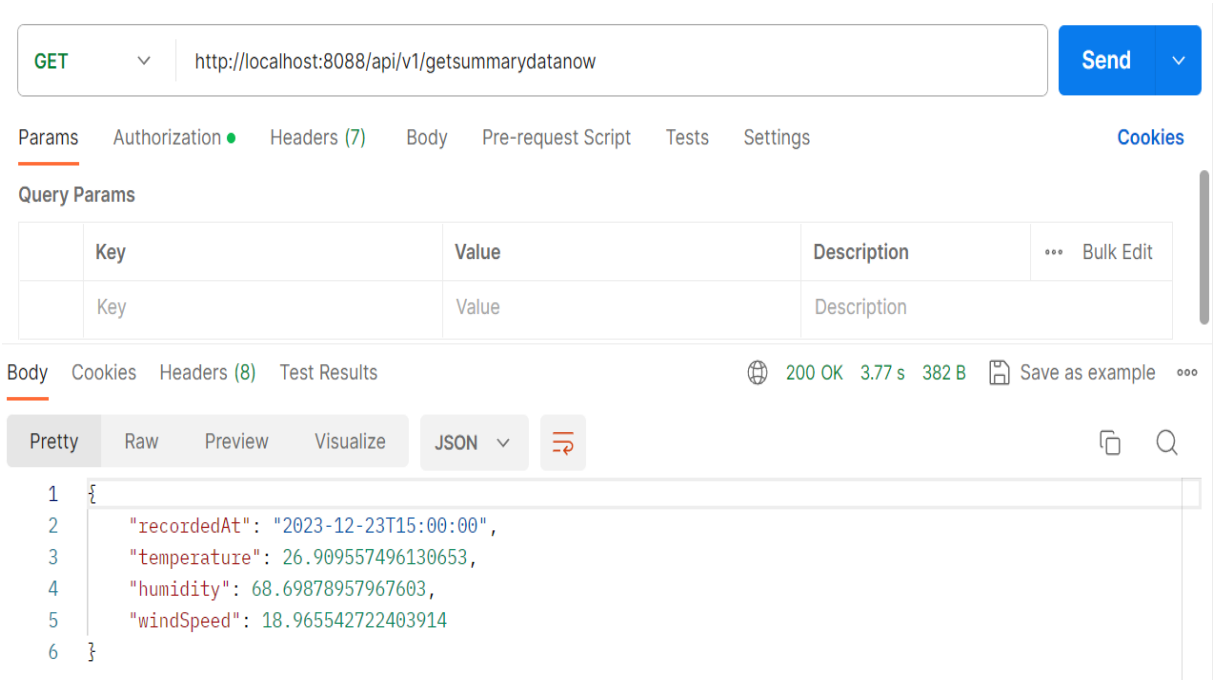
Tiếp theo đến thư mục components chứa file ListTemp.jsx chính là nơi viết logic và trang giao diện mà người dùng tương tác.Chạy project ReactJS hoạt động ở đường dẫn http://localhost:3000/.

3.2.5. Kiểm thử

Mở ứng dụng Postman để kiểm tra xem API :



Hình 3.17: Kết quả nhận khi gọi API tạo dữ liệu nhiệt độ ngày 22-12-2023



Hình 3.18: Kết quả khi gọi API lấy thông tin kho hiện tại

Kết quả thu được :

- **Định dạng dữ liệu trả về:** kết quả trả về được định dạng theo chuẩn JSON.
- **Sử dụng HTTP methods:** sử dụng các phương thức HTTP như GET để lấy tài nguyên, POST để tạo ra tài nguyên.

  • **Sử dụng URL và tài nguyên:**

  http://localhost:8088/api/v1/getsummarydatanow"

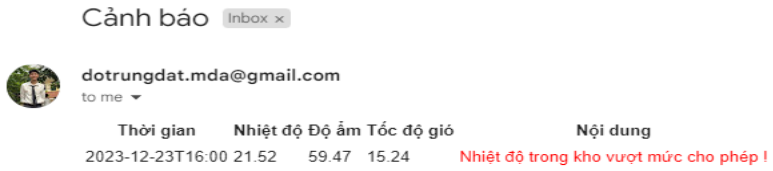
  localhost là một địa chỉ được sử dụng để tham chiếu đến máy tính đến chính máy tính đó. 8088 là cổng mà máy chủ đã cấu hình từ trước. "/api/v1" chính là đường dẫn cơ sở cho toàn bộ các phương thức trong SummaryDataController. Phần còn lại là đường dẫn và tham số truyền vào(nếu có).
- **Trạng thái và trả về HTTP Codes**: trả về HttpStatus là OK(200) và message là "thành công".

Như vậy có thể thấy API đã hoạt động đúng cách và đúng chuẩn REST như đã được đề ra. Và đây chính là ứng dụng của RESTful API cho dữ liệu Iot.

Tiếp theo truy cập đường dẫn http://localhost:3000/ để kiểm tra giao diện đã hiển thị đúng theo yêu cầu chưa:

Hình 3.19: Giao diện phần mềm

Giao diện, các thao tác đã hoạt động đúng theo yêu cầu.



Hình 3.20: Thông tin gửi về email

3.2.6. Triển khai và bảo trì

Tại giai đoạn này sản phẩm sẽ được bàn giao cho khách hàng, nếu có lỗi hoặc khách hàng có yêu cầu mới từ khách hàng. Phía nhà phát triển sẽ có những sửa đổi, và nâng cấp trong tương lai để khách hàng nhận được mức độ hài lòng cao nhất.

3.3. Kết luận chương 3

Trong chương 3, đã thấy được ứng dụng của RESTful API cho dữ liệu Iot. Phần mềm quản lý kho lạnh hoạt động dựa trên RESTful API - một giải pháp hiệu quả cho việc quản lý và theo dõi kho lạnh. Sử dụng kiến trúc RESTful API, em đã thiết kế và triển khai các endpoint để cung cấp thông tin về tình trạng kho hàng, dữ liệu cảm biến, và các chức năng quản lý khác. Việc sử dụng RESTful API cho phần mềm quản lý kho lạnh mang lại nhiều lợi ích. Nó cho phép giao tiếp linh hoạt giữa các hệ thống khác nhau, tăng cường khả năng mở rộng, và cung cấp dữ liệu theo yêu cầu. Điều này giúp cải thiện hiệu suất, tiết kiệm thời gian và tối ưu hóa quy trình quản lý kho lạnh.

TỔNG KẾT

Để đảm bảo hiệu suất và an toàn cho hệ thống quản lý kho lạnh, việc sử dụng RESTful API đóng vai trò quan trọng trong việc tối ưu hóa quy trình quản lý và giám sát. Xu hướng hiện nay đang dần chuyển hướng tích cực với sự phát triển của nhiều công nghệ mới, nâng cao đáng kể độ tin cậy và an toàn của hệ thống. Các nhà cung cấp phần mềm quản lý kho lạnh cần cân nhắc kỹ lưỡng khi triển khai hệ thống sử dụng RESTful API để đảm bảo tính linh hoạt, khả năng mở rộng và bảo mật. Việc áp dụng các giải pháp công nghệ tiên tiến sẽ giúp hệ thống vận hành một cách trơn tru, đồng thời mang lại trải nghiệm tốt nhất cho người dùng cuối. Điều này không chỉ tăng cường hiệu quả quản lý mà còn giúp giảm thiểu rủi ro về việc mất mát hàng hóa và các vấn đề liên quan đến an ninh dữ liệu.

Trong quá trình thực tìm hiểu đề tài thì em xin tóm tắt lại những kết quả đạt được và những khó khăn trong quá trình thực hiện đồ án "KIẾN TRÚC RESTFUL API VÀ ỨNG DỤNG TRONG HỆ THỐNG IOT" như sau:

CHƯƠNG 1: TỔNG QUAN VỀ PHÁT TRIỂN PHẦN MỀM:

- Nắm được khái niệm về phần mềm.
- Nắm được lịch sử, những giai đoạn và quy trình của phát triển phần mềm.
- Nắm được Design Pattern và phân tích được một Design Pattern nổi tiếng.

CHƯƠNG 2: KIẾN TRÚC REST VÀ RESTFUL API:

- Nêu ra được kiến trúc của REST, từ đó liên kết với RESTful API.
- Giới thiệu về các khái niệm quan trọng trong ngành công nghệ phần mềm và giới thiệu một số ứng dụng của RESTful API.

CHƯƠNG 3: ỨNG DỤNG CỦA RESTFUL API CHO DỮ LIỆU IOT

- Tìm hiểu chi tiết về cách phát triển phần mềm quản lạnh.
- Từ đó thấy được những lợi ích khi phát triển phần mềm theo đúng các giai đoạn và những ưu điểm khi dùng RESTful API trong phát triển phần mềm

TÀI LIỆU THAM KHẢO

Tiếng Anh

[1] Hans-Petter Halvorsen,"Software Development A Practical Approach", 2020.

[2] Roy Thomas Fielding, “Architectural Styles and the Design of Network-based Software Architectures”, 2000.

[3] Neal Ford, Mark Richards, “Software Architecture”, 2021.

[4] Mike Amundsen, “RESTful Web API Patterns and Practices Cookbook”, 2022.

Website

[1] https://vi.wikipedia.org/wiki/Alan_Turing

[2] https://itnavi.com.vn/blog/quy-trinh-phat-trien-phan-mem

[3] https://rabiloo.com/vi/blog/tong-quan-ve-mo-hinh-thac-nuoc-waterfall-model

[4] https://tigosoftware.com/vi/rad-model-mo-hinh-phat-trien-nhanh

[5] https://fmit.vn/tin-tuc/agile-la-gi

[6] https://vietnix.vn/tim-hieu-mo-hinh-mvc-la-gi/

[7] https://topdev.vn/blog/restful-api-la-gi/.

[8] https://devera.vn/blog/our-blog-1/post/cac-loai-apis-web-va-web-service-90

[9] https://blog.vinahost.vn/restful-api-la-gi/