

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA CÔNG NGHỆ PHẦN MỀM



KHOÁ LUẬN TỐT NGHIỆP

TÌM HIỂU NOSQL VÀ XÂY DỰNG
ỨNG DỤNG MINH HOẠ

Giảng viên hướng dẫn : **Th.S PHẠM THI VƯƠNG**
Sinh viên thực hiện : **ĐƯƠNG THÂN DÂN - 08520057**
BÙI NGỌC HUY - 08520544
Lớp : **CNPM03**
Khoá : **2008 - 2012**

TP. Hồ Chí Minh, tháng 03 năm 2013

LỜI MỞ ĐẦU

Ngày nay với kỷ nguyên công nghệ bùng nổ, thành công của Internet đã khiến cho số lượng người dùng truy cập vào cùng một hệ thống ngày càng tăng. Điển hình như Facebook một tháng phục vụ hơn 1000 tỉ truy cập và hơn 800 triệu lượt khách ghé thăm thì ta mới hình dung được sự bùng nổ của thông tin như thế nào. Để giải quyết vấn đề bùng nổ như trên thì chúng ta đã mở rộng các hệ thống máy chủ siêu lớn, phân thành nhiều cụm đặt khắp nơi trên thế giới. Nhưng với tốc độ phát triển theo cấp số như hiện nay thì việc tăng số lượng máy chủ thôi vẫn chưa đủ. Ta cần xem xét và nâng cấp các giải pháp lưu trữ cho tương lai.

Hệ thống máy chủ cơ sở dữ liệu đòi hỏi phải rất mạnh mẽ nếu không máy chủ sẽ bị quá tải. Với các hệ thống với số lượng lên đến hàng triệu cho đến hàng tỉ thì việc hiệu năng tốt là việc bắt buộc. Các hệ RDBMS hiện nay thì vấn đề hiệu năng thường không tốt cho trường hợp này. Ngôn ngữ SQL là ngôn ngữ thông dịch với các ràng buộc trong các bảng khiến cho hiệu năng thực sự của hệ thống cơ sở dữ liệu khi thực thi là khá ỉ ạch với hệ thống lớn như kể trên. Chưa kể là với hệ thống lớn thì vấn đề phân tán dữ liệu, tính toàn vẹn dữ liệu là việc rất quan trọng. NoSQL đáp ứng được tất cả các yêu cầu này. Với tốc độ nhanh do không phải qua các câu truy vấn SQL, có tính sẵn sàng, phân tán cao và độ ổn định tuyệt vời, NoSQL rất thích hợp cho các hệ thống có số lượng lượt truy vấn lớn. Ở trong khoá luận này, chúng tôi sẽ nghiên cứu về một loại NoSQL khá phổ biến – RavenDB.

RavenDB là một cơ sở dữ liệu mã nguồn mở có hỗ trợ transactional (giao dịch) được viết cho nền tảng .NET. RavenDB đưa ra mô hình dữ liệu linh hoạt (flexible data model) nhằm đáp ứng yêu cầu của các hệ thống thế giới thực (real-world systems). RavenDB cho phép xây dựng những ứng dụng có hiệu suất cao (high-performance), độ trễ thấp (low-latency) một cách nhanh chóng và hiệu quả. RavenDB xứng đáng là một cơ sở dữ liệu đáng tin cậy.

LỜI CẢM ƠN

Chúng em xin gửi lời cảm ơn sâu sắc đến thầy Phạm Thi Vương, đã giúp đỡ, tạo điều kiện cho nhóm hoàn thành tốt khóa luận tốt nghiệp này. Thầy đã tận tình hướng dẫn và đưa ra những nhận xét vô cùng quý giá để đề tài ngày càng hoàn thiện hơn. Những góp ý của thầy giúp cho chúng em tiếp cận, hiểu rõ và giải quyết vấn đề dễ dàng hơn.

Đồng thời, chúng em cũng xin bày tỏ lòng biết ơn đến quý thầy, cô Trường Đại Học Công Nghệ Thông Tin – Đại Học Quốc Gia Thành Phố Hồ Chí Minh, đặc biệt là các thầy, cô khoa Công nghệ Phần Mềm đã tận tình truyền đạt kiến thức, kinh nghiệm cho chúng em từ những ngày đầu học tập tại trường. Sự nhiệt tình của các thầy, cô đã giúp cho chúng em có kiến thức nền tảng vững chắc cũng như kinh nghiệm thực tiễn quý báu để chúng em có thể hoàn thành tốt các nhiệm vụ học tập, làm việc và nghiên cứu.

Bên cạnh đó, chúng em cũng gửi lời cảm ơn đến gia đình, các anh, chị, bạn bè đã động viên, giúp đỡ chúng em rất nhiều trong quá trình học tập cũng như trong cuộc sống.

Thành phố Hồ Chí Minh, ngày 09 tháng 03 năm 2013

Nhóm sinh viên thực hiện

Dương Thân Dân – Bùi Ngọc Huy

This image shows a full page of white paper with horizontal dotted lines. The lines are evenly spaced and run across the width of the page, providing a guide for handwriting practice. There are no margins, text, or other markings on the page.

[illegible]

MỤC LỤC

DANH MỤC CÁC BẢNG, SƠ ĐỒ.....	I
DANH MỤC CÁC HÌNH.....	III
CHƯƠNG 1 - Giới thiệu đề tài	1
1.1 Vấn đề tìm hiểu	1
1.2 Mục tiêu đề tài.....	2
1.3 Nội dung báo cáo	3
CHƯƠNG 2 - Tổng quan về cơ sở dữ liệu NoSQL	4
2.1 Tại sao chọn NoSQL?	4
2.2 NoSQL là gì ?	4
2.3 Ưu nhược điểm của cơ sở dữ liệu NoSQL:.....	6
2.3.1 Ưu điểm:.....	6
2.3.2 Nhược điểm:	8
2.4 Kiến trúc.....	9
2.5 Một số thuật ngữ liên quan	11
2.6 So sánh NoSQL với các loại cơ sở dữ liệu khác.....	13
2.6.1 So sánh NoSQL với XML.....	13
2.6.2 So sánh NoSQL với RDBMS.....	14
2.7 Cách triển khai một ứng dụng NoSQL	16
2.7.1 Xác định NoSQL có phù hợp.....	16
2.7.2 Thiết kế cấu trúc dữ liệu dạng document	17
CHƯƠNG 3 – Phân loại cơ sở dữ liệu NoSQL.....	22
3.1 Key-Value Store.....	22
3.2 Column Families / Wide Column Store.....	24
3.3 Document database	25

3.4	Graph Database	26
3.5	Làm sao để lựa chọn một giải pháp cơ sở dữ liệu tốt	28
3.6	Tìm hiểu một số loại NOSQL phổ biến.....	29
3.6.1	RavenDB	29
3.6.2	Hadoop	31
3.6.3	Cassandra	31
3.6.4	MongoDB.....	32
3.6.5	CouchDB.....	33
CHƯƠNG 4 – Tìm hiểu về RavenDB.....		34
4.1	Tại sao chọn RavenDB	34
4.2	Lý thuyết cơ bản RavenDB.....	35
4.2.1	RavenDB server	35
4.2.2	Documents, Collections và Document xác định duy nhất:	35
4.2.3	The Management Studio	36
4.3	.NET client API.....	37
4.3.1	Giới thiệu .NET client API	37
4.3.2	Nguyên tắc thiết kế .NET client API	37
4.3.3	Kết nối tới RavenDB data store	38
4.3.4	Những thao tác cơ bản với cơ sở dữ liệu	38
4.3.5	Sử dụng Index để truy vấn dữ liệu	40
4.4	Tổng quan HTTP API.....	41
4.5	Mở rộng hệ thống theo chiều ngang (scaling out)	42
4.5.1	Replication	42
4.5.2	Sharding	43
4.5.3	Kết hợp Replication và Sharding	46

4.6	So sánh hiệu suất RavenDB với MSSQL Express 2012.....	47
4.7	So sánh RavenDB với CouchDB và MongDB	48
CHƯƠNG 5-Xây dựng ứng dụng sử dụng RavenDB		50
5.1	Giới thiệu về ứng dụng	50
5.2	Lý do lựa chọn ứng dụng này	50
5.3	Đặc tả ứng dụng	51
5.3.1	Yêu cầu chức năng	51
5.3.2	Phân rã chức năng website	53
5.3.3	Yêu cầu phi chức năng	54
5.4	Ý tưởng thiết kế	56
5.4.1	Thiết kế mô hình 3 tầng (3 Tier): Clients – Web server – Database server.....	56
5.4.2	Kiến trúc Website.....	57
5.5	Phân tích, thiết kế hệ thống.....	58
5.5.1	Sơ đồ use case	58
5.5.2	Class diagram	72
5.5.3	Sequence diagram	75
5.6	Thiết kế giao diện.....	83
5.6.1	Danh sách màn hình	83
5.6.2	Mô tả giao diện người dùng	84
CHƯƠNG 6 – Kết luận		92
6.1	Kết quả đạt được	92
6.2	Hướng phát triển	93
Phụ lục		94
7.1	Tính năng đầy đủ của RavenDB	94
7.2	Quản lý mối quan hệ giữa các document.....	97

7.2.1	Phương pháp 1: Denormalization (Phi chuẩn hóa)	97
7.2.2	Phương pháp 2: Includes	99
7.2.3	Live Projections.....	101
7.2.4	Phương pháp kết hợp.....	101
7.2.5	Kết luận	102
7.3	Index.....	102
7.3.1	Static index	102
7.3.2	Stale index (index chứa kết quả cũ, chưa cập nhật)	105
7.4	Giới thiệu mô hình ASP.NET MVC4	106

TÀI LIỆU THAM KHẢO

DANH MỤC CÁC BẢNG, SƠ ĐỒ

Bảng 1.1: Hiệu suất hoạt động trên MySQL và Cassandra	2
Bảng 2.1: Bảng tương quan giữa RDBMS và NoSQL.....	6
Bảng 2.2: So sánh giữa NoSQL và cơ sở dữ liệu quan hệ	16
Bảng 3.1: So sánh 2 document	25
Bảng 4.1: Bảng so sánh RavenDB với MongoDB và CouchDB	48
Bảng 5.1: Bảng danh sách yêu cầu chức năng của ứng dụng.....	52
Bảng 5.2: Danh sách actor.....	62
Bảng 5.3: Danh sách use case.....	63
Bảng 5.4: Mô tả Usecase đăng nhập	63
Bảng 5.5: Mô tả Usecase đăng ký	64
Bảng 5.6: Mô tả Usecase xem Group public.....	65
Bảng 5.7: Mô tả Usecase tìm kiếm.....	65
Bảng 5.8: Mô tả Usecase xem topic	66
Bảng 5.9: Mô tả Usecase đăng bình luận	66
Bảng 5.10: Mô tả Usecase xóa bình luận	67
Bảng 5.11: Mô tả Usecase đăng topic	67
Bảng 5.12: Mô tả Usecase xóa Topic	68
Bảng 5.13: Mô tả Usecase chỉnh sửa thông tin cá nhân.....	68
Bảng 5.14: Mô tả Usecase tạo group.....	69
Bảng 5.15: Mô tả Usecase chỉnh sửa thông tin group.....	70
Bảng 5.16: Mô tả Usecase thêm member	70
Bảng 5.17: Mô tả Usecase tạo thay đổi quyền	71
Bảng 5.18: Danh sách màn hình.....	84
 Sơ đồ 5.1: Sơ đồ phân rã chức năng của Owner	 53
Sơ đồ 5.2: Sơ đồ phân rã chức năng của Manager	53
Sơ đồ 5.3: Sơ đồ phân rã chức năng của Member.....	54
Sơ đồ 5.4: Sơ đồ thiết kế mô hình 3 tầng: Clients – Web server – Database server	56
Sơ đồ 5.5: Sơ đồ kiến trúc website	57

Sơ đồ 5.6: Use case trong trường hợp chưa đăng nhập	58
Sơ đồ 5.7: Use case của Member	59
Sơ đồ 5.8: Use case của Manager	60
Sơ đồ 5.9: Use case của Owner	61
Sơ đồ 5.10: Sơ đồ lớp cung cấp các chức năng chính cho website	72
Sơ đồ 5.11: Sequence diagram của chức năng Login	75
Sơ đồ 5.12: Sequence diagram thực hiện chức năng Register	76
Sơ đồ 5.13: Sequence diagram của chức năng Create Group	77
Sơ đồ 5.14: Sequence diagram của trang Home	78
Sơ đồ 5.15: Sequence diagram của chức năng Join Group	79
Sơ đồ 5.16: Sequence diagram của chức năng Accept Member	79
Sơ đồ 5.17: Sequence diagram của chức năng Create Topic	80
Sơ đồ 5.18: Sequence diagram của chức năng đăng bình luận	81
Sơ đồ 5.19: Sequence diagram của chức năng tìm kiếm	82

DANH MỤC CÁC HÌNH

Hình 2.1: Ví dụ cơ bản về Key/ value	10
Hình 2.2: Sơ đồ thiết kế hệ thống database Master -Slave.....	10
Hình 2.3: So sánh cách thiết kế giữa NoSQL và RDBMS	11
Hình 2.4: Kết quả insert data	14
Hình 2.5: Kết quả truy vấn data.....	15
Hình 2.6: Ví dụ về blog đơn giản	18
Hình 2.7: Mô hình quan hệ trong RDBMS	19
Hình 2.8: Ví dụ về thiết kế dữ liệu chuẩn hoá và document của NoSQL	20
Hình 2.9: Ví dụ về thiết kế dữ liệu chuẩn hoá và document của NoSQL	21
Hình 3.1: Key-Value store.....	23
Hình 3.2: Column Families.....	24
Hình 3.3: Super Column.....	25
Hình 3.4: Graph database	26
Hình 3.5: Ví dụ về các nút trong một graph database	27
Hình 3.6: Kiến trúc client-server	30
Hình 4.1: RavenDB server.....	35
Hình 4.2: Document trong RavenDB	36
Hình 4.3: Management studio	36
Hình 4.4 : Kiến trúc hệ thống với .NET client API	37
Hình 4.5: Index	40
Hình 4.6 : Kiến trúc hệ thống với HTTP API.....	41
Hình 4.7 : Kiến trúc của Replication	42
Hình 4.8 : Kiến trúc của Sharding	43
Hình 4.9: Phân tán với các nút chuyển đổi dự phòng chuyên dụng.....	46
Hình 4.10: Phân tán với các nút chuyển đổi dự phòng nội bộ	46
Hình 4.11: Biểu đồ so sánh khả năng Insert dữ liệu của RavenDB so với MSSQL	47
Hình 4.12: Biểu đồ so sánh khả năng Select dữ liệu của RavenDB so với MSSQL ...	47
Hình 5.1: Màn hình chính của chương trình	84
Hình 5.2: Màn hình đăng nhập	85
Hình 5.3: Màn hình đăng ký tài khoản.....	85

Hình 5.4: Màn hình tạo mới nhóm	86
Hình 5.5: Màn hình tạo mới bài viết	86
Hình 5.6: Màn hình danh sách bài viết.....	87
Hình 5.7: Màn hình bài viết và tất cả bình luận	87
Hình 5.8: Màn hình upload file	88
Hình 5.9: Màn hình cài đặt Group.....	88
Hình 5.10: Màn hình quản lý user	89
Hình 5.11: Màn hình thêm thành viên nhóm.....	89
Hình 5.12: Màn hình tùy chỉnh cài đặt trong nhóm	90
Hình 5.13: Màn hình tìm kiếm	90
Hình 5.14: Màn hình thay đổi mật khẩu.....	91
Hình 5.15: Màn hình thay đổi thông tin cá nhân.....	91
Hình 7.1: Index replication to SQL	96
Hình 7.2: Geo-spatial search support	96
Hình 7.3: Multi-tenancy	96
Hình 7.4: Mẫu kiến trúc Model – View – Controller	107

CHƯƠNG 1 - GIỚI THIỆU ĐỀ TÀI

1.1 Vấn đề tìm hiểu

Trong khoảng hơn 2 thập niên trở lại đây, hệ quản trị cơ sở dữ liệu quan hệ - RDBMS là sự lựa chọn duy nhất cho việc quản trị cơ sở dữ liệu. Tuy nhiên, với các yêu cầu mới hiện nay thì RDBMS đã bộc lộ yếu điểm. Chính sự quá chặt chẽ, yêu cầu nhất quán dữ liệu đã gây ra sự rườm rà, phức tạp làm giảm hiệu suất hoạt động, nhất là trong trường hợp phải chứa một lượng lớn dữ liệu. Nhưng với sự bùng nổ công nghệ như hiện nay, nhất là với mạng Internet thì lượng dữ liệu cần lưu trữ ngày càng tăng. Yêu cầu cho việc lưu trữ ngày càng cao như: lưu trữ nhiều dữ liệu, tốc độ truy xuất nhanh, phân tán dữ liệu trên nhiều máy chủ... thì với mô hình cơ sở dữ liệu quan hệ như hiện nay thì rõ ràng chưa đáp ứng đủ các yêu cầu trên.

Mọi vấn đề đều có giải pháp. Thật vậy, những năm gần đây đã nổi lên một xu hướng lưu trữ mới có cách thức lưu trữ không dùng mô hình cơ sở dữ liệu quan hệ như đa số các cơ sở dữ liệu hiện tại, mà sử dụng mô hình cơ sở dữ liệu phi quan hệ - NoSQL. NoSQL sinh ra để khắc phục các vấn đề mà một cơ sở dữ liệu dạng RDBMS gặp phải. NoSQL sinh ra không phải để cạnh tranh với RDBMS mà là để đảm nhiệm những việc mà RDBMS chưa làm tốt.

Mục tiêu mà NoSQL nhắm đến đó là hiệu suất hoạt động cao với số lượng dữ liệu cực lớn. Tuy nhiên để đạt được điều đó thì NoSQL đã bỏ qua thông dịch trong SQL cùng với những truy vấn rườm rà. Việc sử dụng các ràng buộc quan hệ cùng truy vấn SQL có vẻ thân thiện và thích hợp với phân đông dữ liệu. Tuy nhiên, nếu dữ liệu quá đơn giản, các thủ tục SQL sẽ không cần thiết (theo Curt Monash - một nhà phân tích cơ sở dữ liệu, một blogger). Đồng thời NoSQL cũng có cách thiết kế dữ liệu khác với cơ sở dữ liệu truyền thống như: tư tưởng thiết kế dữ liệu phi quan hệ, lưu trữ dữ liệu dạng document, sử dụng tối đa indexes... Trong các đặc tính đó, dữ liệu phi quan hệ là một yếu tố quan trọng góp phần làm nên thành công cho NoSQL. Dữ liệu phi quan hệ tức là không tuân theo các dạng chuẩn hóa mà cơ sở dữ liệu RDBMS đặt ra. Thay vào đó, khi thiết kế một cơ sở dữ liệu NoSQL ta phải tuân theo một số quy tắc mới mà NoSQL đặt ra để đạt được hiệu suất hoạt động cao.

Bảng dưới đây chỉ ra kết quả làm việc trên MySQL và cơ sở dữ liệu Cassandra của Facebook.

Facebook Search > 50 GB Data		
	MySQL	Cassandra
Writes Average	~300ms	0.12ms
Reads Average	~350ms	15ms

Bảng 1.1: Hiệu suất hoạt động trên MySQL và Cassandra

Chính sự khác biệt giữa 2 loại cơ sở dữ liệu này dẫn đến cách thiết kế cũng khác nhau. Đa số các lập trình viên đều quen với mô hình quan hệ truyền thống, do đó cần phải tìm hiểu kỹ cách thiết kế dữ liệu của NoSQL để đạt được hiệu suất mong muốn. Đồng ý rằng RDBMS cung cấp một mô hình tuyệt vời để đảm bảo tính toàn vẹn dữ liệu. Tuy nhiên, rất nhiều người lựa chọn NoSQL đã nói rằng chúng không quá cần thiết cho nhu cầu của họ.

Như vậy, trong đề tài này chúng tôi sẽ tìm hiểu xem NoSQL đã giải quyết các vấn đề trên như thế nào và áp dụng kiến thức tìm hiểu đó vào việc xây dựng một ứng dụng sử dụng cơ sở dữ liệu dạng NoSQL.

1.2 Mục tiêu đề tài

Với những vấn đề nêu trên, đề tài này cần đạt được các mục tiêu như sau:

Tìm hiểu NoSQL, kiến trúc, phân loại và đặc điểm từng loại để có cái nhìn tổng quan về NoSQL đồng thời biết được cách mà NoSQL đã giải quyết được vấn đề hiệu suất cao với lượng dữ liệu lớn như thế nào.

Tìm hiểu trường hợp áp dụng cơ sở dữ liệu dạng NoSQL, trường hợp nào không phù hợp với NoSQL. Phân tích làm rõ ưu khuyết điểm của việc áp dụng cơ sở dữ liệu NoSQL. So sánh giữa việc sử dụng cơ sở dữ liệu RDBMS hoặc XML và cơ sở dữ liệu NoSQL trên cùng một ứng dụng. So sánh hiệu suất giữa một cơ sở dữ liệu dạng NoSQL và cơ sở dữ liệu dạng RDBMS để làm rõ hiệu suất hoạt động của NoSQL.

Tìm hiểu tổng quan các cơ sở dữ liệu NoSQL phổ biến như: RavenDB, Hadoop, Cassandra, MongoDB, CouchDB.

Do có bốn loại cơ sở dữ liệu NoSQL (xem chi tiết tại chương 3: Phân loại cơ sở dữ liệu NoSQL) nên chúng em tập trung tìm hiểu cách thiết kế dữ liệu cho cơ sở dữ liệu loại Document database là loại phổ biến nhất. Sau đó tìm hiểu chi tiết về kỹ thuật của một cơ sở dữ liệu thuộc loại này là RavenDB.

Sử dụng các kiến thức về RavenDB để xây dựng một ứng dụng sử dụng cơ sở dữ liệu NoSQL đồng thời để tổng hợp lại kiến thức đã học trước đây. Ở đây chúng tôi quyết định xây dựng một website cho phép các người dùng có thể thảo luận về vấn đề nào đó (với các chức năng cơ bản như Google Group) bởi vì ứng dụng có các tính chất phù hợp với cơ sở dữ liệu dạng NoSQL.

1.3 Nội dung báo cáo

Nội dung đề tài được tổ chức thành 6 chương:

Chương 1 – Giới thiệu đề tài: Trong chương này sẽ trình bày về vấn đề cần tìm hiểu trong luận văn này, mục tiêu cần đạt được của luận văn.

Chương 2 – Tổng quan về cơ sở dữ liệu NoSQL: Nội dung chương này sẽ trình bày kiến thức tổng quan về NoSQL, phân tích ưu nhược điểm của cơ sở dữ liệu NoSQL.

Chương 3 – Phân loại cơ sở dữ liệu NoSQL: Nội dung chương này mô tả 4 loại NoSQL chính. Với mỗi loại sẽ giới thiệu khái quát và trường hợp áp dụng.

Chương 4 – Tìm hiểu về RavenDB: Chương này chúng em sẽ tìm hiểu kỹ về kỹ thuật, cách áp dụng của một cơ sở dữ liệu thuộc loại document database đó là RavenDB.

Chương 5 – Xây dựng ứng dụng sử dụng RavenDB: Sử dụng kết quả tìm hiểu của các chương trên để áp dụng vào xây dựng một ứng dụng sử dụng RavenDB làm cơ sở dữ liệu.

Chương 6 – Kết luận: Chương cuối này, chúng em ghi nhận lại kết quả đạt được cũng như hạn chế của báo cáo và chương trình. Ngoài ra, chúng em cũng trình bày định hướng phát triển tiếp theo của ứng dụng web này.

CHƯƠNG 2 - TỔNG QUAN VỀ CƠ SỞ DỮ LIỆU NOSQL

2.1 Tại sao chọn NoSQL?

Cơ sở dữ liệu quan hệ được thiết kế cho những mô hình dữ liệu không quá lớn trong khi các dịch vụ mạng xã hội lại có một lượng lớn dữ liệu và cập nhật liên tục do số lượng người dùng quá nhiều. Do đó cơ sở dữ liệu NoSQL sinh ra với mục tiêu giải quyết các thiếu sót của RDBMS trong các hệ thống phần mềm hiện đại. NoSQL sẽ tập trung giải quyết các vấn đề như tốc độ thực thi, khả năng lưu trữ, các nghiệp vụ phức tạp (phân trang, đánh chỉ mục ...). Nhờ vậy giải pháp sử dụng cơ sở dữ liệu NoSQL sẽ hạ thấp chi phí nếu so sánh với RDBMS truyền thống.

NoSQL vừa mang lại một giải pháp tốt hơn vừa tiết kiệm chi phí hơn do NoSQL có hiệu suất làm việc tốt hơn và các cơ sở dữ liệu NoSQL thường là miễn phí. Ngoại trừ một số trường hợp đặc biệt, với cùng một chi phí thì giải pháp sử dụng NoSQL sẽ mang lại lợi ích to lớn. Hãy tưởng tượng, với một hệ thống cho bạn đầy đủ quyền kiểm soát (mã nguồn mở), đáp ứng được tốc độ thực thi, khả năng lưu trữ, phân tán dữ liệu... và nhất là chi phí sẽ thấp hơn thì NoSQL chính là sự lựa chọn tuyệt vời.

Mặc khác, thường chúng ta sử dụng rất hạn chế những khả năng mà các cơ sở dữ liệu RDBMS cung cấp nhưng vẫn phải trả phí cho nó. Nếu không cần đến các tính năng cao cấp, không cần các chức năng của SQL hoặc rất ghét viết các câu lệnh SQL thì hãy nghĩ đến NoSQL.

2.2 NoSQL là gì ?

NoSQL là một xu hướng cơ sở dữ liệu mà không dùng mô hình dữ liệu quan hệ để quản lý dữ liệu trong lĩnh vực phần mềm. NoSQL có nghĩa là Non-Relational (NoRel) - không ràng buộc. Tuy nhiên, thuật ngữ đó ít phổ biến hơn và ngày nay người ta thường dịch NoSQL thành Not Only SQL - Không chỉ SQL.

NoSQL được xem như thế hệ database kế tiếp của RDBMS, là một thế hệ cơ sở dữ liệu non-relational (không ràng buộc), distributed (phân tán), open source, horizontal scalable (khả năng mở rộng theo chiều ngang) có thể lưu trữ, xử lý từ một lượng rất nhỏ cho tới hàng petabytes dữ liệu trong hệ thống có độ chịu tải, lỗi cao với

những đòi hỏi về tài nguyên phần cứng thấp. Để hiểu thêm về các khái niệm này trong NoSQL, có thể xem chi tiết ở phần 2.5 Một số thuật ngữ liên quan.

Một số đặc điểm nhận dạng cho thể hệ database mới này bao gồm:

- Lược đồ tự do (Schema-free).
- Hỗ trợ mở rộng dễ dàng.
- API đơn giản.
- Eventual consistency (nhất quán cuối) và transactions hạn chế trên các thành phần dữ liệu đơn lẻ.
- Không giới hạn không gian dữ liệu...

NoSQL storage đặc biệt phổ dụng trong thời kỳ Web 2.0 bùng nổ, nơi các mạng dịch vụ dữ liệu cộng đồng cho phép người dùng tạo hàng tỷ nội dung trên web. Do đó, dữ liệu lớn rất nhanh vượt qua giới hạn phần cứng và cần phải giải quyết bằng bài toán phân tán. Nửa đầu năm 2009, người ta đã manh nha thuật ngữ NoSQL đánh dấu sự trưởng thành của thể hệ database mới: distributed (phân tán) + non-relational (không ràng buộc).

Khi làm việc với NoSQL ta sẽ gặp một số khái niệm sau:

- **Fields:** tương đương với khái niệm Columns trong SQL
- **Document:** thay thế khái niệm row trong SQL. Đây cũng chính là khái niệm làm nên sự khác biệt giữa NoSQL và SQL, 1 document chứa số cột (fields) không cố định trong khi 1 row thì số cột(columns) là định sẵn trước.
- **Collection:** tương đương với khái niệm table trong SQL. Một collection là tập hợp các document. Điều đặc biệt là một collection có thể chứa các document hoàn toàn khác nhau.
- **Key-value:** cặp khóa - giá trị được dùng để lưu trữ dữ liệu trong NoSQL
- **Cursor:** tạm dịch là con trỏ. Sử dụng cursor để lấy dữ liệu từ database.

Trong các hệ cơ sở dữ liệu quan hệ, các cột được định nghĩa theo bảng còn với hệ cơ sở dữ liệu không ràng buộc, các cột được định nghĩa ở mỗi document. Bởi thế, các document quản lý gần như tất cả, các collection không cần quản lý chặt chẽ những gì đang xảy ra trong nó nữa.

RDBMS	NoSQL
Columns	Fields
Row	Document
Table	Collection
Query: SQL	Query: using API
Foreign keys	Non Foreign keys
Schema	Free schema

Bảng 2.1: Bảng tương quan giữa RDBMS và NoSQL

2.3 Ưu nhược điểm của cơ sở dữ liệu NoSQL:

2.3.1 Ưu điểm:

Hiệu suất hoạt động cao: NoSQL có hiệu suất hoạt động cao, lưu trữ lượng lớn dữ liệu để đáp ứng nhu cầu lưu trữ ngày càng tăng hiện nay. Tuy nhiên để đạt được điều này cần loại bỏ đi một số thứ như: ràng buộc dữ liệu của mô hình quan hệ, tính nhất quán dữ liệu, ngôn ngữ truy vấn SQL. Đồng thời NoSQL có một số cải tiến mới như sử dụng tốt index, khả năng phân tán dễ dàng đã giúp NoSQL có một hiệu suất hoạt động rất cao.

Khả năng phân trang: phân trang trong cơ sở dữ liệu quan hệ khá khó khăn khi không có một phương pháp chính thống nào để phục vụ cho việc này. Người lập trình phải dùng các phương pháp khác nhau để có thể lấy đúng số item cần lấy. Trong khi NoSQL hỗ trợ rất tốt việc này đồng thời hiệu suất khi phân trang không hề giảm.

NoSQL là nguồn mở: Các sản phẩm nguồn mở đưa ra cho những người phát triển với nhiều lợi ích to lớn, trong đó việc sử dụng miễn phí là một lợi ích lớn. Những lợi ích khác: phần mềm nguồn mở có xu hướng sẽ là tin cậy hơn, an ninh hơn và nhanh hơn để triển khai so với các lựa chọn thay thế sở hữu độc quyền. Ví dụ như các hệ quản trị cơ sở dữ liệu (CSDL) NoSQL: Cassandra, CouchDB, Hbase, RavenDB, MongoDB và Redis.

Việc mở rộng phạm vi là mềm dẻo: NoSQL thay thế câu thần chú cũ của các nhà quản trị CSDL về 'mở rộng phạm vi' với một thứ mới: 'mở rộng ra ngoài'. Thay vì bổ sung thêm các máy chủ lớn hơn để điều khiển nhiều tải dữ liệu hơn, thì CSDL NoSQL cho phép một công ty phân tán tải qua nhiều máy chủ khi mà tải gia tăng.

Các CSDL NoSQL khác nhau cho những dự án khác nhau:

- MongoDB và Redis là những lựa chọn tốt cho việc lưu trữ các dữ liệu thống kê ít được đọc mà lại được viết thường xuyên, như một số đếm truy cập web chẳng hạn.
- Hadoop, một CSDL dạng tự do, phân tán làm tốt công việc lưu trữ các dữ liệu lớn như các con số thống kê thời tiết hoặc công việc phân tích nghiệp vụ.
- Memcache, một CSDL nhất thời chóng tàn, tuyệt vời trong lưu trữ các phiên làm việc web, các khóa, và các con số thống kê ngắn hạn.
- Cassandra và Riak (các lưu trữ dư thừa, tự động tạo bó cluster) làm tốt trong các môi trường với các ứng dụng có tính sẵn sàng cao, khi thời gian sống tối đa là sống còn.

NoSQL được các hãng lớn sử dụng: Các công ty như Amazon, BBC, Facebook và Google dựa vào các CSDL NoSQL.

NoSQL phù hợp với công nghệ đám mây: NoSQL và đám mây là một sự trùng khớp tự nhiên. Các máy chủ ngày nay là không đắt và có thể dễ dàng mở rộng phạm vi được theo yêu cầu có sử dụng một dịch vụ như là Amazon EC2. Giống như tất cả công nghệ đám mây, EC2 dựa vào ảo hóa. Liên kết yếu của ảo hóa là sự thực thi của I/O, với bộ nhớ và CPU các các kết nối mạnh.

Các CSDL NoSQL hầu hết sử dụng bộ nhớ qua đĩa như là vị trí ghi đầu tiên - vì thế ngăn ngừa được sự thực thi không ổn định của I/O. Và vì NoSQL lưu trữ dữ liệu thường thúc đẩy được tính mở rộng phạm vi theo chiều ngang thông qua việc ngăn chia, chúng có khả năng tận dụng được việc cung cấp mềm dẻo của đám mây.

2.3.2 Nhược điểm:

Cấu trúc dữ liệu phi quan hệ: với cấu trúc dữ liệu phi quan hệ đã giúp NoSQL giảm đi rất nhiều tính toán không cần thiết. Điều này dẫn đến dữ liệu sẽ không ràng buộc chặt chẽ và ảnh hưởng tính nhất quán dữ liệu. Như vậy với các ứng dụng yêu cầu dữ liệu phải chặt chẽ như ứng dụng về tài chính, ngân hàng với các con số phải rất chính xác thì NoSQL không phải một sự lựa chọn tốt.

Nguồn mở có thể có nghĩa là sự hỗ trợ không đồng đều cho các doanh nghiệp:

- Trong khi các nhà cung cấp chủ chốt của RDBMs như Oracle, IBM hay Sybase đưa ra sự hỗ trợ tốt nổi tiếng cho các khách hàng doanh nghiệp cỡ vừa, thì các doanh nghiệp nhỏ hơn, thường là các nhà cung cấp nguồn mở mới thành lập không thể mong đợi được cung cấp sự hỗ trợ có thể so sánh được (ngoại trừ một nhóm các khách hàng blue chip).
- Nhà cung cấp nguồn mở trung bình thiếu sự tiếp cận toàn cầu, các dịch vụ hỗ trợ và sự tin cậy của Oracle hay IBM.

Chưa đủ “chín” cho các doanh nghiệp: Dù chúng đã được triển khai tại một số công ty lớn thì các CSDL NoSQL vẫn đối mặt với một vấn đề về sự tin cậy chính với nhiều doanh nghiệp. Điểm sống còn của NoSQL là thiếu về độ “chín” muối và các vấn đề về tính không ổn định, trong khi đó tính chín muối, hỗ trợ đầy đủ chức năng và tính ổn định của các RDBMS được thiết lập đã từ lâu.

Những hạn chế về tri thức nghiệp vụ: Có một vài câu hỏi xung quanh những khả năng về tri thức nghiệp vụ (BI) của các CSDL NoSQL. Liệu các CSDL này có thể cung cấp dạng phân tích dữ liệu lớn và mạnh mà các doanh nghiệp đã quen với các RDBMS? Cần bao nhiêu sự tinh thông về lập trình cần có để tiến hành những truy vấn và phân tích hiện đại?

- Các câu trả lời là không tích cực. Các CSDL NoSQL không có nhiều sự đeo bám tới các công cụ BI thường được sử dụng, trong khi những yêu cầu và phân tích hiện đại đơn giản nhất thì cũng liên quan khác nhiều

tới sự tinh thông về lập trình. Tuy vậy, các giải pháp là sẵn sàng. Quest Software, ví dụ, đã tạo ra Toad cho các CSDL đám mây, mà nó phân phối các khả năng truy vấn hiện đại tới một số CSDL NoSQL.

Thiếu sự tinh thông: Tính rất mới mẻ của NoSQL có nghĩa là không có nhiều lập trình viên và người quản trị mà biết công nghệ này - là khó khăn cho các công ty tìm người với sự tinh thông phù hợp. Đối lại, thế giới của RDBMS có hàng ngàn những người đủ tư cách.

Những vấn đề về tính tương thích: Không giống như các CSDL quan hệ, các CSDL NoSQL chia sẻ ít theo cách thức của các tiêu chuẩn. Mỗi CSDL NoSQL có các giao diện lập trình ứng dụng API riêng của mình, các giao diện truy vấn độc nhất vô nhị, và những sự riêng biệt. Sự thiếu hụt các tiêu chuẩn có nghĩa là nó không có khả năng để chuyển một cách đơn giản từ một nhà cung cấp này sang một nhà cung cấp khác nếu bạn không hài lòng với dịch vụ.

2.4 Kiến trúc

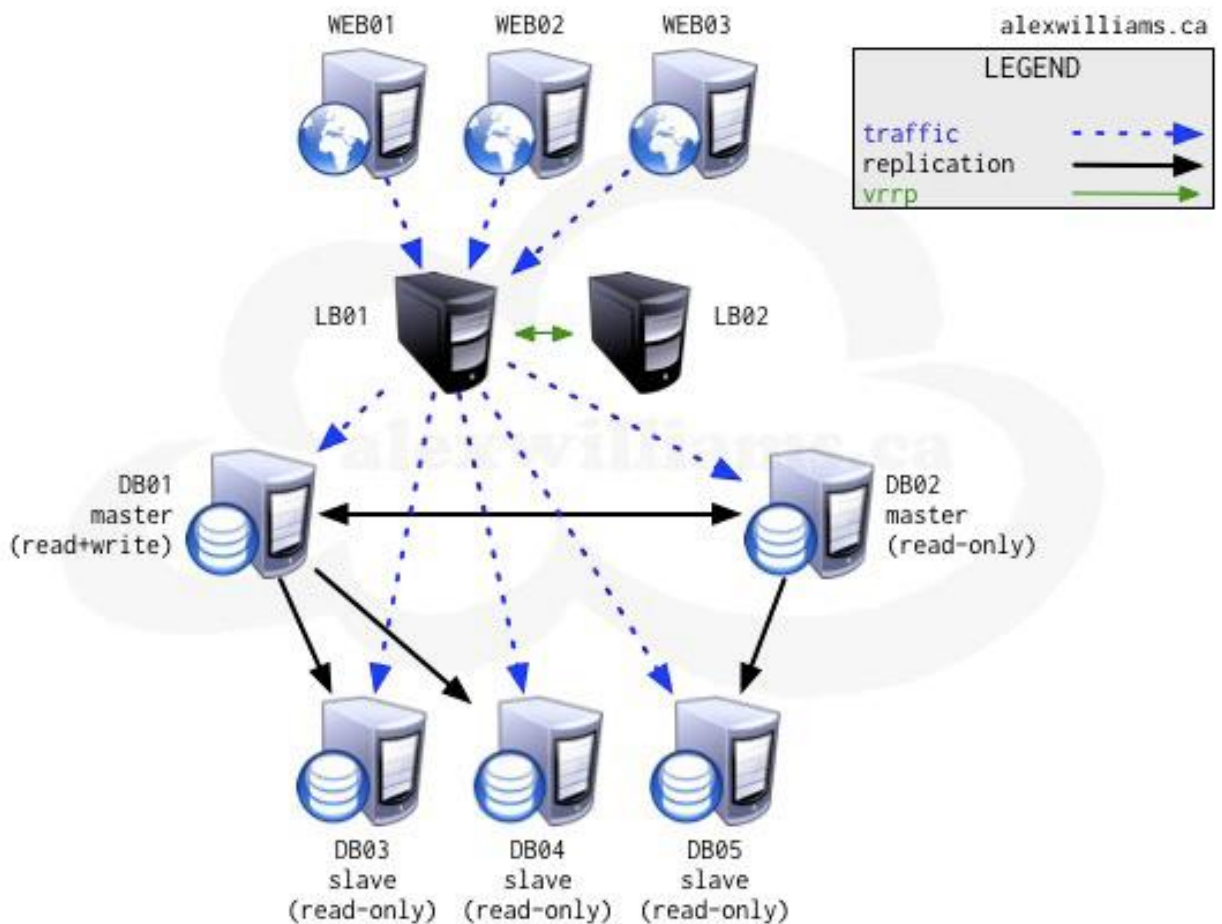
Các RDBMS hiện tại đã bộc lộ những yếu kém như việc đánh chỉ mục một lượng lớn dữ liệu, phân trang, hoặc phân phối luồng dữ liệu media (phim, ảnh, nhạc...). Cơ sở dữ liệu quan hệ được thiết kế cho những mô hình dữ liệu nhỏ thường xuyên đọc viết trong khi các Social Network Services lại có một lượng dữ liệu cực lớn và cập nhật liên tục do số lượng người dùng quá nhiều ở một thời điểm. Thiết kế trên Distributed NoSQL giảm thiểu tối đa các phép tính toán, I/O liên quan kết hợp với batch processing đủ đảm bảo được yêu cầu xử lý dữ liệu của các mạng dịch vụ dữ liệu cộng đồng này. Facebook, Amazon là những ví dụ điển hình.

Về cơ bản, các thiết kế của NoSQL lựa chọn mô hình lưu trữ tập dữ liệu theo cặp giá trị key-value. Khái niệm node được sử dụng trong quản lý dữ liệu phân tán.

Key	Value
User1923_Color	Red
User1923_Age	18
User1923_Size	Large
User1923_Name	Smith
User1923_Title	The Brown Dog

Hình 2.1: Ví dụ cơ bản về Key/ value

Với các hệ thống phân tán, việc lưu trữ chấp nhận trùng lặp dữ liệu. Một yêu cầu truy vấn dữ liệu có thể gửi tới nhiều máy cùng lúc, khi một máy nào đó bị chết cũng không ảnh hưởng nhiều tới toàn bộ hệ thống. Để đảm bảo tính thời gian thực trong các hệ thống xử lý lượng lớn dữ liệu, thông thường người ta sẽ tách biệt database ra làm 2 hoặc nhiều database như sơ đồ dưới đây:

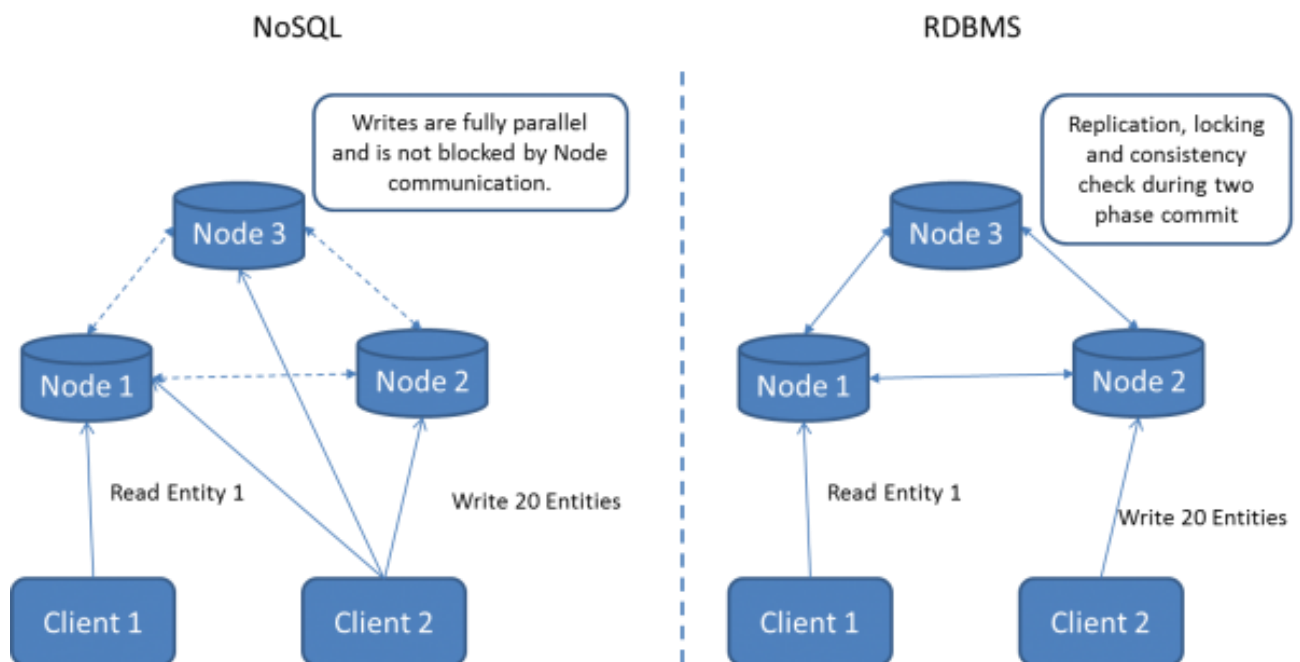


Hình 2.2: Sơ đồ thiết kế hệ thống database Master -Slave

Một database nhỏ (master database) đảm bảo vào ra liên tục, khi đạt tới ngưỡng thời gian hoặc dung lượng, database nhỏ sẽ được gộp (merge) vào database lớn có thiết kế tối ưu cho phép đọc (read operation, slave database). Mô hình đó cho phép tăng cường hiệu suất I/O - một trong những nguyên nhân chính khiến performance trở nên kém.

2.5 Một số thuật ngữ liên quan

Non-relational: relational - ràng buộc - thuật ngữ sử dụng chỉ đến các mối quan hệ giữa các bảng trong cơ sở dữ liệu quan hệ (RDBMS) sử dụng mô hình khóa gồm 2 loại khóa: khóa chính và khóa phụ (primary key + foreign key) để ràng buộc dữ liệu nhằm thể hiện tính nhất quán dữ liệu từ các bảng khác nhau. Non-relational là khái niệm không sử dụng các ràng buộc dữ liệu cho nhất quán dữ liệu ở NoSQL database.



Hình 2.3: So sánh cách thiết kế giữa NoSQL và RDBMS

Nhìn vào hình trên ta thấy NoSQL có cách thiết kế lỏng lẻo, không ràng buộc chặt chẽ như RDBMS. Các mối liên kết giữa các Node trong NoSQL chỉ là liên kết ảo, NoSQL không nhìn thấy mối liên kết gì ở đây cả. Tuy nhiên nhờ bỏ qua

tính ràng buộc này đã giúp cho NoSQL có khả năng làm việc tốt với lượng dữ liệu lớn.

Distributed storage: mô hình lưu trữ phân tán các file hoặc dữ liệu ra nhiều máy tính khác nhau trong mạng LAN hoặc Internet dưới sự kiểm soát của phần mềm.

Eventual consistency (nhất quán cuối): tính nhất quán của dữ liệu không cần phải đảm bảo ngay tức khắc sau mỗi phép write. Một hệ thống phân tán chấp nhận những ảnh hưởng theo phương thức lan truyền và sau một khoảng thời gian (không phải ngay tức khắc), thay đổi sẽ đi đến mọi điểm trong hệ thống, tức là cuối cùng (eventually) dữ liệu trên hệ thống sẽ trở lại trạng thái nhất quán.

Vertical scalable (khả năng mở rộng chiều dọc): Khi dữ liệu lớn về lượng, phương pháp tăng cường khả năng lưu trữ và xử lý bằng việc cải tiến phần mềm và cải thiện phần cứng trên một máy tính đơn lẻ được gọi là khả năng mở rộng chiều dọc. Ví dụ việc tăng cường CPUs, cải thiện đĩa cứng, bộ nhớ trong một máy tính... cho RDBMS nằm trong phạm trù này. Khả năng mở rộng chiều dọc còn có một thuật ngữ khác scale up.

Horizontal scalable (khả năng mở rộng chiều ngang):

- Khi dữ liệu lớn về lượng, phương pháp tăng cường khả năng lưu trữ và xử lý là dùng nhiều máy tính phân tán. Phân tán dữ liệu được hỗ trợ bởi phần mềm tức cơ sở dữ liệu.
- Trong khi giá thành phần cứng ngày càng giảm, tốc độ xử lý, bộ nhớ ngày càng tăng thì horizontal scalable là một lựa chọn đúng đắn. Hàng trăm máy tính nhỏ được ghép lại tạo thành một hệ thống tính toán mạnh hơn nhiều so với vi xử lý RISC truyền thống đơn lẻ. Mô hình này tiếp tục được hỗ trợ bởi các công nghệ kết nối Myrinet và InfiniBand. Từ đó chúng ta có thể quản lý, bảo trì từ xa, xây dựng batch procession (xử lý đồng loạt tập lệnh) tốt hơn. Do những đòi hỏi về tốc độ xử lý I/O cao, lượng cực lớn dữ liệu,... scale horizontally sẽ thúc đẩy các công nghệ lưu trữ mới phát triển giống như object storage devices (OSD).

2.6 So sánh NoSQL với các loại cơ sở dữ liệu khác

Để thấy sự khác biệt của NoSQL với các phương thức lưu trữ khác, chúng tôi sẽ so sánh NoSQL với XML và RDBMS. Lý do lựa chọn XML và RDBMS để so sánh là vì:

- XML là phương thức lưu trữ dữ liệu dạng văn bản tương tự như cách lưu trữ của một số NoSQL sử dụng encoding là XML hoặc JSON.
- RDBMS là hệ quản trị cơ sở dữ liệu đã rất thành công với mô hình dữ liệu quan hệ cho hệ thống vừa và nhỏ.

2.6.1 So sánh NoSQL với XML

Cả NoSQL và XML đều có phương thức lưu trữ tương tự nhau: lưu dạng văn bản. XML dùng để lưu trữ dữ liệu sử dụng các thẻ đánh dấu. Tuy nhiên để sử dụng XML như một cơ sở dữ liệu sẽ có một số thuận lợi và khó khăn như sau:

- Thuận lợi: có thể kiểm soát tất cả, nắm được luồng xử lý của hệ thống.
- Khó khăn: XML chỉ là các file văn bản nên không có một platform cho truy xuất dữ liệu do đó cần phải xây dựng mới hoàn toàn lớp thao tác dữ liệu với XML như insert, delete, update, query như vậy rất tốn chi phí. Ngoài ra việc tự xây dựng nhiều khi mang đến một kết quả không tốt ví dụ như source code chưa được tối ưu, chưa có giải thuật tốt.

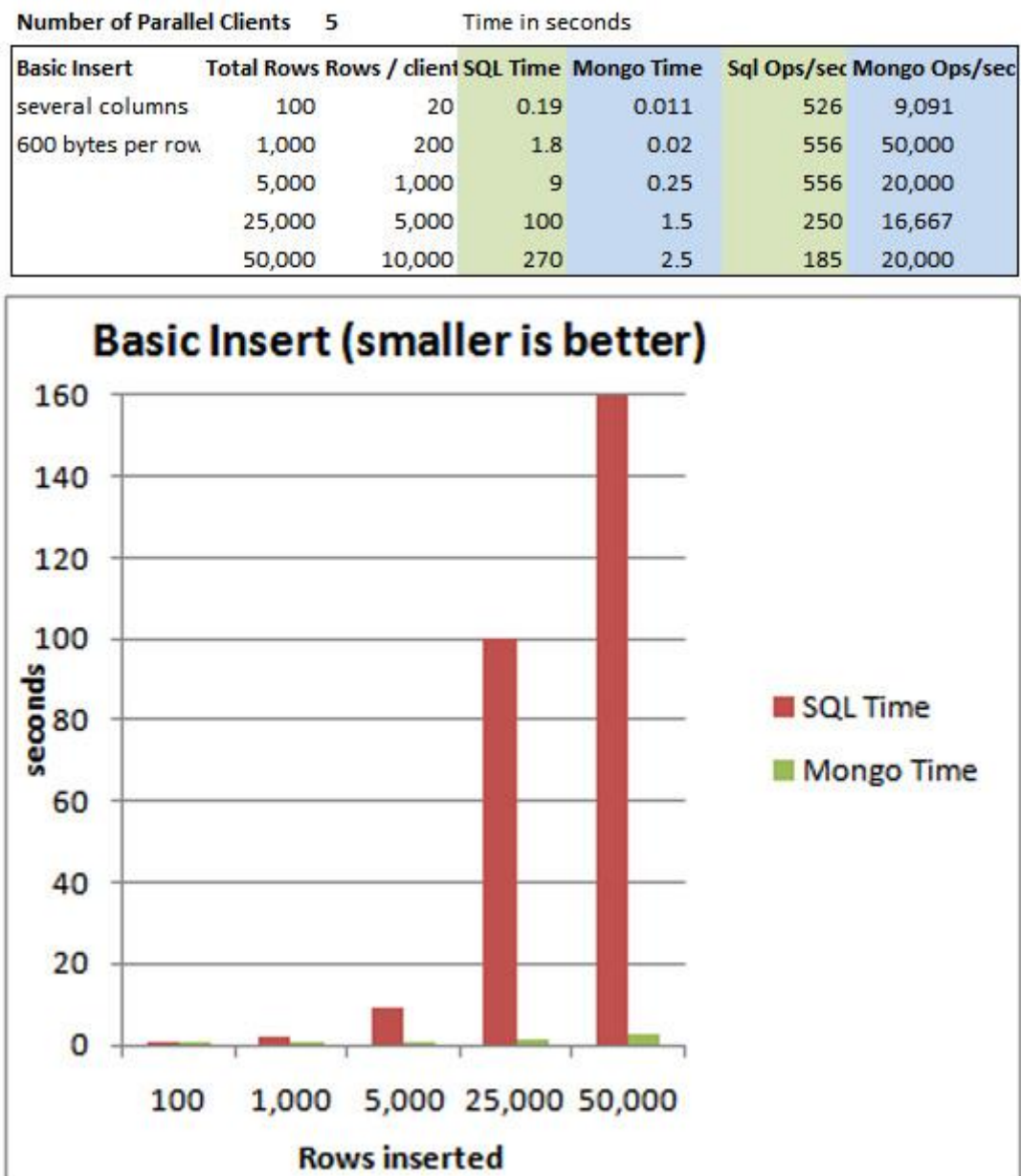
Do đó, hiệu suất hoạt động có tốt hay không phụ thuộc rất nhiều vào lớp mới mà người lập trình tạo ra. Nên không có một đảm bảo nào cho việc sử dụng XML có thể cho hiệu suất tốt hơn NoSQL khi mà:

- Cơ sở dữ liệu NoSQL được các nhà lập trình chuyên nghiệp xây dựng ra với nhiều giải thuật, khả năng tối ưu source code cao mang đến một hiệu suất làm việc tuyệt vời.
- Các đặc điểm khác như: khả năng phân tán dữ liệu, đánh số index, phân trang, transaction hoặc các gói hỗ trợ nâng cao như bảo mật, mã hoá thông tin... thì khó mà lập trình ra.

Như vậy, NoSQL đã làm tốt nhiệm vụ của nó đồng thời còn là mã nguồn mở thì ta đâu có lý do gì phải xây dựng một hệ thống lưu trữ mới dựa trên các file XML đầy khó khăn.

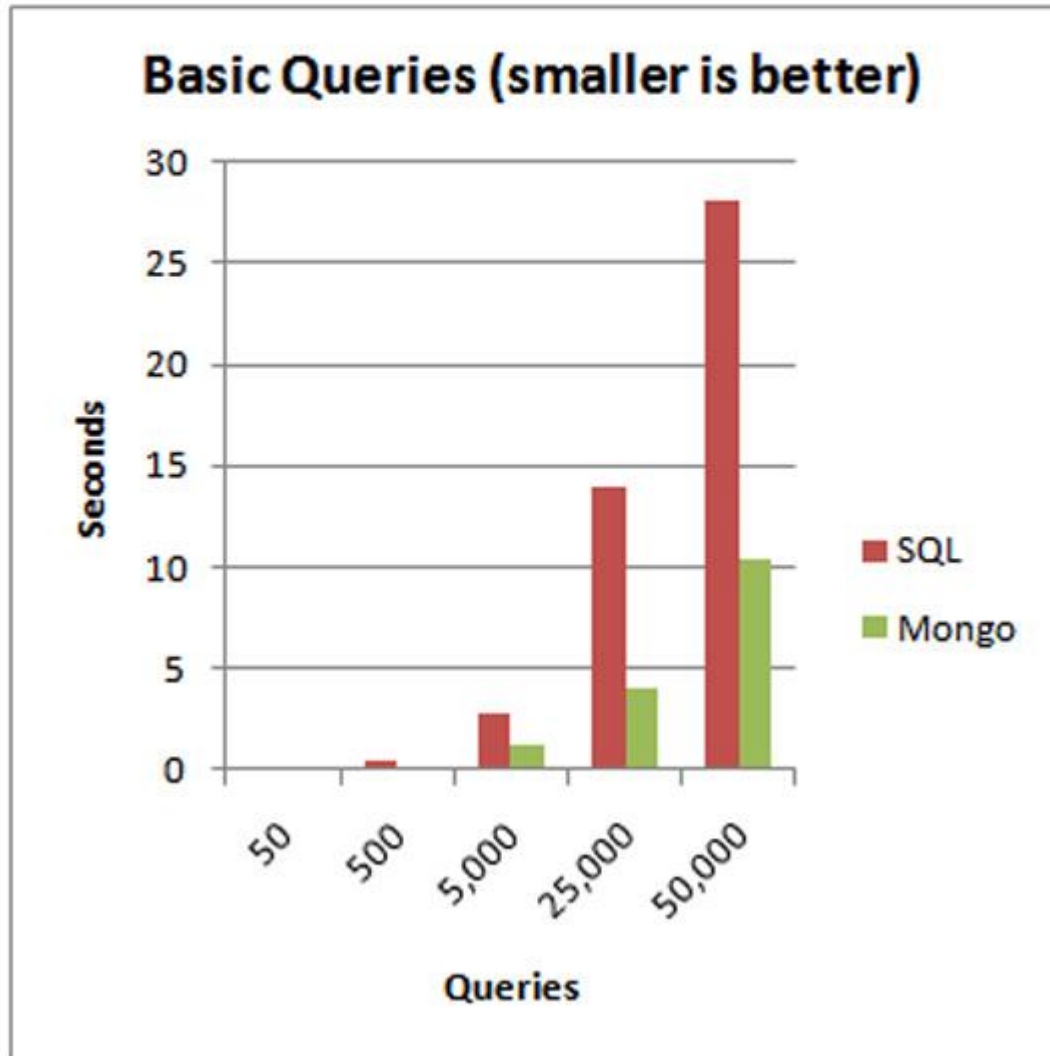
2.6.2 So sánh NoSQL với RDBMS

Như đã đề cập ở trên, cơ sở dữ liệu NoSQL sinh ra để giải quyết các thiếu sót của RDBMS trong các hệ thống phần mềm hiện đại. NoSQL sẽ tập trung giải quyết các vấn đề như tốc độ thực thi, khả năng lưu trữ, các nghiệp vụ phức tạp (phân trang, đánh chỉ mục, ...) và thật sự NoSQL đã làm được điều đó. Để thấy hiệu suất mà NoSQL đạt được, hãy xem 2 biểu đồ dưới đây là kết quả của phép so sánh giữa MongoDB (một cơ sở dữ liệu NoSQL) và MSSQL 2008.



Hình 2.4: Kết quả insert data

Number of Parallel Clients		Time in seconds				
	Total Rows	Rows / client	SQL Time	Mongo Time	Sql Ops/sec	Mongo Ops/sec
Basic Query with index	50	10	0.1	0.08	500	625
	500	100	0.38	0.1	1,316	5,000
	5,000	1,000	2.8	1.2	1,786	4,167
	25,000	5,000	14	4	1,786	6,250
	50,000	10,000	28	10.4	1,786	4,808



Hình 2.5: Kết quả truy vấn data

Bảng so sánh sau sẽ cho chúng ta thấy sự khác nhau giữa NoSQL và RDBMS:

Đặc điểm	CSDL quan hệ	NoSQL
Hiệu suất	Kém hơn SQL Relational giữa các table	Cực tốt Bỏ qua SQL Bỏ qua các ràng buộc dữ liệu
Khả năng mở rộng	Hạn chế về lượng.	Hỗ trợ một lượng rất lớn các node.
Hiệu suất đọc-ghi	Kém do thiết kế để đảm bảo sự vào/ra liên tục của dữ liệu	Tốt với mô hình xử lý lô và những tối ưu về đọc-ghi dữ liệu.
Thay đổi số node trong hệ thống	Phải shutdown cả hệ thống. Việc thay đổi số node phức tạp.	Không cần phải shutdown cả hệ thống. Việc thay đổi số node đơn giản, không ảnh hưởng đến hệ thống.
Phần cứng	Đòi hỏi cao về phần cứng.	Đòi hỏi thấp hơn về giá trị và tính đồng nhất của phần cứng

Bảng 2.2: So sánh giữa NoSQL và cơ sở dữ liệu quan hệ

Như vậy, NoSQL khắc phục nhiều nhược điểm của cơ sở dữ liệu quan hệ và mang đến một giải pháp rất tốt cho nhu cầu lưu trữ dữ liệu lớn.

2.7 Cách triển khai một ứng dụng NoSQL

Trong phạm vi của luận văn này, chúng tôi chỉ tập trung vào loại phổ biến nhất trong cơ sở dữ liệu NoSQL đó là loại Document Store. Do đó trong mục này, chúng tôi sẽ trình bày cách triển khai một ứng dụng sử dụng cơ sở dữ liệu NoSQL loại Document Store. Để hiểu rõ các loại này, vui lòng xem “Chương 3: Phân loại cơ sở dữ liệu NoSQL”.

2.7.1 Xác định NoSQL có phù hợp

Khi làm việc với một lượng lớn dữ liệu, bạn hãy nghĩ đến NoSQL. NoSQL rất thích hợp để làm việc với dữ liệu lớn bằng cách loại bỏ các ràng buộc toàn vẹn dữ

liệu, cách thiết kế mô hình phi chuẩn hoá, cách sử dụng index.... Đã giúp NoSQL trở nên mạnh mẽ để làm việc với lượng lớn dữ liệu. Tuy nhiên, có một số tính chất sau đây cần lưu ý khi lựa chọn cơ sở dữ liệu NoSQL.

Như đã đề cập trong mục 2.5 Một số thuật ngữ liên quan, tính nhất quán cuối (Eventual consistency) cần phải được ứng dụng chấp nhận. Có nghĩa là ứng dụng không yêu cầu ràng buộc dữ liệu, không yêu cầu dữ liệu phải cập nhật chính xác ngay tức thì. Một số ứng dụng phù hợp như các trang mạng xã hội, các ứng dụng ghi log tự động... Các ứng dụng loại này chấp nhận dữ liệu cũ trong một khoảng thời gian ngắn trước khi được cập nhật mới. Đổi lại chúng ta đạt được những tiêu chuẩn cao về khả năng mở rộng và hiệu quả về chi phí, trong khi phục vụ liên tục hàng triệu khách hàng từ khắp nơi trên trái đất. Đặc biệt chúng ta đạt được một hiệu suất hoạt động cao hơn gấp nhiều lần nhờ vào việc loại bỏ các yêu cầu nhất quán dữ liệu.

Các ứng dụng không phù hợp với cơ sở dữ liệu NoSQL là các ứng dụng yêu cầu tính nhất quán dữ liệu cao. Tính nhất quán dữ liệu được xem như tính sống còn của ứng dụng. Ví dụ như các ứng dụng tài chính, ngân hàng... với các con số luôn được cập nhật và cần được cập nhật tức thì. Sự chậm trễ có thể phải trả giá rất đắt. Bởi thế nếu các ứng dụng của bạn thuộc loại này thì hãy lựa chọn cơ sở dữ liệu RDBMS với mô hình quan hệ truyền thống.

Các yêu cầu phân tích hiện đại (BI - business intelligence) cũng không phù hợp với cơ sở dữ liệu NoSQL này. Bởi vì NoSQL hỗ trợ rất ít các câu truy vấn. Tất cả đều phụ thuộc vào sự tinh thông lập trình. Như vậy, với một yêu cầu phân tích đơn giản thì cũng cần đến lập trình trong đó. Trong khi với cơ sở dữ liệu RDBMS sử dụng ngôn ngữ SQL để truy vấn, SQL giúp chúng ta rất nhiều việc trong truy vấn, phân tích.

2.7.2 Thiết kế cấu trúc dữ liệu dạng document

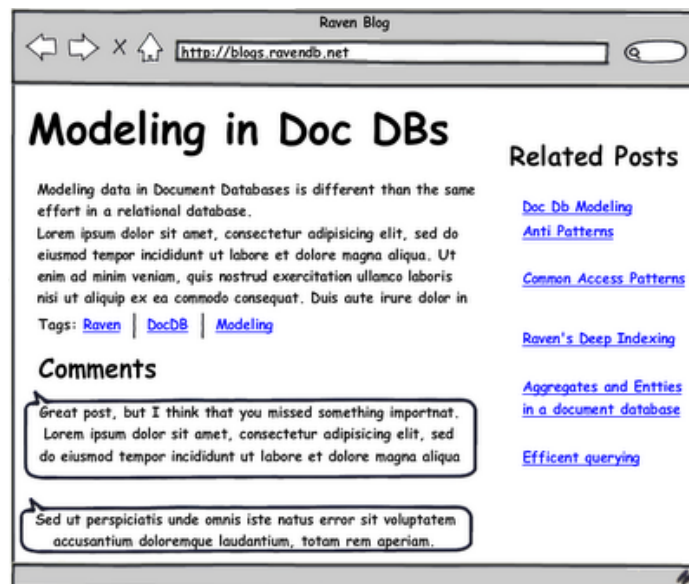
NoSQL lưu trữ dữ liệu không theo một lược đồ cố định, nó có lược đồ tùy ý tùy biến. Nhưng điều đó không có nghĩa rằng chúng ta không nên dành nhiều thời gian để xem xét làm thế nào để thiết kế các document đảm bảo rằng chúng ta có thể truy cập tất cả dữ liệu chúng ta cần để phục vụ các yêu cầu của người dùng một cách hiệu quả, đáng tin cậy và chi phí bảo trì ít nhất có thể.

Lỗi điển hình nhất mà chúng ta mắc phải là cố gắng thiết kế mô hình dữ liệu của document database giống với cách chúng ta thiết kế mô hình dữ liệu trong cơ sở dữ liệu quan hệ. NoSQL lưu trữ dữ liệu phi quan hệ. Nếu cố gắng thiết kế theo mô hình quan hệ thì chúng ta có được nhiều kết quả tốt. Nhưng chúng ta sẽ đạt được kết quả vô cùng to lớn nếu sử dụng những điểm mạnh của document database.

Để thiết kế cấu trúc dữ liệu dạng document chúng ta cần lưu ý 2 điểm sau: các document có cấu trúc không giống nhau và thiết kế dữ liệu phi quan hệ.

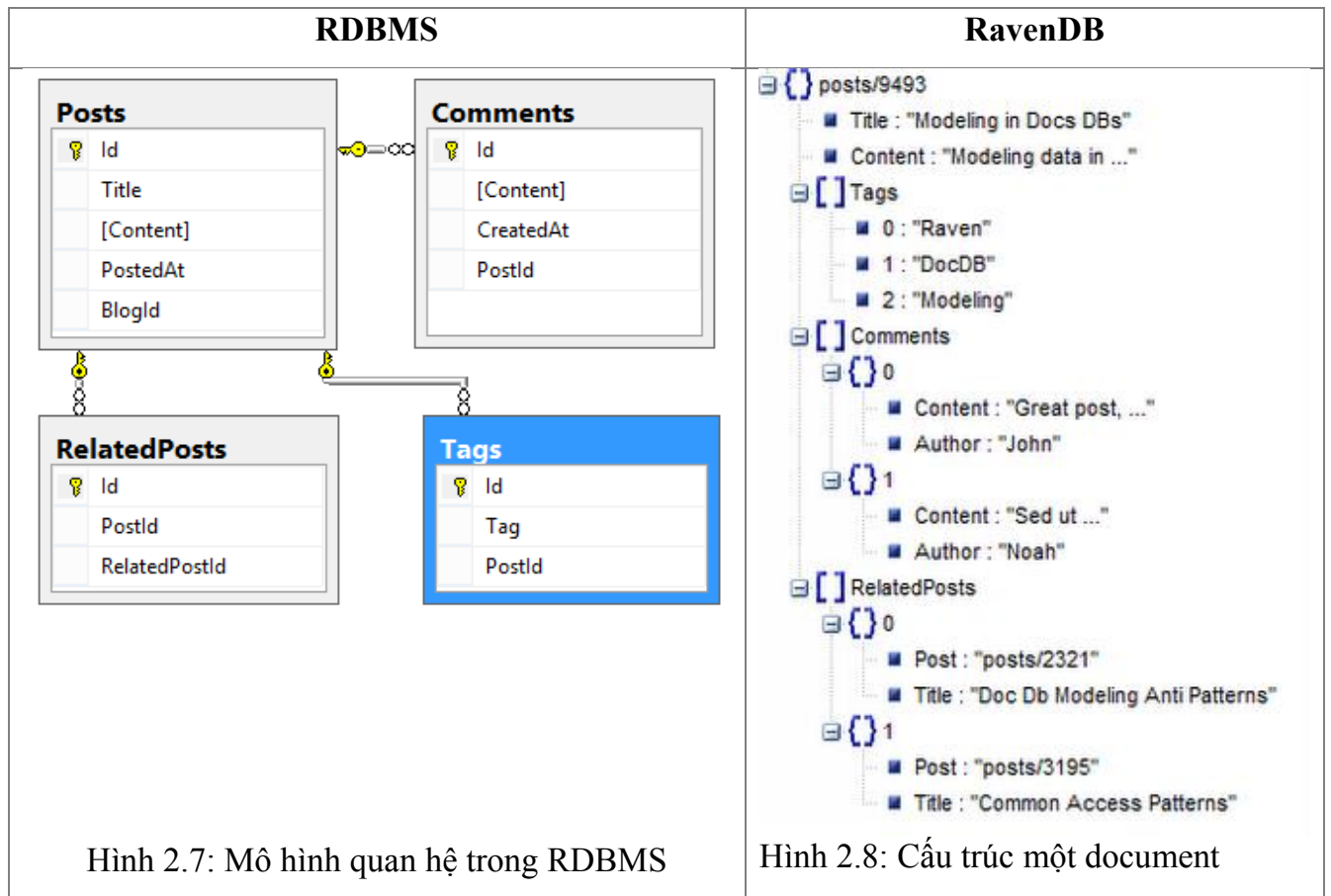
2.7.2.1 Các document có cấu trúc không giống nhau (Document is not flat)

Trong RDBMS, một dòng chỉ có thể chứa dữ liệu đơn giản và những cấu trúc dữ liệu phức tạp hơn cần được lưu trữ như là mối quan hệ (có nghĩa là lưu dữ liệu trong nhiều bảng khác nhau và sử dụng khóa ngoại để tham chiếu). Đối với document database, một document có thể lưu một đối tượng phức tạp tùy ý. Đối tượng đó có thể là arrays, dictionaries và trees. Xem ví dụ blog đơn giản dưới đây:



Hình 2.6: Ví dụ về blog đơn giản

Trong cơ sở dữ liệu quan hệ, cần ít nhất 4 table để hiển thị dữ liệu trong một trang đơn (Posts, Comments, Tags, RelatedPosts). Sử dụng RavenDB, chúng ta lưu trữ tất cả thông tin chúng ta cần vào trong một document.



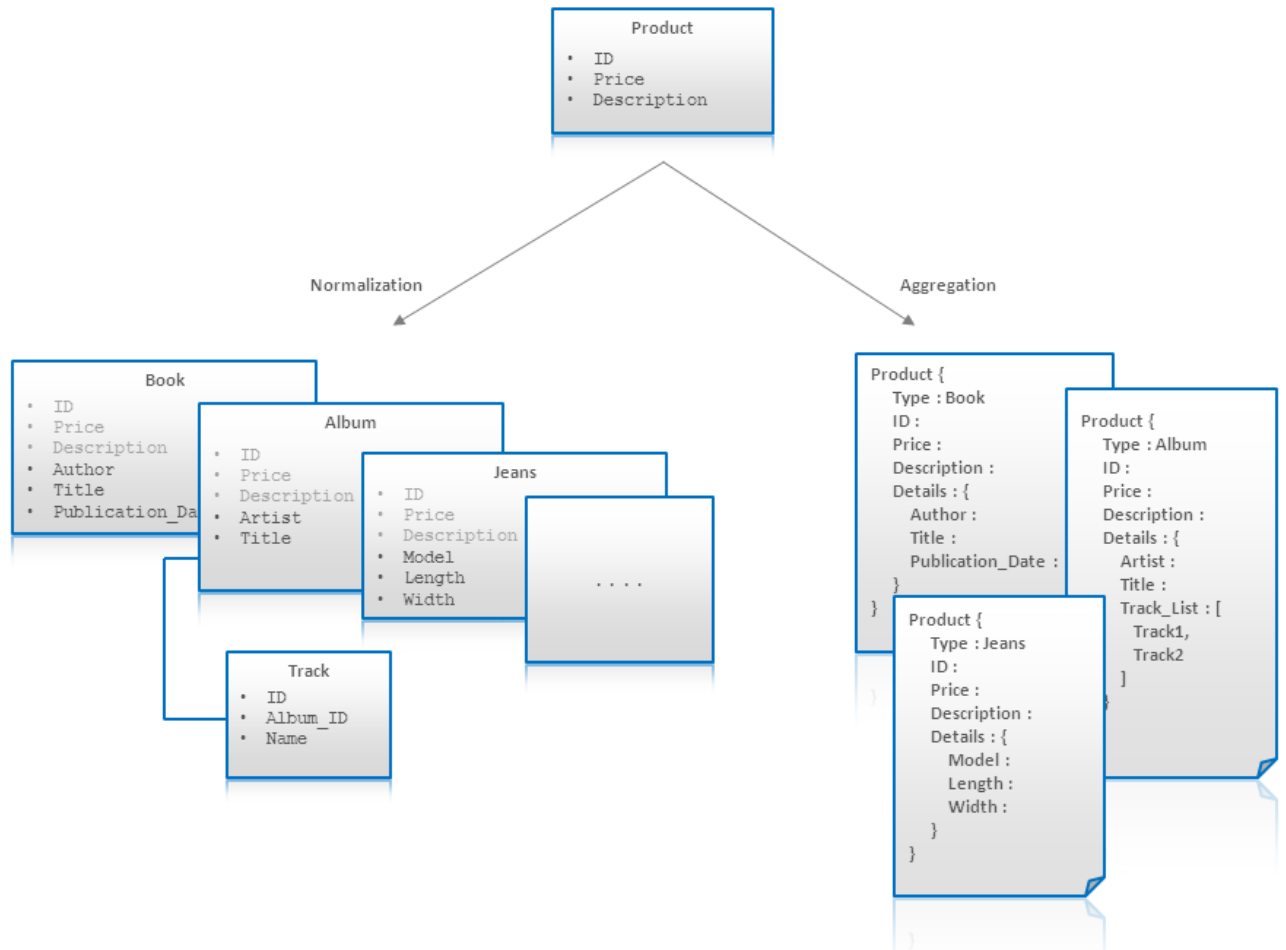
Với cấu trúc document này, chúng ta có thể lấy mọi thông tin chúng ta cần để hiện thị lên trang chỉ trong một yêu cầu.

2.7.2.2 Thiết kế dữ liệu phi quan hệ

Khi bắt đầu dùng RavenDB, chúng ta sẽ gặp những vấn đề khi chúng ta cố gắng sử dụng các khái niệm của cơ sở dữ liệu quan hệ. Vấn đề chính đó là Raven phi quan hệ. Raven xem mỗi document như một thực thể độc lập.

Ví dụ yêu cầu quản lý thông tin sản phẩm (Product). Các thông tin của một sản phẩm gồm có: ID, giá, mô tả sản phẩm.

- Đối với sản phẩm sách có thêm thông tin: tác giả, tiêu đề, ngày xuất bản.
- Đối với sản phẩm Album nhạc có thêm thông tin: nhạc sĩ, tên Album.
Trong mỗi Album có nhiều bài hát, mỗi bài hát có tên bài hát.
- Đối với sản phẩm quần Jean có thêm thông tin: Model, chiều dài, chiều rộng.



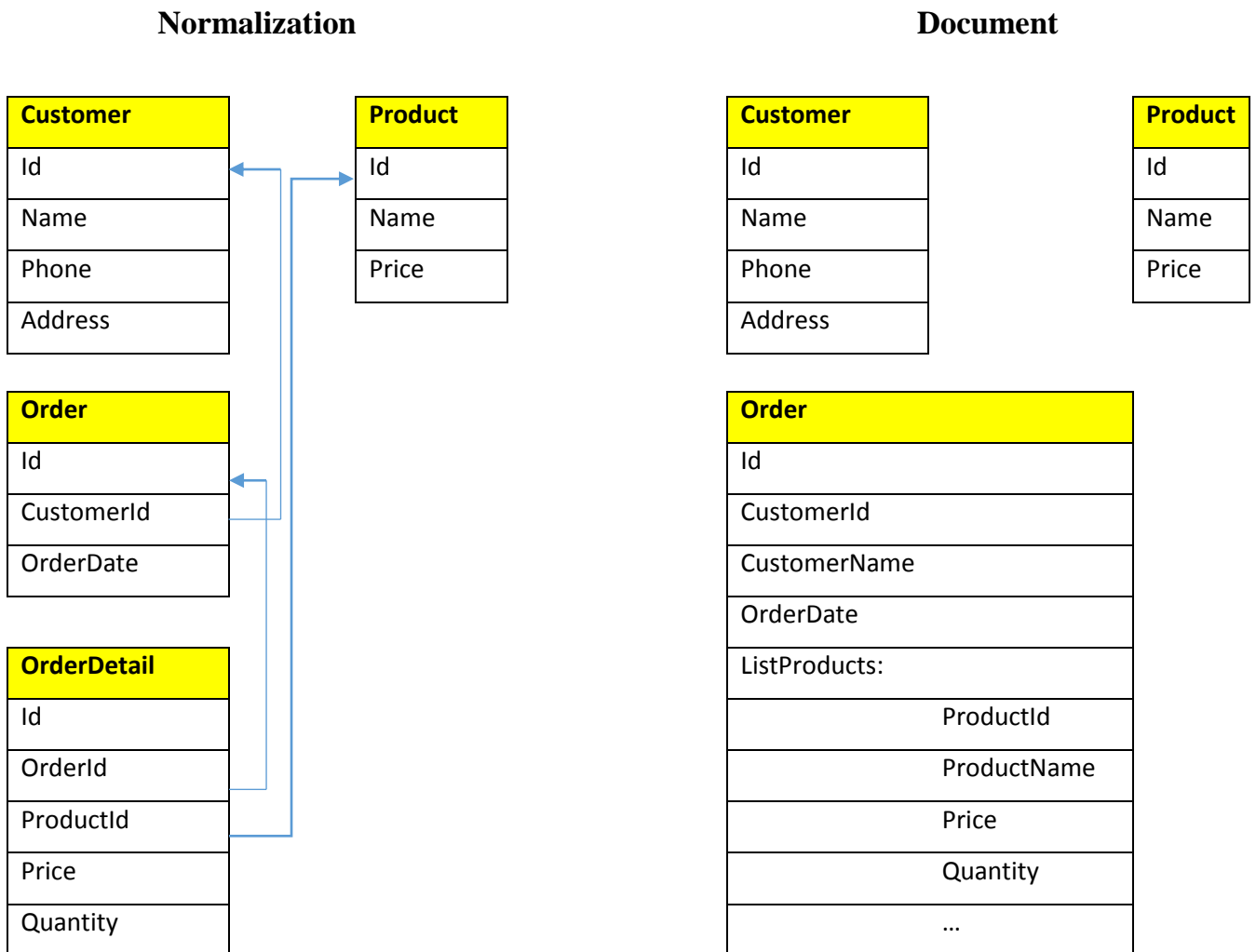
Hình 2.8: Ví dụ về thiết kế dữ liệu chuẩn hoá và document của NoSQL

Với thiết kế chuẩn hoá, các table quan hệ khoá ngoại với nhau tạo nên tính nhất quán dữ liệu. Nhưng với cách thiết kế document, chúng ta gom tất cả vào một document và không chia ra nhiều table. Nên khi cần truy xuất dữ liệu, chúng ta chỉ cần một vài truy vấn đã lấy được tất cả dữ liệu cần thiết mà không cần dùng đến các khoá ngoại rườm rà.

Tóm lại, tư tưởng thiết kế ở đây là đi ngược lại với thiết kế chuẩn hoá, mục tiêu sao cho hạn chế các phép “join” rườm rà. Ở đây chúng ta có thể chấp nhận dữ liệu dư thừa và không thống nhất trong 1 khoảng thời gian và sau đó sẽ được cập nhập lại. Bởi ta nhận được một hiệu suất hoạt động mạnh mẽ với lượng lớn dữ liệu.

Vấn đề đặt ra khi ta cần cập nhập dữ liệu. Như ví dụ sau đây, tên của “Customer” cần được cập nhập. Đối với thiết kế chuẩn hoá, ta chỉ cần cập nhập ở 1

nơi là table Customer. Nhưng đối với thiết kế document thì khác, tên của Customer đặt ở nhiều nơi: trong object Customer và trong các object Order.

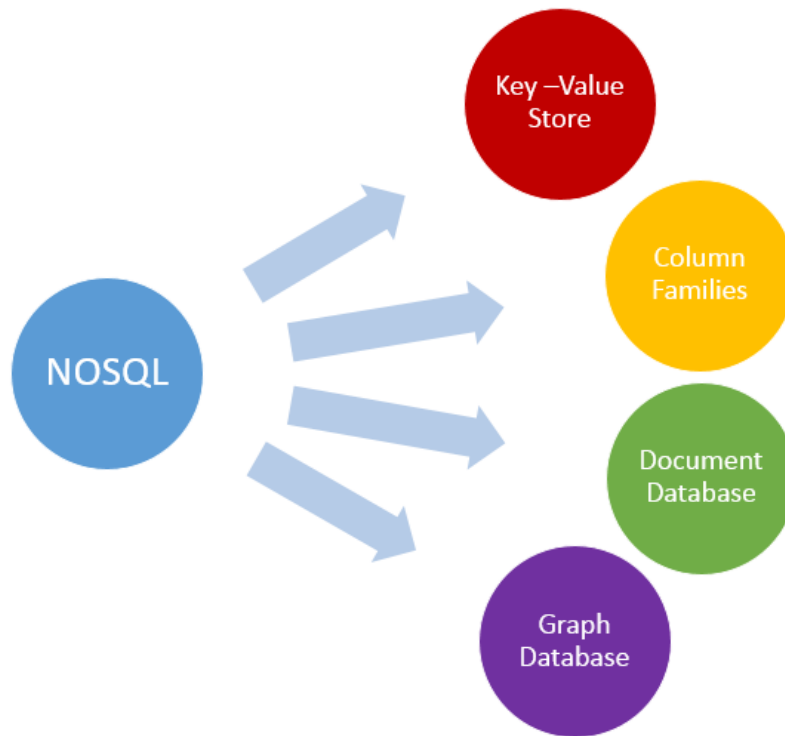


Hình 2.9: Ví dụ về thiết kế dữ liệu chuẩn hoá và document của NoSQL

Đến đây thì không có một quy tắc nào hết. Việc cập nhập lại tên của Customer là phụ thuộc vào nghiệp vụ của chương trình. Khi xây dựng chương trình, ta phân tích xem tên của Customer có cần được cập nhập ở tất cả các nơi hay chỉ cần ở 1 số nơi. Từ đó ta sẽ viết code cho việc cập nhập này. Tất cả đều do phân tích cho từng chương trình sao cho hiệu suất hoạt động tốt nhất và nghiệp vụ vẫn đúng.

CHƯƠNG 3 – PHÂN LOẠI CƠ SỞ DỮ LIỆU NOSQL

Cơ sở dữ liệu NoSQL được phân loại theo cách mà nó lưu trữ dữ liệu và gồm có 4 loại chính:



3.1 Key-Value Store

Cơ sở dữ liệu NoSQL đơn giản nhất chính là Key/Value stores. Nó đơn giản nhất là vì những API của nó đơn giản, những triển khai thực tế của NoSQL thường rất phức tạp. Hầu hết Key/Value stores thường có những API sau:

```
void Put(string key, byte[] data);
```

```
byte[] Get(string key);
```

```
void Remove(string key);
```

key	value
firstName	Bugs
lastName	Bunny
location	Earth

Hình 3.1: Key-Value store

Với key-value store thì việc truy xuất, xóa, cập nhật giá trị thực (value) đều thông qua key tương ứng. Giá trị được lưu dưới dạng BLOB (Binary large object). Xây dựng một key/value store rất đơn giản và mở rộng chúng cũng rất dễ dàng. Key/value store có hiệu suất rất tốt bởi vì mô hình truy cập dữ liệu trong key/value store được tối ưu hóa tối đa. Key/Value store là cơ sở cho tất cả những loại cơ sở dữ liệu NOSQL khác.

Key-value store rất hữu ích khi chúng ta cần truy cập dữ liệu theo khóa. Ví dụ như chúng ta cần lưu trữ thông tin phiên giao dịch hoặc thông tin giỏ hàng của người dùng thì key-value store là một sự lựa chọn hợp lý bởi vì nhờ vào id của người dùng chúng ta có thể nhanh chóng lấy được các thông tin liên quan trong phiên giao dịch hoặc giỏ hàng của người dùng đó. Giỏ mua hàng của Amazon chạy trên key value store (Amazon Dynamo). Vì thế có thể thấy rằng key-value store có khả năng mở rộng cao.

Một số loại key-value store phổ biến:

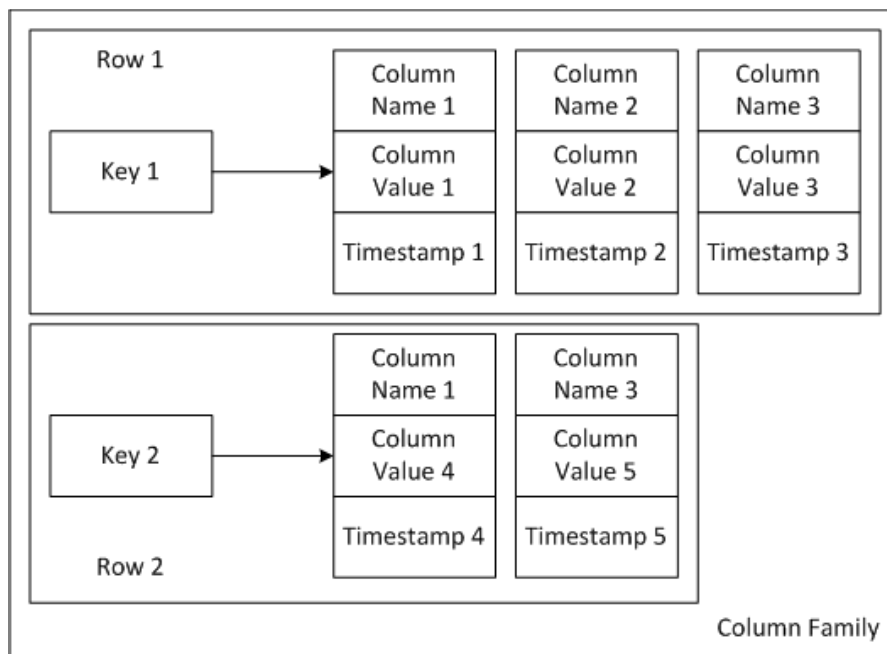
- Key/value cache in RAM: memcached, Citrusleaf database, Velocity, Redis...
- Key/value save on disk: Memcachedb, Berkeley DB, Tokyo Cabinet, Redis...
- Eventually Consistent Key Value Store: Amazon Dynamo, Voldemort, Dynomite, KAI, Cassandra, Hibari, Project Voldemort...
- Ordered key-value store: NMDB, Memcachedb, Berkeley DB...
- Distributed systems: MEMBASE, Azure Table Storage, Amazon Dynamo ...

3.2 Column Families / Wide Column Store

Column families database là hệ cơ sở dữ liệu phân tán cho phép truy xuất ngẫu nhiên / tức thời với khả năng lưu trữ một lượng cực lớn dữ liệu có cấu trúc. Dữ liệu tồn tại dạng bảng ghi và mỗi bảng ghi có thể chứa rất nhiều cột.

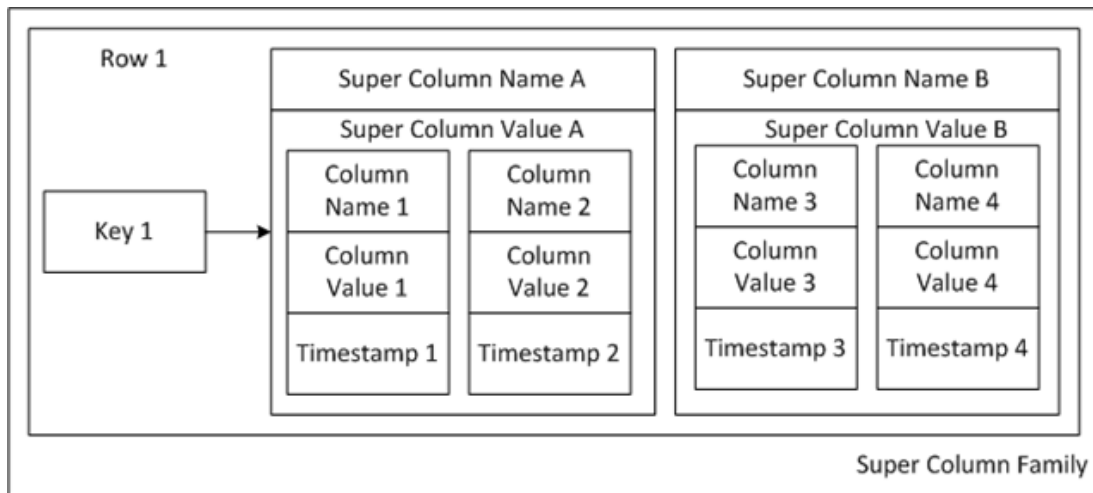
Một trong những khác biệt với cơ sở dữ liệu quan hệ đó chính là việc lưu trữ dữ liệu theo cột (trong column family databases) so với việc lưu trữ dữ liệu theo dòng (trong cơ sở dữ liệu quan hệ). Trong column family database có khái niệm sau: column family, super column, column. Các khái niệm này rất quan trọng để hiểu được column family database làm việc như thế nào:

- Column family (cột quan hệ): Một column family là cách thức dữ liệu được lưu trữ trên đĩa cứng. Tất cả dữ liệu trong một cột sẽ được lưu trên cùng một file. Một column family có thể chứa super column hoặc column.



Hình 3.2: Column Famies

- Super column: Một super column có thể được dùng như một dictionary (kiểu từ điển). Nó là một column có thể chứa những column khác (mà không phải là super column).



Hình 3.3: Super Column

- Column: Một column là một bộ gồm tên, giá trị và dấu thời gian (thông thường chỉ quan tâm tới key-value).

3.3 Document database

Khái niệm trung tâm của document database là khái niệm “document”. Tất cả documents đều được đóng gói và mã hóa dữ liệu dưới định dạng tiêu chuẩn. Một số định dạng được sử dụng bao gồm XML, YAML, JSON và BSON cũng như kiểu nhị phân như PDF và các tài liệu Microsoft Office (MS Word, Excel ...). Trên thực tế, tất cả document database đều sử dụng JSON(hoặc BSON) hoặc XML.

Các document bên trong một document database thì tương tự nhau, nó gần giống với khái niệm record/row trong cơ sở dữ liệu quan hệ nhưng nó ít cứng nhắc hơn. Documents không bắt buộc phải tuân theo một lược đồ tiêu chuẩn cũng không cần phải có tất cả các thuộc tính, khóa tương tự nhau. Xem ví dụ dưới đây:

Document 1	Document 2
<pre>{ FirstName:"Bob", Address:"5 Oak St.", Hobby:"sailing" }</pre>	<pre>{ FirstName:"Jonathan", Address:"15 Wanamassa Point Road", Children:[{Name:"Michael",Age:10}, {Name:"Elena", Age:2}] }</pre>

Bảng 3.1: So sánh 2 document

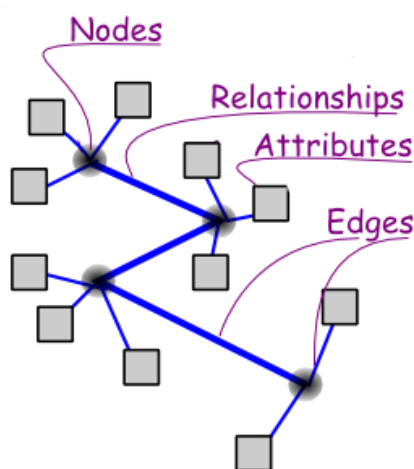
Cả hai document trên có một số thông tin tương tự và một số thông tin khác nhau. Không giống như một cơ sở dữ liệu quan hệ truyền thống, nơi mỗi record(row) có cùng một tập hợp trường dữ liệu (fields hay columns) và các trường dữ liệu này nếu không sử dụng thì có thể được lưu trữ rỗng(empty), còn trong document database thì không có trường dữ liệu rỗng trong document. Hệ thống này cho phép thông tin mới được thêm vào mà không cần phải khai báo rõ ràng.

Các document được đánh dấu trong document database thông qua một khóa duy nhất đại diện cho document đó. Thông thường, khóa này là một chuỗi đơn giản. Trong một số trường hợp, chuỗi này có thể là một URI hoặc đường dẫn (path). Chúng ta có thể sử dụng khóa này để lấy document từ cơ sở dữ liệu. Thông thường, cơ sở dữ liệu vẫn lưu lại một chỉ số (index) trong khóa của document để document có thể được tìm kiếm nhanh chóng.

Các document database phổ biến là: BaseX, ArangoDB, Clusterpoint, Couchbase Server, CouchDB, eXist, FleetDB, Jackrabbit, Lotus Notes, MongoDB, MUMPSDatabase, OrientDB, Apache Cassandra, Redis, Rocket U2, RavenDB....

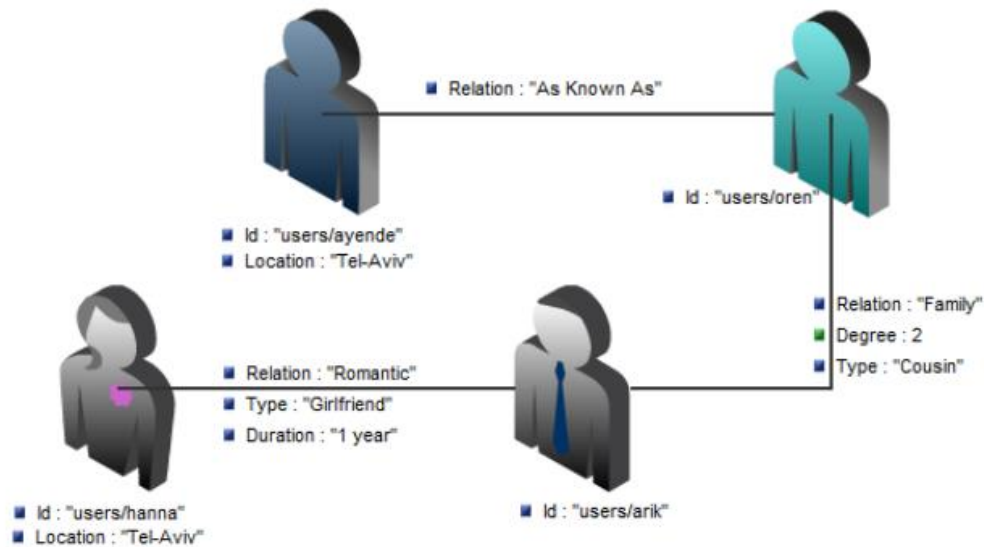
3.4 Graph Database

Graph database là một dạng cơ sở dữ liệu được thiết kế riêng cho việc lưu trữ thông tin đồ thị như cạnh, nút, các thuộc tính.



Hình 3.4: Graph database

Chúng ta có thể nghĩ graph database như một document database với các kiểu document đặc biệt và các mối quan hệ. Một ví dụ điển hình đó chính là mạng xã hội, có thể xem hình bên dưới:



Hình 3.5: Ví dụ về các nút trong một graph database

Trong ví dụ trên ta có 4 document và 3 mối quan hệ. Mỗi quan hệ trong graph database thì có ý nghĩa nhiều hơn con trỏ đơn thuần. Một mối quan hệ có thể một chiều hoặc hai chiều nhưng quan trọng hơn là mối quan hệ được phân loại. Một người có thể liên kết với người khác theo nhiều cách, có thể là khách hàng, có thể là người trong gia đình... Mỗi quan hệ tự bản thân nó có thể mang thông tin. Trong ví dụ trên ta chỉ lưu lại loại quan hệ và mức độ gần gũi (bạn bè, người trong gia đình, người yêu...).

Graph database thường được sử dụng để giải quyết các vấn đề về mạng. Trong thực tế, hầu hết các trang web mạng xã hội đều sử dụng một số hình thức của graph database để làm những việc mà chúng ta đã biết như: kết bạn, bạn của bạn... Một vấn đề đối với việc mở rộng graph database là rất khó để tìm thấy một đồ thị con độc lập, có nghĩa là rất khó để ta phân tán graph database thành nhiều mảnh.

Một số sản phẩm tiêu biểu của graph database là: Neo4J, Sones, AllegroGraph, Core Data, DEX, FlockDB, InfoGrid, OpenLink Virtuoso...

3.5 Làm sao để lựa chọn một giải pháp cơ sở dữ liệu tốt

Như các phần trên đã đề cập đến các giải pháp cơ sở dữ liệu NoSQL thì mỗi loại trong số đó có những điểm mạnh và điểm yếu riêng của nó. Một câu hỏi chúng ta thường hay gặp là: “Tôi muốn sử dụng công nghệ NoSQL X cho việc Y thì làm sao?”. Với câu hỏi này, chúng ta thường gặp phải vấn đề là:

- Cố gắng áp dụng các khái niệm, kỹ thuật, kinh nghiệm của mô hình cơ sở dữ liệu quan hệ truyền thống vào trong NOSQL.
- Cố gắng sử dụng một loại cơ sở dữ liệu NoSQL trên toàn bộ ứng dụng mà có thể có những phần khác nhau của ứng dụng không phù hợp với cơ sở dữ liệu NoSQL này.

Trong một ứng dụng, chúng ta có thể sử dụng key-value store để lưu trữ thông tin phiên làm việc (session), sử dụng graph database để phục vụ những truy vấn xã hội và document database để lưu trữ các thực thể. Nếu chúng ta lưu trữ dữ liệu theo một loại cơ sở dữ liệu NoSQL duy nhất thì việc này giống như chúng ta muốn lưu trữ tất cả code trên một file duy nhất. Chúng ta có thể làm được việc này nhưng có vẻ vụng về, không được tối ưu lắm. Điều nên làm là cố gắng phân chia ứng dụng thành từng phần mà mỗi phần thích hợp với một mô hình truy cập dữ liệu để đem lại hiệu quả cao nhất. Ví dụ như trong danh mục sản phẩm luôn làm việc với những truy vấn bởi Product SKU và tốc độ là cốt yếu thì ta nên sử dụng key-value store. Nhưng điều đó không có nghĩa là đơn đặt hàng cũng được lưu trữ ở đó mà chúng ta cần tính linh hoạt hơn nên sẽ sử dụng document database...

Kết luận: Trong một ứng dụng chúng ta có thể sử dụng nhiều công nghệ lưu trữ dữ liệu khác nhau để làm cho ứng dụng của chúng ta hoạt động tốt nhất và mỗi phần khác nhau của ứng dụng có thể sử dụng công nghệ khác nhau sao cho phù hợp với mục đích của chúng ta. Điều đó cũng nói lên rằng: trong hệ thống sử dụng nhiều công nghệ lưu trữ, một công nghệ lưu trữ dữ liệu mới chỉ thực sự có nghĩa khi mà lợi ích nó mang lại lớn hơn chi phí phải trả để sử dụng công nghệ đó. Nếu chúng ta cần hỗ trợ lưu trữ các trường dữ liệu người dùng tự định nghĩa thì chúng ta nhanh chóng sử dụng document database hơn là cố gắng thực hiện điều đó với RDBMS.

Lưu ý: Không nên quên RDMBS. NoSQL thực sự là viết tắt của Not only SQL(không chỉ SQL). NoSQL đảm nhận những phần mà RDBMS chưa làm tốt chứ không phải là để thay thế RDBMS. Vì thế, khi chọn công nghệ lưu trữ dữ liệu chúng ta cần quan tâm tới việc kết hợp với RDBMS. RDBMS là một công cụ rất mạnh mẽ và không nên bị bỏ đi chỉ vì đối thủ còn non trẻ và hấp dẫn hơn.

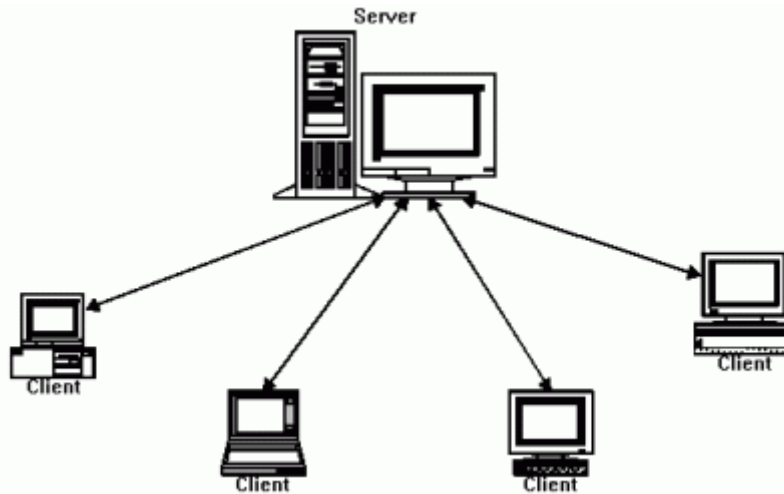
Trong đề tài này, chúng tôi chọn Document Database làm cơ sở dữ liệu NoSQL để xây dựng ứng dụng chính. Những lý do mà chúng tôi chọn Document Store là:

- Dữ liệu trong document database được lưu trữ dưới định dạng mà cơ sở dữ liệu hiểu được. Các định dạng có thể là XML, JSON, Binary JSON(BSON). Hầu hết các ứng dụng sử dụng JSON (hoặc BSON) hoặc XML. Đây đều là những định dạng được sử dụng rất phổ biến và con người có thể đọc được.
- Cơ sở dữ liệu hiểu được định dạng của dữ liệu thì nó có thể thực hiện thao tác trên dữ liệu này phía máy chủ và dễ dàng hơn để viết các công cụ quản lý dữ liệu vì có thể hiển thị và chỉnh sửa dữ liệu.
- Document database có lược đồ tùy ý. Chúng ta không cần phải định nghĩa trước lược đồ và tuân thủ theo lược đồ này. Điều này cho phép chúng ta lưu trữ dữ liệu phức tạp tùy ý. Có thể lưu trữ dữ liệu dạng cây, tập hợp hay dạng từ điển một cách dễ dàng.
- Lợi ích chính của việc sử dụng document database là bên cạnh lợi ích của key-value store, chúng ta không bị giới hạn bởi việc truy vấn theo khóa.
- Document không hỗ trợ mối quan hệ. Điều đó có nghĩa là mỗi document là độc lập và chúng ta sẽ dễ dàng phân tán dữ liệu hơn so với RDBMS bởi vì chúng ta không cần lưu trữ tất cả các quan hệ trên cùng một mảnh của hệ thống và không cần hỗ trợ phép join trên hệ thống phân tán.

3.6 Tìm hiểu một số loại NOSQL phổ biến

3.6.1 RavenDB

RavenDB được viết trên C# bởi Hibernating Rhinos với giấy phép GNU AGPL v3.0. RavenDB là một giải pháp NoSQL trên nền tảng .NET được xây dựng dựa trên kiến trúc client-server. Dữ liệu được lưu trữ trên một thực thể máy chủ và những yêu cầu dữ liệu được gửi tới máy chủ này từ một hoặc nhiều máy người dùng khác nhau.



Hình 3.6: Kiến trúc client-server

Những yêu cầu gửi tới máy chủ được thực hiện bằng cách sử dụng những Client API có sẵn trong bất kỳ ứng dụng .NET hoặc ứng dụng SilverLight, hoặc bằng cách truy cập trực tiếp tới Server's RESTful API. Nếu là một .NET developer thì sử dụng .NET Client API là cách dễ nhất để làm việc với RavenDB vì nó cung cấp một lượng lớn các tính năng và nhiều API hỗ trợ. RESTful API làm cho RavenDB có thể được truy cập từ nhiều nền tảng khác nhau như truy vấn AJAX trong trang web hoặc là các ứng dụng Non-Windows được viết bằng Ruby-on-Rail.

Các đặc điểm chính của RavenDB:

- Mặc định an toàn dữ liệu: Hỗ trợ ACID (Atomicity, Consistency, Isolation, Durability), No locking, Automatic batching, client/server chatter projection.
- .Net client API: hỗ trợ tốt cho việc lập trình trên nền tảng .NET
- REST API: Tất cả dữ liệu đều có một địa chỉ duy nhất được lấy qua HTTP. Giao thức REST sử dụng các phương thức của HTTP như GET, POST, PUT và DELETE.
- Dễ dàng triển khai ứng dụng một cách nhanh chóng (chưa đến 5 phút)
- Kiến trúc phân tán: mở rộng ứng dụng một cách dễ dàng bằng cách sử dụng tính năng mạnh mẽ của RavenDB cho việc mở rộng là Sharding và Replication. Có thể kết hợp cả hai tính năng này trong cùng ứng dụng. Có hỗ trợ multi-database.

- Hỗ trợ nhiều gói tiện ích hữu dụng như: Versioning, Expiration, IndexReplication, Authorization, Authentication. Chúng ta có thể tự viết các gói mở rộng cho RavenDB bằng cách sử dụng Triggers và Responders.

3.6.2 Hadoop

Hadoop là một framework nguồn mở viết bằng Java cho phép phát triển các ứng dụng phân tán có cường độ dữ liệu lớn một cách miễn phí. Nó cho phép các ứng dụng có thể làm việc với hàng ngàn node khác nhau và hàng petabyte dữ liệu. Hadoop lấy được phát triển dựa trên ý tưởng từ các công bố của Google về mô hình MapReduce và hệ thống file phân tán Google File System (GFS).

Map/Reduce là mô hình mà ứng dụng sẽ được chia nhỏ ra thành nhiều phần đoạn khác nhau, và các phần này sẽ được chạy song song trên nhiều node khác nhau. Thêm vào đó, Hadoop cung cấp 1 hệ thống file phân tán (HDFS) cho phép lưu trữ dữ liệu lên trên nhiều node. Cả Map/Reduce và HDFS đều được thiết kế sao cho framework sẽ tự động quản lý được các lỗi, các hư hỏng về phần cứng của các node. Hadoop giúp các nhà phát triển ứng dụng phân tán tập trung tối đa vào phần logic của ứng dụng (phần chi tiết kỹ thuật phân tán bên dưới do Hadoop tự động quản lý).

3.6.3 Cassandra

Cassandra là một hệ quản trị cơ sở dữ liệu nguồn mở, được viết bằng Java với mục tiêu chính là trở thành “Best of BigTable”. Cassandra được thiết kế với khả năng xử lý một khối dữ liệu cực lớn được trải ra trên rất nhiều máy chủ trong khi cung cấp một dịch vụ có tính sẵn sàng cao và không hỏng. Nó là một giải pháp NoSQL bước đầu được phát triển bởi Facebook.

Cassandra được dùng tốt nhất khi bạn ghi nhiều hơn bạn đọc, ví dụ ở đây là hệ thống logging nhiều như các mạng xã hội, hệ thống ngân hàng, tài chính chứng khoán. Với tốc độ ghi nhanh hơn tốc độ đọc, nó thích hợp cho việc phân tích dữ liệu thời gian thực.

Các đặc điểm nổi bật:

- Tính phân cấp: Mỗi node trong một cụm có cùng một luật. Dữ liệu được phân tán dọc theo các cụm đó, nhưng không có master bởi mỗi một node có thể phục vụ bất kỳ một yêu cầu nào.

- Hỗ trợ nhân bản và nhân bản nhiều trung tâm dữ liệu: Cassandra được thiết kế cho các hệ thống phân tán, có thể triển khai một số lượng lớn các node trên nhiều trung tâm dữ liệu khác nhau. Kiến trúc phân phối các đặc trưng khóa của Cassandra thích hợp cho việc triển khai nhiều tập dữ liệu.
- Tính đàn hồi: Thông lượng đọc và ghi đều tăng tuyến tính khi các máy mới thêm vào vì giảm được thời gian chết hoặc bị gián đoạn giữa các ứng dụng
- Tính dung lỗi: Dữ liệu được nhân bản ra thành nhiều node cho khả năng dung lỗi.
- Tính điều hướng nhất quán: Đọc và ghi đưa ra một yêu cầu về tính nhất quán với việc "việc ghi không bao giờ bị lỗi".
- Hỗ trợ Map/Reduce: Cassandra có tích hợp thêm cả Hadoop đồng nghĩa với việc hỗ trợ map/reduce.
- Có truy vấn theo ngôn ngữ riêng: CQL (viết tắt của Cassandra Query Language) là một thay thế của SQL.

3.6.4 MongoDB

Mongo là một cơ sở dữ liệu NoSQL nguồn mở, hiệu năng cao, có tính mở rộng cao. Được viết bằng C++ . Dùng cách lưu trữ BSON (Json được biên dịch) với giấy phép AGPL. Thay vì lưu trữ dữ liệu theo các bảng như các cơ sở dữ liệu cổ điển. MongoDB lưu cấu trúc dữ liệu thành các văn bản dựa JSON với mô hình động (gọi đó là BSON), khiến cho việc tích hợp dữ liệu cho các ứng dụng trở nên dễ dàng và nhanh hơn. Với mục tiêu là kết hợp các điểm mạnh của mô hình key-values (nhanh mà tính mở rộng cao) với mô hình dữ liệu quan hệ (giàu chức năng).

MongoDB được sử dụng tốt nhất với nhu cầu cần truy vấn động, nếu bạn muốn định nghĩa chỉ mục mà không cần các hàm map/reduce. Đặc biệt nếu bạn cần tốc độ nhanh cho một cơ sở dữ liệu lớn vì MongoDB ngoài tốc độ đọc nhanh ra thì tốc độ ghi của nó rất nhanh.

Các đặc điểm chính của mongoDB là:

- Đánh chỉ mục: các trường trong MongoDB đều được đánh chỉ mục.
- Replication: Mongo hỗ trợ mô hình Master-slave.

- Cân bằng tải: Mongo mở rộng theo chiều ngang bằng cách sử dụng Sharding. Dữ liệu sẽ được tách thành các khoảng dựa vào khóa và phân tán dọc theo các Shard.
- Lưu trữ file: Mongo lưu trữ bằng file hệ thống, rất tốt cho việc cân bằng tải và nhân bản dữ liệu.

3.6.5 CouchDB

CouchDB được viết bằng Erlang với mục tiêu là tạo ra một cơ sở dữ liệu bền vững, chịu lỗi cao, dễ dàng trong việc sử dụng. Dạng cách lưu trữ thông thường là JSON với giấy phép Apache 2.0. Với CouchDB thì mỗi một cơ sở dữ liệu là một tập các văn bản riêng biệt. Trên mỗi văn bản còn bao gồm các thông tin về phiên bản, khiến cho việc dễ dàng đồng bộ các dữ liệu với nhau khi cơ sở dữ liệu bị mất kết nối một thời gian giữa các thiết bị.

CouchDB sử dụng MVCC (multi-Version Concurrency Control) để tránh việc deadlock cơ sở dữ liệu trong suốt quá trình ghi. Tức là trong khi ghi dữ liệu, chúng ta vẫn có thể đọc dữ liệu vì CouchDB sinh ra một bản copy và chúng ta đọc trên bản copy đó. Sau khi ghi xong nó sẽ tiến hành nhập dữ liệu giữa các thiết bị và xóa bản ghi cũ đi.

Các đặc điểm chính của CouchDB:

- Lưu trữ theo hướng văn bản (document storage) .
- Sử dụng ngữ nghĩa ACID: Cho phép điều khiển việc đồng bộ việc ghi và đọc cường độ rất cao mà không lo bị xung đột.
- Sử dụng Map/Reduce và các chỉ mục để truy vấn dữ liệu.
- Kiến trúc phân tán có nhân bản: CouchDB được thiết kế với khả năng nhân bản 2 chiều với các dữ liệu offline. Tức là ta có thể chỉnh sửa dữ liệu offline và sau đó đồng bộ chúng sau khi có kết nối trở lại.
- REST API: Tất cả dữ liệu đều có một địa chỉ duy nhất được lấy qua HTTP. Giao thức REST sử dụng các phương thức của HTTP như GET, POST, PUT và DELETE với 4 chức năng cơ bản (Tạo, đọc, ghi, xóa, sửa)
- Built for Offline: Có khả năng nhân bản dữ liệu cho từng thiết bị và tự động đồng bộ dữ liệu khi thiết bị hoạt động trở lại.

CHƯƠNG 4 – TÌM HIỂU VỀ RAVENDB

4.1 Tại sao chọn RavenDB

RavenDB là một document database nên nó thừa hưởng những lợi ích to lớn của cơ sở dữ liệu NoSQL nói chung và cơ sở dữ liệu hướng tài liệu nói riêng. Những lợi ích to lớn này chúng ta đã đề cập ở những phần trên. Ngoài ra, RavenDB còn có những đặc điểm, tính năng nổi bật khác như sau:

- RavenDB là một cơ sở dữ liệu hướng tài liệu mã nguồn mở có hỗ trợ transactional được viết cho nền tảng .NET. RavenDB đưa ra mô hình dữ liệu linh hoạt nhằm đáp ứng yêu cầu của các hệ thống thể giới thực. RavenDB cho phép xây dựng những ứng dụng có hiệu suất cao, độ trễ thấp một cách nhanh chóng và hiệu quả.
- Dữ liệu trong RavenDB được lưu trữ dưới dạng JSON, phi lược đồ (scheme-less) và có thể truy vấn hiệu quả bằng cách sử dụng truy vấn Linq từ đoạn mã .NET hay sử dụng các RESTful API. RavenDB sử dụng “Index” (sẽ nói rõ hơn ở phần tiếp theo) để truy vấn dữ liệu một cách nhanh chóng.
- RavenDB thích hợp để xây dựng các ứng dụng web-scale (các ứng dụng web có khả năng mở rộng lớn). RavenDB còn hỗ trợ replication (nhân bản) và sharding (phân tán dữ liệu).
- Xây dựng ứng dụng trên cơ sở hạ tầng đã có nhằm mở rộng đáng kể kích thước của ứng dụng (RavenDB có thể lưu trữ đến 16 terrabytes trên một máy đơn).
- Chạy và làm việc tốt trên môi trường Windows. So với CouchDB thì muốn chạy CouchDB trên Windows, ta cần phải biên dịch từ Erlang source code.
- RavenDB không chỉ là Server. Có thể nhúng RavenDB vào trong ứng dụng.
- Hỗ trợ System.Transaction và có thể thực hiện các transactions trong hệ thống phân tán.
- Hỗ trợ thao tác map/reduce trên các documents dựa vào truy vấn Linq
- Hỗ trợ đầy đủ .NET client API, thực hiện mẫu “Unit Of Work”, theo dõi sự thay đổi, tối ưu hóa thao tác đọc/ ghi, và nhiều gói dữ liệu khác.

- Có công cụ quản lý (Raven Studio Management) giao diện web trực quan, có thể xem, thao tác và truy vấn dữ liệu.
- Có thể mở rộng bằng cách viết các plugins Managed Extensibility Framework.
- Hỗ trợ “partial document update” có nghĩa là không cần phải gửi toàn bộ dữ liệu của các document theo yêu cầu, chỉ gửi những dữ liệu cần thiết.

4.2 Lý thuyết cơ bản RavenDB

4.2.1 RavenDB server

Một số cách để chạy RavenDB server:

- Chạy ứng dụng console Raven.Server.exe (tại thư mục /Server/)
- Chạy RavenDB như là một dịch vụ (service)
- Tích hợp RavenDB với IIS trên máy chủ dựa trên Windows của bạn
- Nhúng vào ứng dụng

Để bắt đầu thì bạn cần tải gói chương trình về, giải nén, và chạy file Server/Raven.Server.exe. Bạn sẽ thấy màn hình như thế này:

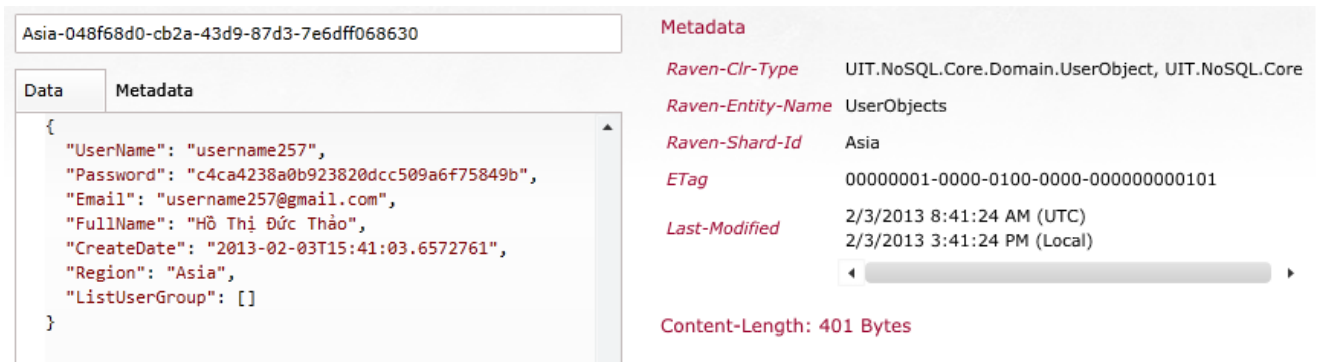


Hình 4.1: RavenDB server

4.2.2 Documents, Collections và Document xác định duy nhất:

Một thực thể dữ liệu duy nhất trong RavenDB được gọi là một document (tài liệu) và tất cả các tài liệu được lưu trữ trong RavenDB như các tài liệu JSON. Định

dạng JSON được lựa chọn vì nó có thể lưu trữ phân cấp, con người có thể đọc được. Mọi document đều có siêu dữ liệu (metadata), theo mặc định nó chỉ chứa dữ liệu được sử dụng trong RavenDB (ví dụ thuộc tính Raven-Entity-Name lưu trữ loại thực thể cho tài liệu). Hình ảnh của một document như hình sau:

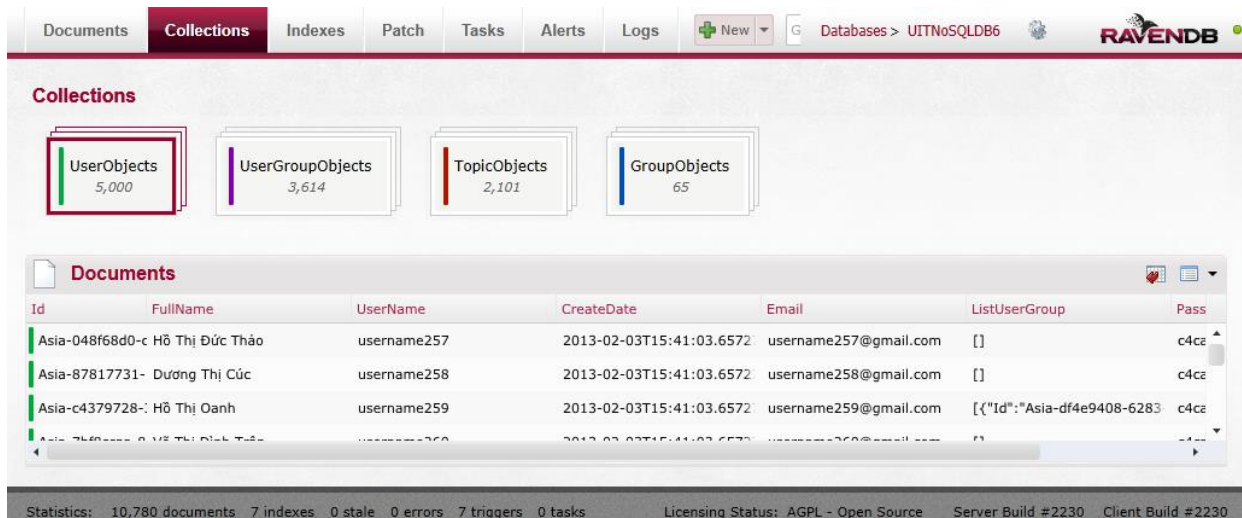


Hình 4.2: Document trong RavenDB

Collections là một tập hợp các tài liệu cùng loại thực thể trong RavenDB. Nó không phải là một “bảng” (table) trong cơ sở dữ liệu. Collection là một cấu trúc hoàn toàn ảo, không có ý nghĩa vật lý đối với cơ sở dữ liệu.

Với RavenDB mỗi document có một ID riêng và duy nhất, nếu chúng ta lưu trữ hai thực thể khác nhau với cùng một id (ví dụ như *users/1*) – bản ghi thứ hai sẽ ghi đè lên bản ghi đầu tiên. Quy ước trong RavenDB : id được kết hợp từ tên collection và id duy nhất của tài liệu trong collection (ví dụ *users/1*). Tuy nhiên, Document ID không phụ thuộc vào loại thực thể, do đó không bắt buộc phải chứa tên collection chứa nó.

4.2.3 The Management Studio



Hình 4.3: Management studio

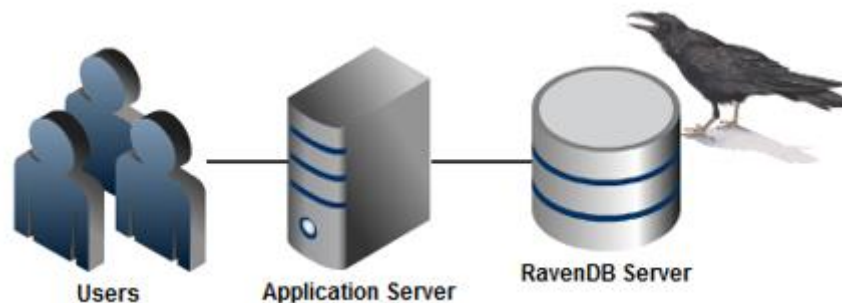
Tất cả máy chủ được quản lý thông qua ứng dụng Silverlight gọi là Management Studio. Trình quản lý này có thể được truy cập bằng cách duyệt đến địa chỉ và cổng lắng nghe (mặc định là <http://localhost:8080>).

4.3 .NET client API

4.3.1 Giới thiệu .NET client API

Khi sử dụng RavenDB với chế độ server, embedded hay remote, client API cho phép developer dễ dàng truy cập đến RavenDB từ bất kỳ ngôn ngữ .NET nào. Client API hiện thực mẫu “Unit of work”, áp dụng quy tắc cho các tiến trình lưu trữ, nạp dữ liệu, tích hợp System.Transaction, gửi một tập yêu cầu đến server, lưu dữ liệu ở bộ nhớ đệm (caching)...

Kiến trúc của hệ thống khi sử dụng .NET client API như hình sau:



Hình 4.4 : Kiến trúc hệ thống với .NET client API

4.3.2 Nguyên tắc thiết kế .NET client API

API bao gồm 2 lớp chính:

- **IDocumentSession**: Document Session dùng để thao tác với cơ sở dữ liệu, load dữ liệu từ cơ sở dữ liệu, truy vấn dữ liệu, lưu trữ và xóa dữ liệu. Đối tượng Session tạo ra tốn rất ít chi phí và là tiến trình không an toàn. Một thực thể của Interface hiện thực mẫu “Unit of Work”, theo dõi sự thay đổi cũng như nhiều tính năng khác đề cập ở trên như quản lý Transaction. Khi sử dụng .NET client API, hầu hết các thao tác với cơ sở dữ liệu đều thông qua đối tượng Session.
- **IDocumentStore**: Là một Session Factory và việc tạo DocumentStore thì tốn nhiều chi phí, là tiến trình an toàn và được tạo 1 lần cho mỗi ứng dụng.

Document Store chịu trách nhiệm thực sự cho các giao tiếp giữa client và server, nắm giữ các quy ước liên quan đến saving/loading dữ liệu và nhiều cấu hình cho ứng dụng, ví dụ như là http cache cho server.

4.3.3 Kết nối tới RavenDB data store

Việc tạo một thực thể document store thì tốn chi phí nhưng là tiến trình an toàn. Vì vậy ta nên tạo 1 documentstore/1 database/1 ứng dụng.

```
var documentStore = new DocumentStore { Url = "http://myravendb.mydomain.com/" };
documentStore.Initialize();
```

"http://myravendb.mydomain.com/" là địa chỉ của RavenDB server

4.3.4 Những thao tác cơ bản với cơ sở dữ liệu

Trong phần này chúng tôi sẽ giới thiệu cách tạo đối tượng session, các thao tác insert/ load/ update / delete và truy vấn dữ liệu với RavenDB. Chúng ta sẽ sử dụng class User sau đây làm ví dụ cho các phần bên dưới:

```
public class User
{
    public string Id { get; set; }
    public string Username { get; set; }
    public string Password { get; set; }
}
```

4.3.4.1 Đối tượng Session

Đối tượng Session được tạo ra từ Document Store và ta dùng nó để thực hiện thao tác tới cơ sở dữ liệu. Lưu ý: khi phương thức SaveChanges() được gọi thì mới thực sự thực hiện thao tác xuống cơ sở dữ liệu:

```
using (var session = documentStore.OpenSession())
{
    var user = new User { Username = username, Password = "password" };
    session.Store(user);
    session.SaveChanges();
}
```

4.3.4.2 Insert document

Để lưu một User xuống cơ sở dữ liệu, ta tạo một mới một user:

```
// tạo thực thể mới của lớp User
User user = new User() { Username = "username", Password = "password" };
```

Lưu user vừa tạo bằng cách gọi hàm Store() và SaveChanges()

```
// lưu dữ liệu xuống RavenDB
session.Store(user);
session.SaveChanges();
```

4.3.4.3 Load và update document

Mỗi document được lưu trữ như là một phần tử của collection. Collection là một tập hợp các document cùng loại. Chúng ta lấy document nhờ vào id của nó:

```
// Users/1 là một thực thể của collection User với Id là 1
User user = session.Load<User>("Users/1");
```

Muốn thay đổi thông tin của đối tượng ta chỉ cần làm như sau:

```
user.Password = "New password";
```

Lưu lại những thay đổi này xuống cơ sở dữ liệu bằng cách gọi:

```
session.SaveChanges();

// chúng ta không cần gọi phương thức Update() hay theo dõi sự thay đổi của đối tượng.
// RavenDB làm điều đó cho chúng ta.
```

4.3.4.4 Delete document

- **Xóa bằng cách tham chiếu đến đối tượng:** Khi ta lấy được document dùng hàm load() thì chúng ta có thể xóa document thông qua hàm delete():

```
session.Delete(user);
session.SaveChanges();
```

- **Xóa dựa vào khóa:** Dùng DatabaseCommands

```
session.Advanced.DatabaseCommands.Delete("Users/1", null);
```

4.3.4.5 Truy vấn dữ liệu trong RavenDB

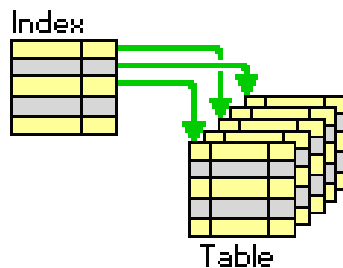
Chúng ta sử dụng cú pháp Linq để truy vấn dữ liệu nhanh chóng.

```
var result = from user in session.Query<User>()
              where user.Username == "NguyenVanA"
              select user;
```

4.3.5 Sử dụng Index để truy vấn dữ liệu

Để đáp ứng yêu cầu của người dùng trong khoảng thời gian rất nhanh, máy chủ RavenDB đánh chỉ mục dữ liệu bên dưới background bất cứ khi nào một document được thêm mới hay cập nhật dữ liệu. Dữ liệu kết quả được lưu trên đĩa cứng.

Cách làm giống như đánh mục lục trong tài liệu. RavenDB sẽ tự động tạo index dựa vào id của document sau đó tổng hợp lại thành một tập hợp. Mỗi phần tử sẽ chứa một id ánh xạ với một document duy nhất như hình minh họa bên dưới:



Hình 4.5: Index

Tất cả indexes trong RavenDB đều dựa trên Lucene Lucene là một “full text search engine library”) và chúng ta tận dụng những ưu điểm này để cung cấp hệ thống truy vấn một cách nhanh chóng, đầy đủ tính năng và linh hoạt. RavenDB cho phép sử dụng cú pháp Lucene để truy vấn cho dù chúng được gửi từ Client API thông qua Linq provider hoặc thông qua HTTP RESTful API. Chúng được chuyển sang truy vấn Lucene và thực thi dựa vào những index thích hợp.

Một điều quan trọng cần lưu ý là tất cả truy vấn tới RavenDB server đều sử dụng index để trả về kết quả. Chúng ta có thể tự định nghĩa index riêng, nếu không thì RavenDB sẽ tạo tự động index.

Có 2 loại index trong RavenDB:

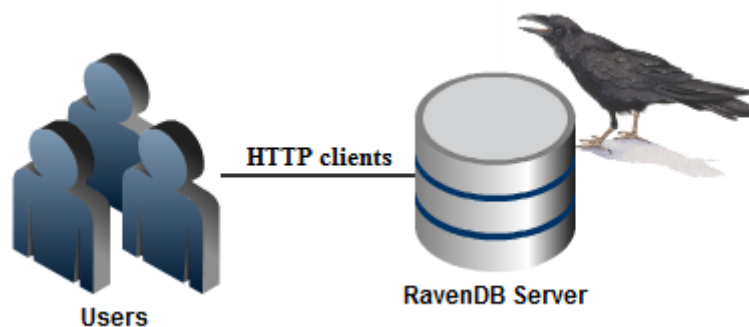
- Static index là index mà do chính người dùng định nghĩa.

- Dynamic index là index được RavenDB tự động tạo ra dựa trên truy vấn của người dùng nếu không có truy vấn nào khớp với yêu cầu. RavenDB sẽ tìm index thích hợp để truy vấn và tạo index kèm với truy vấn nếu nó không tồn tại.

Một khái niệm cũng cần nói đến là khái niệm “stale index”. Bởi vì phương pháp tiếp cận của RavenDB là “better stale than offline” – truy vấn index có thể trả về kết quả cũ. Ví dụ, khi người dùng truy vấn dữ liệu trong khi có quá trình cập nhật một lượng lớn dữ liệu. RavenDB sẽ thông báo cho người dùng biết được nếu kết quả đó là cũ và cũng có thể cho biết phải chờ đợi cho đến khi kết quả mới (non-stale result).

4.4 Tổng quan HTTP API

RavenDB hỗ trợ HTTP API cho việc truy cập và thao tác dữ liệu trên máy chủ. HTTP API cung cấp hầu hết các chức năng tương tự C# .NET client API, nhưng với platform agnostic (đa nền tảng) và giao diện web thân thiện. Sử dụng HTTP API chúng ta có thể viết được ứng dụng RavenDB với đầy đủ chức năng chỉ cần sử dụng Javascript và HTML. Cách giao tiếp rất đơn giản và minh họa như hình sau:



Hình 4.6 : Kiến trúc hệ thống với HTTP API

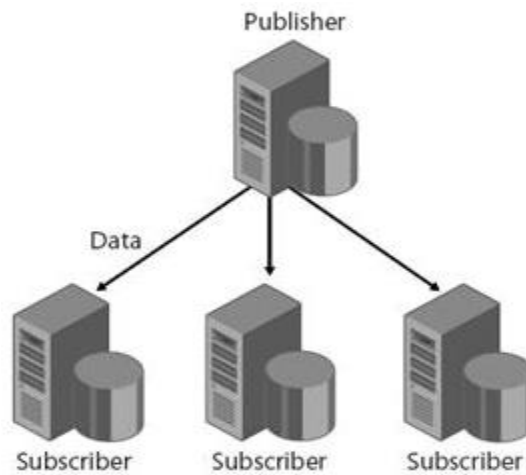
Là một phần của web thân thiện, HTTP API hiểu được những nguyên tắc chung RESTful. Ví dụ, document database là những tài nguyên thông qua những địa chỉ URLs duy nhất và những nguồn tài nguyên có thể thực thi bằng cách sử dụng các động từ đặc trưng của HTTP như: GET, PUT, POST và DELETE. RESTful là mục đích của HTTP API nhưng chỉ là mục đích thứ yếu so với mục đích dễ dàng sử dụng những tính năng mạnh mẽ như batching và multi-document transactions.

4.5 Mở rộng hệ thống theo chiều ngang (scaling out)

RavenDB hỗ trợ sẵn 2 gói mở rộng là Replication và Sharding. 2 tính năng này có thể sử dụng kết hợp với nhau.

4.5.1 Replication

Replication là kỹ thuật nhân bản dữ liệu từ một nguồn dữ liệu gốc ra một hoặc nhiều bản.



Hình 4.7 : Kiến trúc của Replication

Sử dụng Replication sẽ có những tác dụng trên hệ thống của chúng ta:

- Theo dõi trên máy chủ những document được viết lên lần đầu (bản gốc). Gói nhân bản sử dụng thông tin này để xác định một document được nhân bản có bị xung đột với document đã tồn tại hay không.
- Những document gặp phải lỗi xung đột sẽ được đánh dấu và được xử lý tự động hoặc có sự tham gia của người dùng.
- Những document sẽ được xóa khi bị đánh dấu và gói nhân bản cũng dùng dấu hiệu này để xóa những document có liên quan. Điều này được thực hiện mà không cần lưu ý cho máy khách.
- Gói nhân bản sẽ không sao chép một số tài liệu hệ thống (có key bắt đầu bằng Raven/)

Gói nhân bản sẽ tạo ra một số tài liệu hệ thống. **Raven/Replication/Destinations** là danh sách máy chủ chúng ta cần nhân bản và **Raven/Replication/Sources/[server]** – thông tin về dữ liệu được nhân bản từ một máy chủ cụ thể.

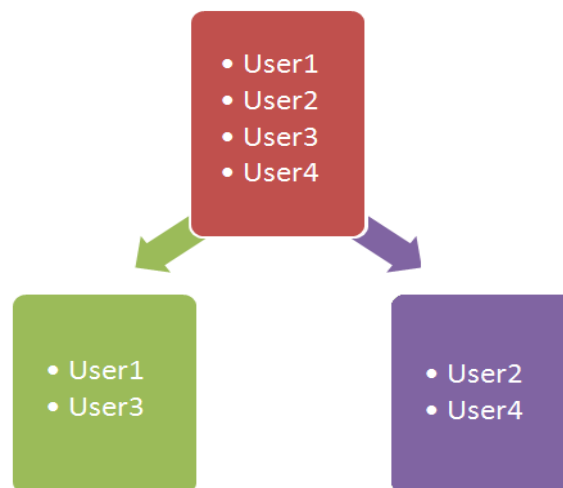
Máy chủ nào cần được nhân bản tài liệu sẽ tạo ra một document với id là **Raven/Replication/Destinations** và có định dạng như sau:

```
{
  "Destinations" : [
    { "Url" : "http://raven_two:8080/" },
    { "Url" : "http://raven_three:8080/" },
  ]
}
// dữ liệu sẽ được nhân bản đến những máy chủ : "Url": "http://raven_two:8080/",
"Url": "http://raven_three:8080/"
```

4.5.2 Sharding

RavenDB hỗ trợ sẵn sharding. Sharding là cách phân tán dữ liệu trên nhiều máy chủ khác nhau. Vì thế mỗi máy chủ sẽ lưu trữ một phần dữ liệu. Điều này là cần thiết trong trường hợp chúng ta cần xử lý một lượng lớn dữ liệu.

Kiến trúc hệ thống khi áp dụng kỹ thuật Sharding như hình dưới đây:



Hình 4.8 : Kiến trúc của Sharding

Giả sử trong một ứng dụng cần xử lý dữ liệu từ rất nhiều công ty trên khắp thế giới thì một sự lựa chọn sẽ là lưu trữ dữ liệu của một công ty trên một mảnh của hệ thống (gọi là shard) và việc lưu trữ này phụ thuộc vào vị trí khu vực (Region) của công ty. Ví dụ, những công ty nằm ở châu Á (Asia) sẽ được lưu trữ trên một mảnh, những công ty nằm ở Trung Đông (Middle East) sẽ được lưu trữ trên một mảnh khác và những công ty từ Mỹ sẽ được lưu trữ vào một mảnh thứ ba.

Ý tưởng trên là xác định vị trí địa lý của mảnh gần nơi mà dữ liệu được sử dụng. Do đó các công ty ở châu Á sẽ được phục vụ từ một máy chủ gần đó và phản hồi nhanh hơn cho người dùng nằm ở châu Á. Điều này cũng làm giảm tải trên mỗi máy chủ bởi vì nó chỉ xử lý một số phần của dữ liệu. Dưới đây là một số thực thể có thể được chia vào các mảnh khác nhau dựa trên khu vực của nó: Company và Invoice

```
public class Company
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Region { get; set; }
}
public class Invoice
{
    public string Id { get; set; }
    public string CompanyId { get; set; }
    public decimal Amount { get; set; }
}
```

Khi sử dụng sharding, chúng ta sử dụng ShardedDocumentStore thay cho DocumentStore thường (chỉ khác nhau ở giai đoạn khởi tạo). Để tạo một ShardedDocumentStore chúng ta cung cấp một thể hiện của ShardStrategy – chứa một từ điển những mảnh hoạt động trên đó. Các khóa và giá trị trong từ điển là ID của mảnh (shardID) và một thể hiện DocumentStore trên mảnh đó. ShardStrategy thông báo cho ShardedDocumentStore biết làm thế nào để tương tác với những mảnh của hệ thống. Đơn giản nhất là chúng ta sử dụng ShardStrategy với thiết lập mặc định:

```
var shards = new Dictionary<string, IDocumentStore>
{
    { "Asia ", new DocumentStore {Url = "http://localhost:8080"}},
    { "Middle East ", new DocumentStore {Url = "http://localhost:8081"}},
    { "America ", new DocumentStore {Url = "http://localhost:8082"}},
};
var shardStrategy = new ShardStrategy(shards)
```

```
.ShardingOn<Company>(company => company.Region)

.ShardingOn<Invoice>(x => x.CompanyId);

var documentStore = new ShardedDocumentStore(shardStrategy).Initialize();
```

Sử dụng phương thức `ShardingOn` trên đối tượng `ShardStrategy` để thiết lập thuộc tính nào lưu giữ thông tin shard id của một thực thể cụ thể. `Company` sẽ giữ shard id trong thuộc tính `Region` và `Invoice` giữ shard id trong thuộc tính `CompanyId`. Giờ thì chúng ta có thể lưu trữ dữ liệu trên nhiều mảnh khác nhau:

```
using (var session = documentStore.OpenSession())
{
    var asian = new Company { Name = "Company 1", Region = "Asia" };
    session.Store(asian);

    var middleEastern = new Company { Name = "Company 2", Region = "Middle-East" };
    session.Store(middleEastern);

    var american = new Company { Name = "Company 3", Region = "America" };
    session.Store(american);

    session.Store(new Invoice { CompanyId = american.Id, Amount = 3 });
    session.Store(new Invoice { CompanyId = asian.Id, Amount = 5 });
    session.Store(new Invoice { CompanyId = middleEastern.Id, Amount = 12 });
    session.SaveChanges();
}
```

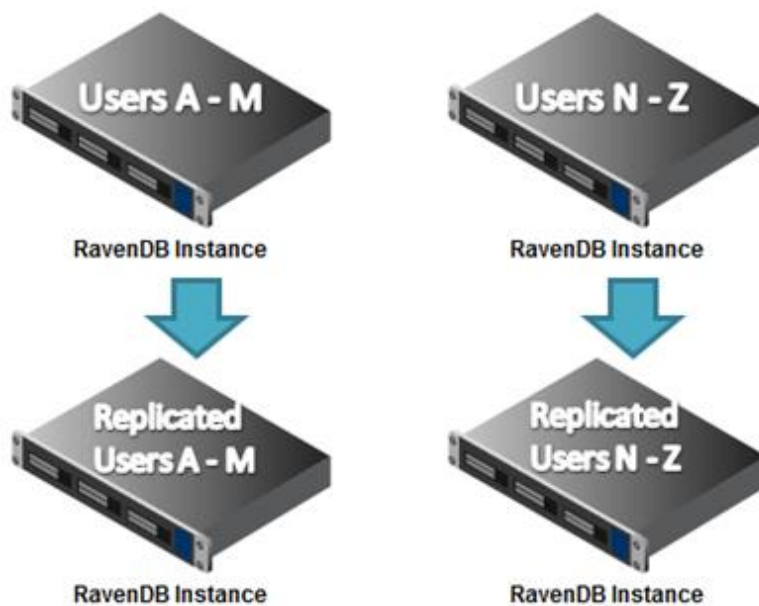
Trong ví dụ trên chúng ta đã lưu trữ mỗi `Company` trên những mảnh khác nhau, và mỗi `Invoice` được lưu trữ trên cùng mảnh với `company` của nó. Chúng ta có thể thực hiện thao tác như `Query`, `Load` hay `LuceneQuery`.

```
using (var session = documentStore.OpenSession())
{
    var allCompanies = session.Query<Company>()
        .Customize(x => x.WaitForNonStaleResultsAsOfNow())
        .Where(company => company.Region == "Asia") // chỉ gửi yêu cầu tới mảnh Asia
        .ToArray();
}
```

4.5.3 Kết hợp Replication và Sharding

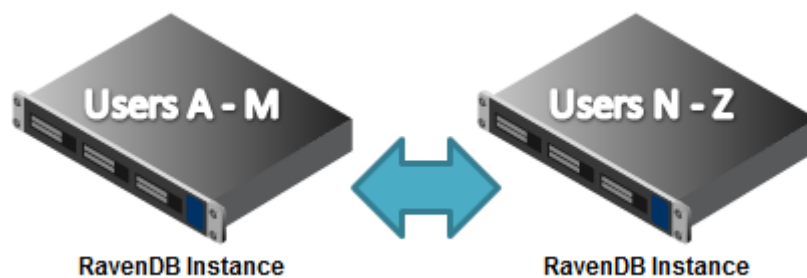
Replication và Sharding là 2 tính năng mạnh mẽ của RavenDB. Chúng ta có thể sử dụng kết hợp Replication với Sharding. Giả sử chúng ta chỉ lưu trữ dữ liệu Users trong RavenDB và chúng ta phân tán dữ liệu trên 2 node dựa vào tên người dùng (người dùng tên bắt đầu là A-M trên một node và N-Z trên node thứ 2).

Phân tán với các node chuyển đổi dự phòng chuyên dụng: Trong thiết lập này, chúng ta có hai node để lưu trữ thông tin người dùng và hai node phục vụ (slave) cho mỗi node chính (master). Nếu một trong các node chính bị hư hỏng thì Raven tự động chuyển đổi sang các bản sao dự phòng.



Hình 4.9: Phân tán với các nút chuyển đổi dự phòng chuyên dụng

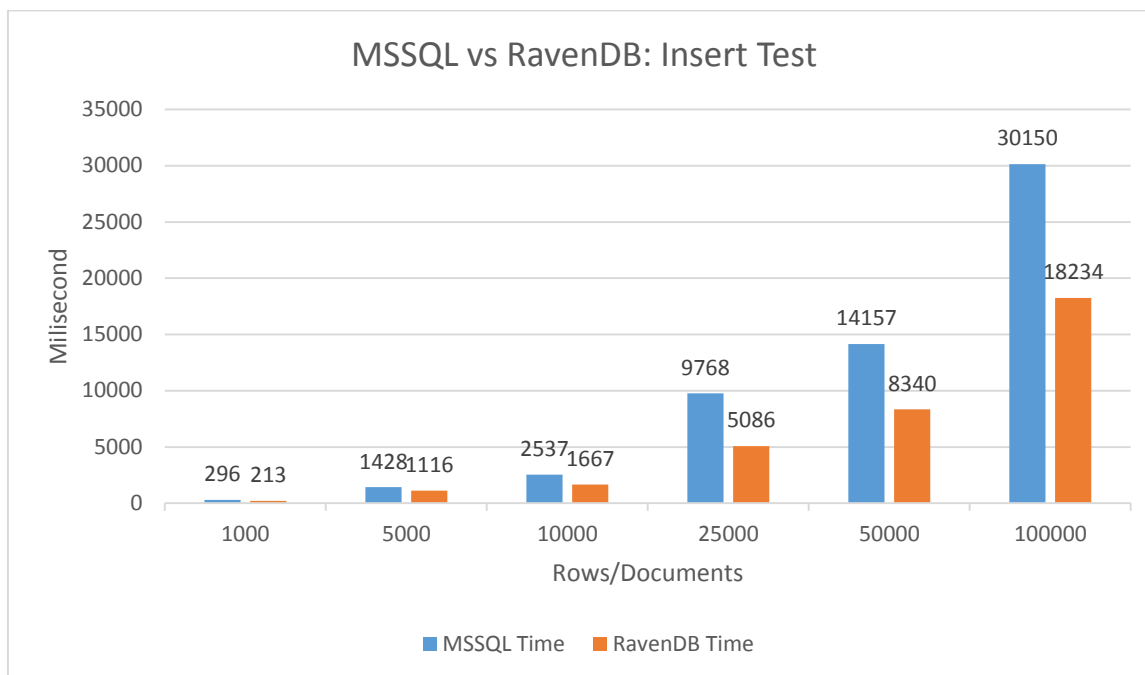
Phân tán với node dự phòng nội bộ: Một lựa chọn khác là sử dụng sharding chủ yếu như một phương tiện giảm tải trên các máy chủ và thiết lập nhân bản giữa các node khác nhau mà không có các node chuyển đổi dự phòng chuyên dụng.



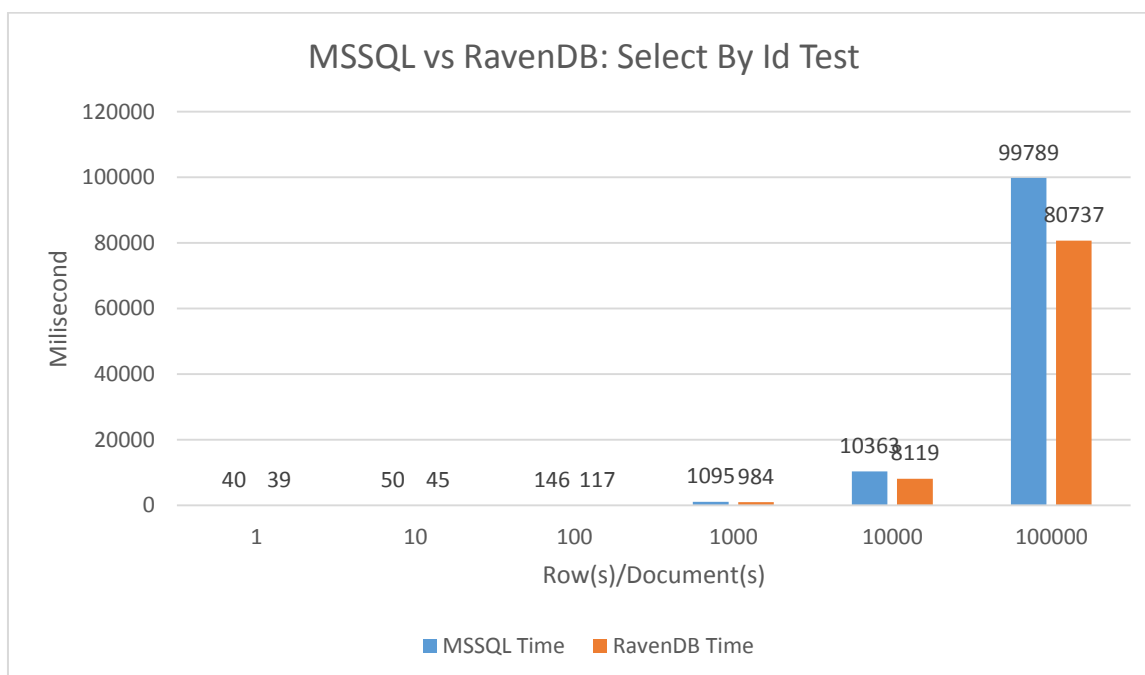
Hình 4.10: Phân tán với các nút chuyển đổi dự phòng nội bộ

4.6 So sánh hiệu suất RavenDB với MSSQL Express 2012

Sau khi tìm hiểu cách làm việc, sử dụng RavenDB, nhóm đã thực hiện test hiệu suất làm việc giữa RavenDB với MSSQL phiên bản Express 2012 với kết quả sau:



Hình 4.11: Biểu đồ so sánh khả năng Insert dữ liệu của RavenDB so với MSSQL



Hình 4.12: Biểu đồ so sánh khả năng Select dữ liệu của RavenDB so với MSSQL

Qua 2 biểu đồ trên ta cũng thấy được khả năng Insert dữ liệu và Select By ID của RavenDB là khá tốt so với MSSQL Express 2012.

4.7 So sánh RavenDB với CouchDB và MongoDB

	MongoDB	CouchDB	RavenDB
Documents			
Format	BSON	JSON	JSON
Metadata	No	System	System + Custom
Versioning	No	Yes	Included Plug-in
Attachments	GridFS	Yes	Yes
Map/Reduce	JavaScript + others	JavaScript	LINQ
Bulk Load	Monogoimport utility	Yes	Yes
Adhoc Query	Yes	No	No
Storage			
Sharding	Available in 1.6	Yes	Yes
Durability	Single Server will be available in 1.8	"crash-only" design	write ahead logging and snapshot isolation for guaranteed crash recovery via ESE
Transactions	No	No	Yes
Concurrency	Update in-place	MVCC (Multi-version Concurrency Control)	Optimistic concurrency
Consistency	Strong Master / Eventual Slave	Strong Node / Eventual Cluster	Eventual
Replication	Master-Slave	Peer-based	Included Plug-in
Interface			
Interface Protocol	Custom protocol over TCP/IP	HTTP/REST	HTTP/REST
.NET API	3 rd Party Projects	3 rd Party Projects	Included
Other			
Triggers	No	Update Validation Security	Yes
Security	Basic	Basic	Basic using included plug-in
Written In	C++	Erlang	C#

Bảng 4.1: Bảng so sánh RavenDB với MongoDB và CouchDB

Qua bảng so sánh trên chúng ta thấy được những điểm mạnh của RavenDB so với CouchDB và MongDB như sau:

- RavenDB hỗ trợ transaction. Điều này có nghĩa là dữ liệu của chúng ta được lưu trữ và xử lý một cách an toàn, đáng tin cậy.
- Mở rộng với RavenDB dễ dàng hơn rất nhiều với 2 gói tính năng mạnh mẽ là: Replication và Sharding. Replication hỗ trợ cả master – slave và master – master. Việc phân tán dữ liệu vô cùng đơn giản, không yêu cầu cấu hình. Một máy đơn RavenDB có thể lưu trữ đến 16 terrabytes dữ liệu.
- Hỗ trợ rất tốt .NET client API. Đây là tính năng có sẵn trong RavenDB.
- Tối ưu hóa cho việc xử lý đồng thời của hàng ngàn người dùng trên một lượng dữ liệu cực lớn.

Ngoài những tính năng trên thì RavenDB còn có một số đặc điểm mà khi sử dụng tôi thấy thích thú, đó là:

- Việc sử dụng RavenDB rất đơn giản, tải về là có thể chạy được, không yêu cầu cài đặt, không yêu cầu cấu hình.
- Thêm thư viện vào trong ứng dụng và bắt đầu lập trình. Việc này cũng không tốn thời gian nhiều. Đối với những ai đã từng lập trình trên .NET thì việc lập trình với RavenDB vô cùng đơn giản. RavenDB hỗ trợ cú pháp LINQ để thực hiện các thao tác cơ sở dữ liệu.
- Raven Studio Management là một công cụ quản lý vô cùng hữu ích. Với giao diện web trực quan, chúng ta có thể xem, thêm, thay đổi dữ liệu một cách dễ dàng với Raven Studio Management. Ngoài ra nó còn giúp chúng ta làm nhiều việc khác như: quản lý logs, patching, tasks, alerts, ...

CHƯƠNG 5-XÂY DỰNG ỨNG DỤNG SỬ DỤNG RAVENDB

5.1 Giới thiệu về ứng dụng

Để thực hành xây dựng ứng dụng sử dụng cơ sở dữ liệu NoSQL, mà cụ thể là cơ sở dữ liệu RavenDB, chúng tôi xây dựng một website cho phép các người dùng có thể thảo luận về vấn đề nào đó (với các chức năng cơ bản như Google Group). Website sử dụng công nghệ ASP.NET MVC 4 (xem phụ lục 7.1) nhằm tận dụng các ưu điểm của mô hình phát triển web này.

Mục tiêu của ứng dụng là có được trải nghiệm về việc sử dụng cơ sở dữ liệu NoSQL, nhìn thấy được lợi ích mà cơ sở dữ liệu NoSQL mang lại so với cơ sở dữ liệu quan hệ đồng thời hiểu biết thêm một giải pháp cho vấn đề lưu trữ dữ liệu cho hệ thống.

5.2 Lý do lựa chọn ứng dụng này

Để nhìn thấy được tốc độ của cơ sở dữ liệu NoSQL, khả năng làm việc với lượng dữ liệu cực kì lớn, khả năng phân tán dữ liệu... Website này là một lựa chọn phù hợp vì nó cũng yêu cầu đến các tính năng đó: hiệu suất cao, lưu trữ dữ liệu nhiều, phân tán dữ liệu và đặc biệt là không đòi hỏi khắc khe về tính nhất quán dữ liệu.

Website này cần lưu trữ một lượng lớn dữ liệu như: thông tin người dùng, thông tin nhóm, nội dung các bài đăng và các bình luận. Website yêu cầu đọc và ghi dữ liệu rất nhiều, tuy nhiên tần suất của việc đọc diễn ra nhiều hơn việc ghi nhiều lần. Đây là một thế mạnh của RavenDB so với các cơ sở dữ liệu NoSQL cùng loại. Với thiết kế đặc biệt của RavenDB và thiết kế dữ liệu dạng document sẽ giúp ứng dụng đạt được hiệu suất hoạt động cao hơn việc sử dụng cơ sở dữ liệu RDBMS.

Website chấp nhận việc dữ liệu không nhất quán trong một khoảng thời gian ngắn trước khi được cập nhật đúng lại. Các yêu cầu cập nhật chưa cần thiết sẽ được xử lý sau, các yêu cầu cần thiết để hiển thị, thông báo dữ liệu sẽ được ưu tiên trước. Như vậy điều này cho phép ứng dụng chạy mượt mà hơn, giảm thời gian phải chờ đợi.

Ta vẫn có thể sử dụng cơ sở dữ liệu RDBMS để làm cơ sở dữ liệu cho website này. Tuy nhiên, chi phí sẽ đắt hơn do cần nhiều server hơn để có thể đạt hiệu suất ngang với NoSQL. Ngoài ra trong ứng dụng này, không cần thiết những tính chất đặc

biệt của RDBMS như: khoá ngoại, mô hình dữ liệu quan hệ... Như vậy, với giải pháp sử dụng NoSQL cho ứng dụng này đã đáp ứng được hoàn toàn yêu cầu đề ra với một chi phí thấp nhất.

5.3 Đặc tả ứng dụng

5.3.1 Yêu cầu chức năng

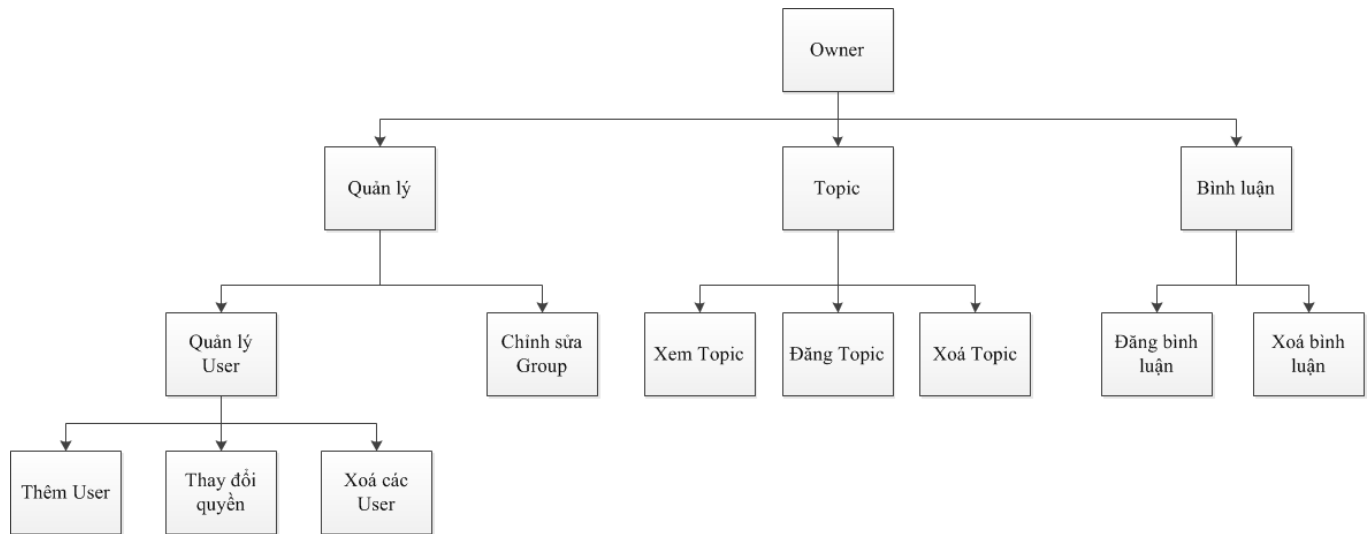
STT	Tên chức năng	Mô tả
1	Chức năng quản lý tài khoản	Cho phép người dùng tạo mới tài khoản và chỉnh sửa thông tin tài khoản. Thông tin tài khoản bao gồm: tên tài khoản, mật khẩu, tên đầy đủ, email, ngày tạo, danh sách nhóm mà tài khoản này tham gia.
2	Chức năng tạo mới nhóm	Cho phép người dùng tạo mới một nhóm để thảo luận vấn đề nào đó sau khi đã đăng nhập vào hệ thống.
3	Chức năng tạo mới bài viết	Cho phép người dùng là thành viên của nhóm có thể đăng bài viết mới vào trong nhóm
4	Chức năng đăng bình luận cho bài viết	Cho phép người dùng là thành viên của nhóm có thể đăng bình luận cho bài viết được đăng. Nếu nhóm là public thì sau khi đăng nhập vào hệ thống thì bất kỳ ai cũng có thể đăng bình luận
5	Chức năng upload file, download file	Cho phép người dùng tải lên tập tin khi đăng bài viết mới hoặc bình luận cho bài viết. Có thể tải tập tin khi bài viết hoặc bình luận đã đăng lên nhóm
6	Chức năng quản lý bài viết	Khi người dùng có quyền là owner (chủ nhóm) hoặc manager (quản lý nhóm) thì có thể xóa bài viết trong nhóm đó

7	Chức năng quản lý thành viên trong nhóm	Khi người dùng có quyền owner hoặc manager thì có thể quản lý thành viên trong nhóm của nhóm: chấp nhận hoặc từ chối yêu cầu tham gia nhóm của người dùng, có thể xóa thành viên ra khỏi nhóm. Nếu là owner có thể chuyển một thành viên(member) thành một người quản lý(manager)
8	Chức năng quản lý thông tin cá nhân của thành viên trong một nhóm cụ thể	Khi đã trở thành một thành viên của nhóm cụ thể thì thành viên này có thể cài đặt một số thông tin về nhóm này (như có nhận mail từ nhóm này khi có bài viết mới hay không...)
9	Chức năng quản lý cài đặt nhóm có là nhóm public hay không	Khi người dùng có quyền là owner hay manager thì có thể thiết lập nhóm này trở thành nhóm public hay không. Nhóm public là nhóm mà mọi người đều có thể xem bài viết và bình luận được. Nhóm private thì chỉ có thành viên trong nhóm mới được phép xem bài đăng.
10	Chức năng thông báo về bài đăng mới	Khi một nhóm bất kỳ có bài đăng mới thì mọi thành viên trong nhóm sẽ được nhận thông báo qua mail về bài đăng mới này. Trên giao diện ứng dụng, khi đăng nhập vào hệ thống thì thành viên cũng được thông báo về bài viết mới nhất của từng nhóm, số lượng bài đăng mới chưa đọc của từng bài viết.
11	Chức năng tìm kiếm	Cho phép người dùng có thể tìm kiếm nhóm theo tên nhóm, mô tả nhóm hoặc người tạo nhóm.

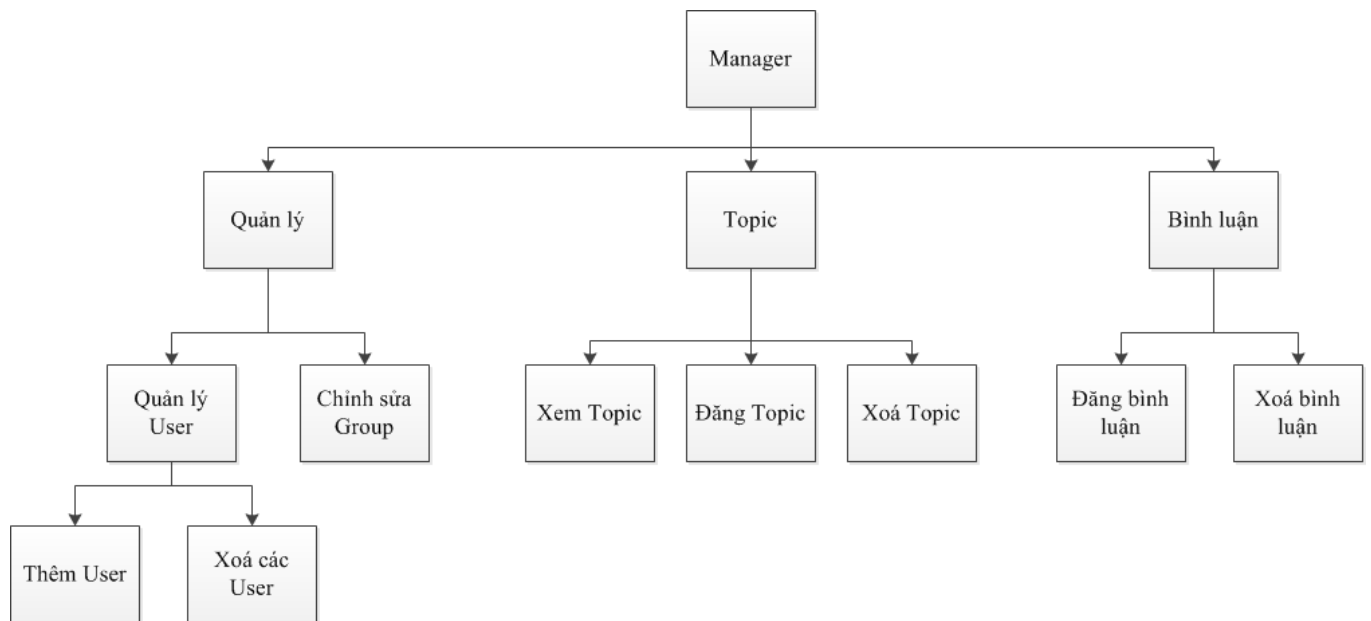
Bảng 5.1: Bảng danh sách yêu cầu chức năng của ứng dụng

5.3.2 Phân rã chức năng website

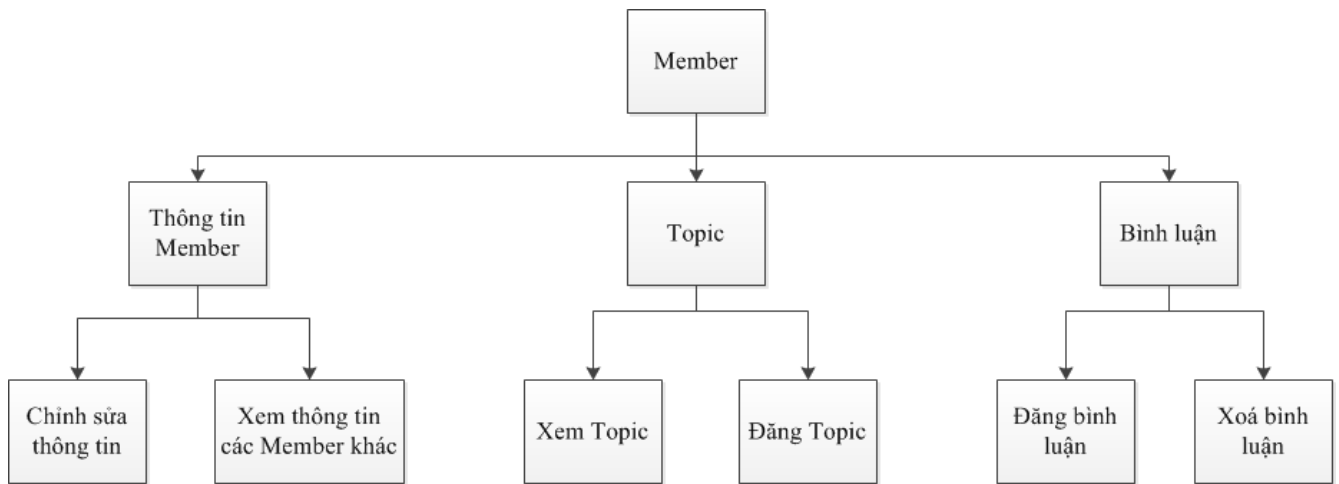
Website có 3 người dùng: Owner (chủ nhóm), Manager (quản lý nhóm) và Member (thành viên). Quyền Owner cao hơn Manager, quyền Manager cao hơn quyền Member và được mô tả chi tiết như sơ đồ phân rã chức năng bên dưới:



Sơ đồ 5.1: Sơ đồ phân rã chức năng của Owner



Sơ đồ 5.2: Sơ đồ phân rã chức năng của Manager



Sơ đồ 5.3: Sơ đồ phân rã chức năng của Member

Diễn giải thêm về phân quyền trong một group:

- Quyền Owner: quyền này chỉ dành cho người thành lập group và đây là quyền cao nhất của group đó. Trong một group thì chỉ có một người Owner.
- Quyền Member: là người xin phép gia nhập group và đã được Owner hoặc Manager duyệt. Owner hoặc Manager có quyền huỷ bỏ yêu cầu gia nhập.
- Quyền Manager: là Member sau khi được Owner chuyển thành Manager. Manager có các quyền quản lý bài viết, member khác.
- Nếu một user chưa tham gia group mà yêu cầu xem bài viết thì tùy vào cài đặt riêng của group đó. Nếu là group public thì user này có quyền xem bài viết và không được comment, ngược lại thì bị từ chối yêu cầu xem bài viết.

5.3.3 Yêu cầu phi chức năng

5.3.3.1 Yêu cầu tính tiện dụng

Giao diện của ứng dụng phải được thiết kế đơn giản, thân thiện, trực quan, và dễ sử dụng đối với người dùng. Các thành phần của ứng dụng phải được bố trí hợp lý giúp cho người dùng có thể thực hiện các chức năng một cách nhanh chóng nhằm nâng cao hiệu quả tương tác của người dùng. Ví dụ: các thành phần giao diện được sắp xếp thành nhóm để dễ dàng phân biệt, các thành phần thường sử dụng sẽ được làm nổi bật và đặt tại vị trí dễ nhận thấy...

5.3.3.2 Yêu cầu hiệu quả và thích nghi

Ứng dụng phải thực hiện các chức năng một cách chính xác và nhanh chóng. Ứng dụng phải chạy được tốt trên hầu hết các trình duyệt FireFox, IE, Chrome... Ứng dụng hoạt động hiệu quả trên cả máy tính và điện thoại di động.

5.3.3.3 Yêu cầu khả năng phát triển

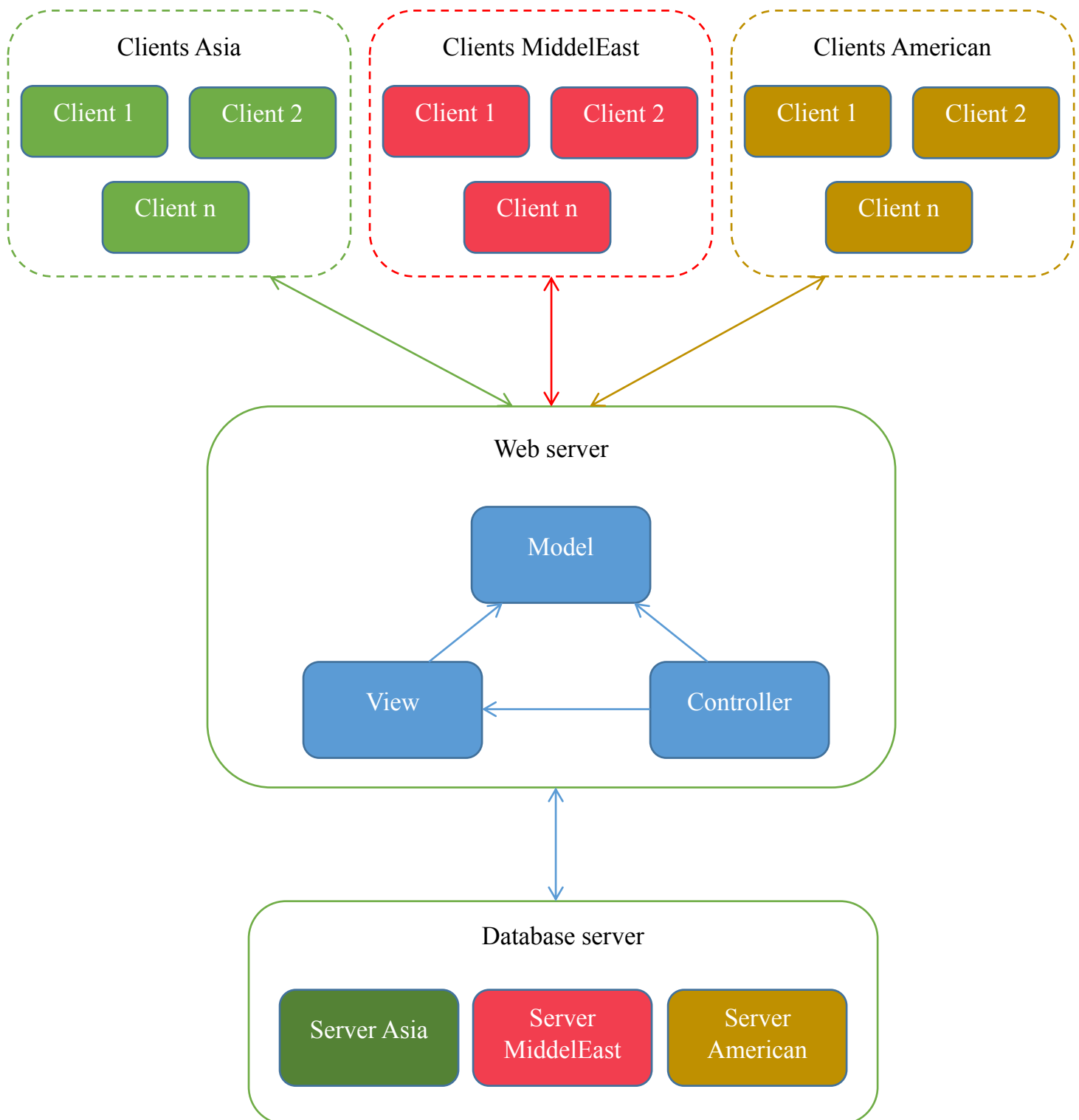
Ứng dụng phải cho phép người sử dụng nâng cấp hệ thống trong tương lai nhưng đảm bảo không ảnh hưởng đến các dữ liệu đã được lưu trữ cũng như không ảnh hưởng đến các chức năng đang được sử dụng.

5.3.3.4 Yêu cầu bảo mật

Ứng dụng phải đảm bảo người sử dụng chỉ được thao tác trên những chức năng trong giới hạn quyền của mình. Mật khẩu của người sử dụng phải được mã hóa trước khi lưu trữ xuống cơ sở dữ liệu để chắc chắn mật khẩu của người dùng không bị đánh cắp dù cho bị đánh cắp cơ sở dữ liệu. Hệ thống bảo mật phải đảm bảo ứng dụng hạn chế được các tấn công từ bên ngoài với mục đích phá hoại.

5.4 Ý tưởng thiết kế

5.4.1 Thiết kế mô hình 3 tầng (3 Tier): Clients – Web server – Database server

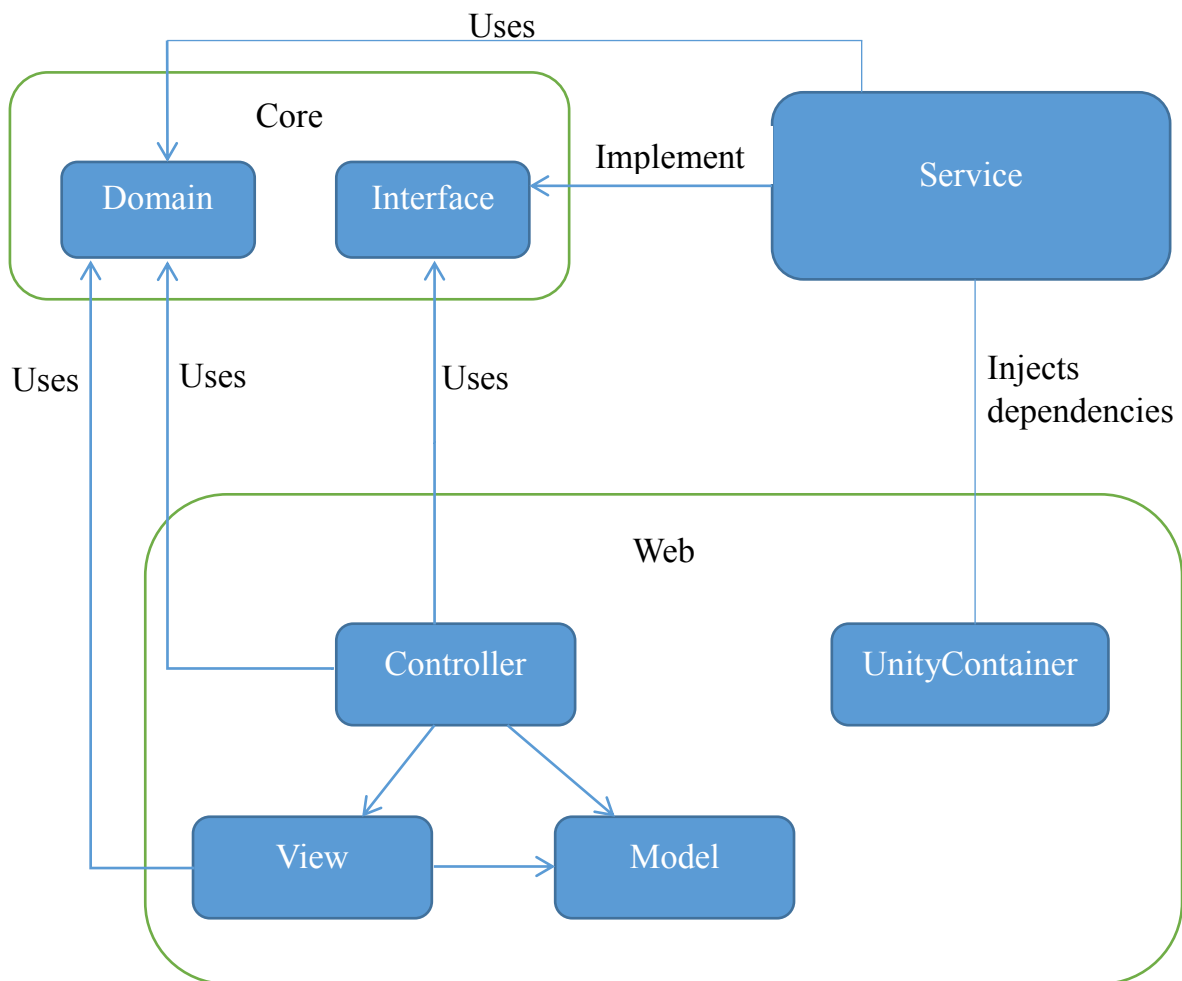


Sơ đồ 5.4: Sơ đồ thiết kế mô hình 3 tầng: Clients – Web server – Database server

Các thành phần trong mô hình này:

- **Clients:** các client nằm ở khắp nơi có thể truy cập vào website qua giao thức HTTP bằng web browser.
- **Web server:** được thiết kế theo mô hình MVC, sử dụng ASP.NET MVC 4 framework. Tầng web server sẽ được mô tả chi tiết ở mục 5.4.2 Kiến trúc website.
- **Database server:** database server sử dụng RavenDB. Sử dụng kỹ thuật Sharding để phân tán dữ liệu ra nhiều Server có thể đặt ở nhiều nơi. Tùy vào request đến từ đâu sẽ đi đến server thích hợp để lấy dữ liệu.

5.4.2 Kiến trúc Website



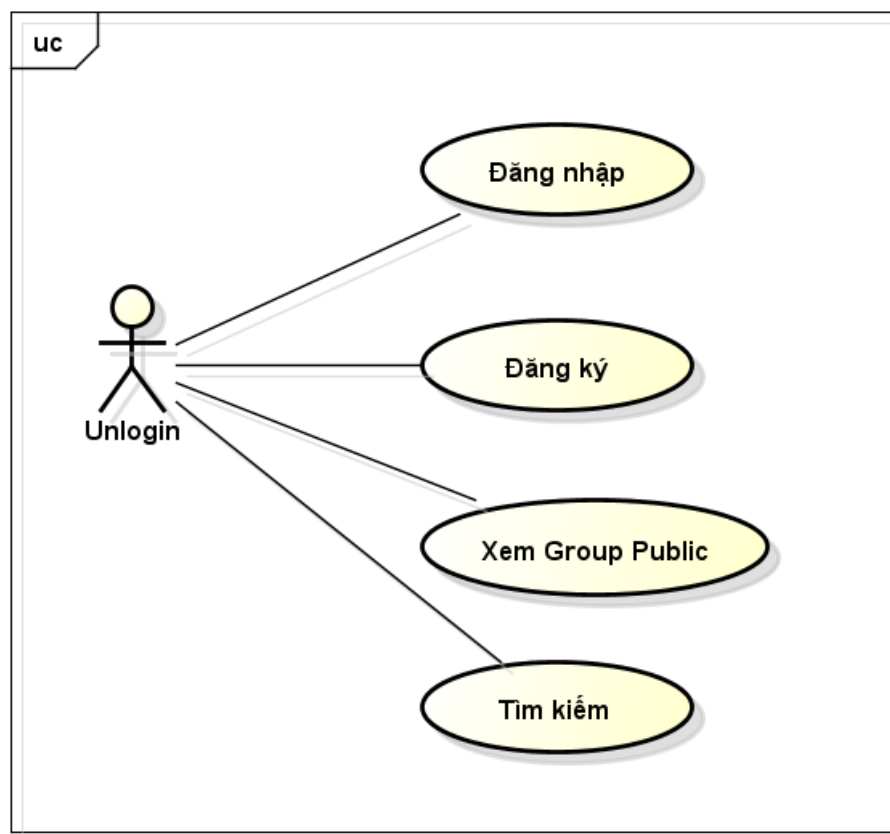
Sơ đồ 5.5: Sơ đồ kiến trúc website

Các thành phần trong mô hình này:

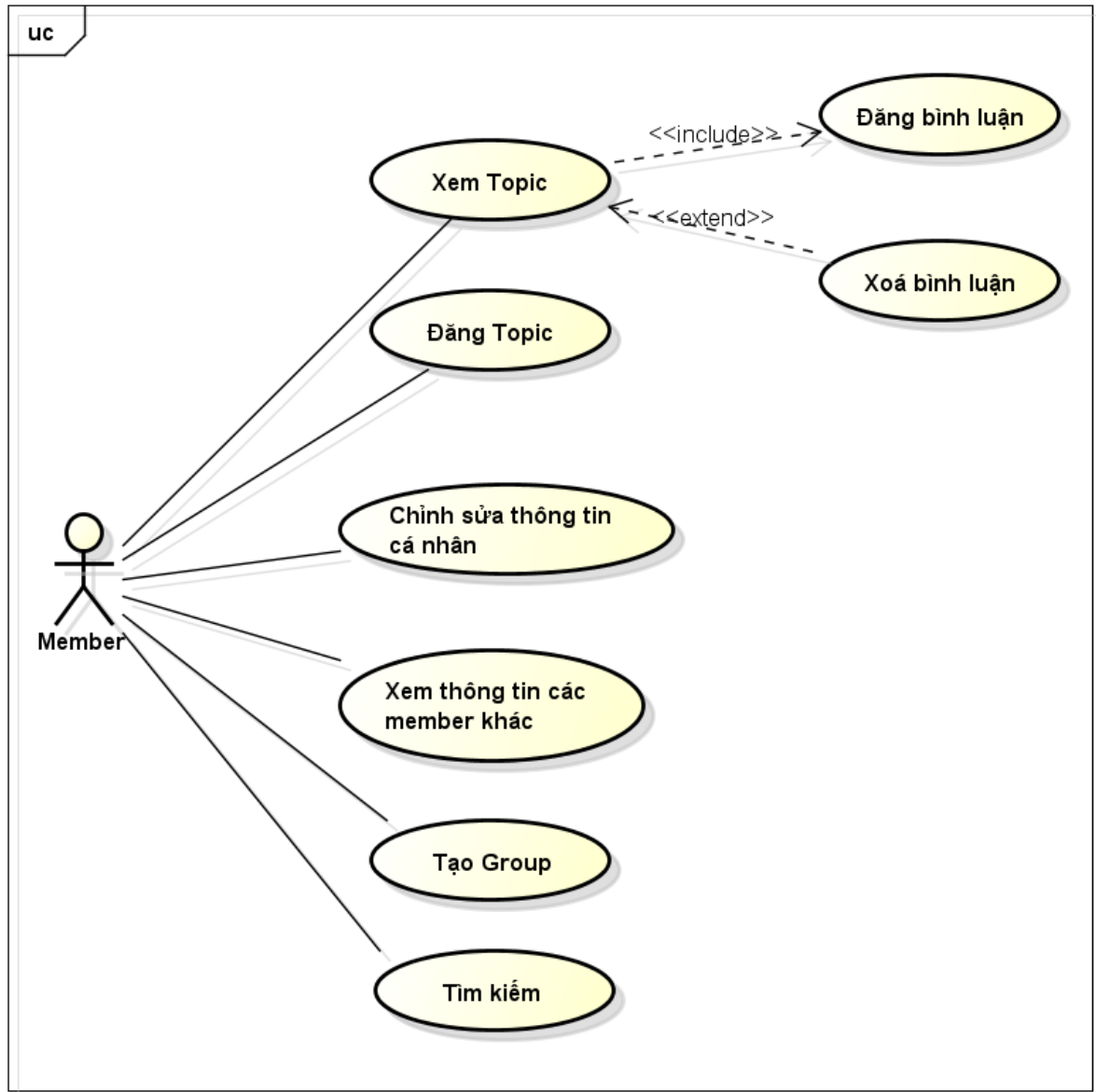
- **Project Core:** Chứa Domain và các Interface. Domain làm nhiệm vụ đại diện cho một đối tượng trong chương trình. Interface là các giao diện cung cấp các hàm giúp cho Controller làm việc, chủ yếu là lấy dữ liệu từ database.
- **Project Server:** các lớp trong này hiện thực các Interface trong project Core. Các lớp này sẽ giao tiếp giữa Controller và Database.
- **Project Web:** thiết kế theo mô hình MVC. Ngoài ra còn sử dụng UnityContainer để tiêm sự phụ thuộc (Inject Dependencies) cho các Interface trong project Core quan hệ với các class trong project Server. Các class trong project Web có thể sử dụng Domain để chứa đối tượng.

5.5 Phân tích, thiết kế hệ thống

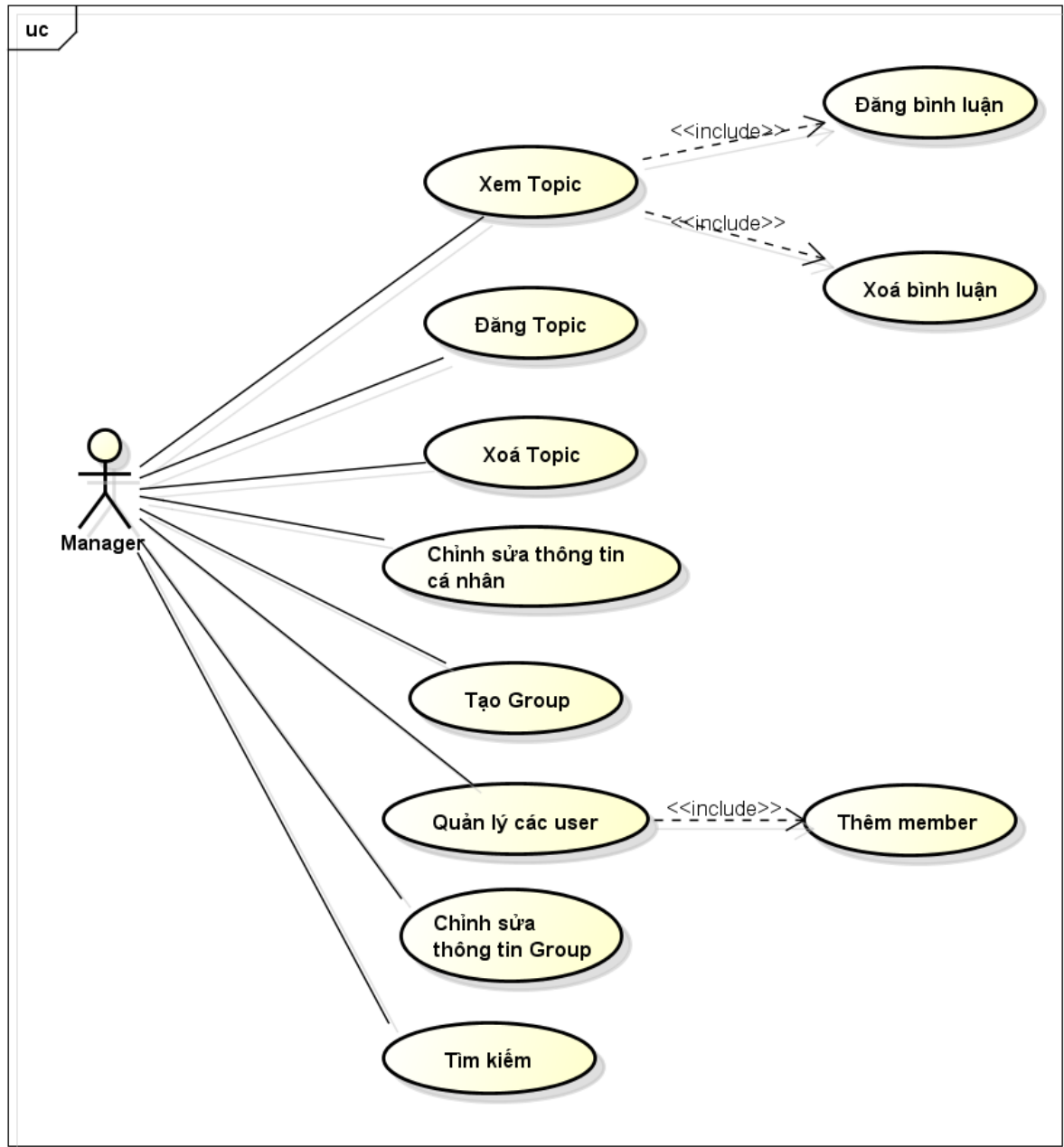
5.5.1 Sơ đồ use case



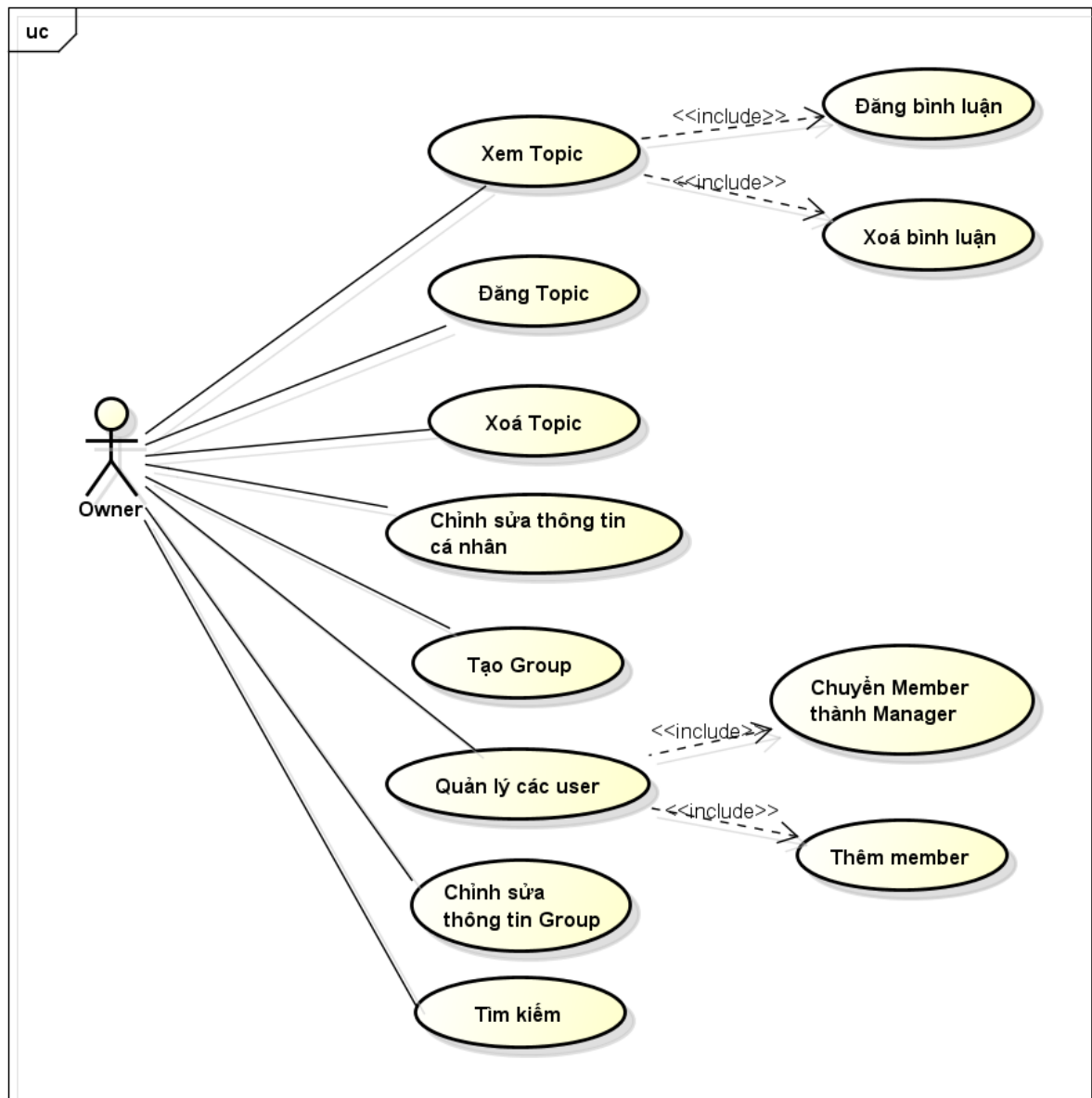
Sơ đồ 5.6: Use case trong trường hợp chưa đăng nhập



Sơ đồ 5.7: Use case của Member



Sơ đồ 5.8: Use case của Manager



Sơ đồ 5.9: Use case của Owner

5.5.1.1 Danh sách actor

STT	Actor	Ý nghĩa
1	Người dùng chưa đăng nhập	Tạo mới tài khoản, đăng nhập vào hệ thống, tìm kiếm những nhóm public và xem nội dung của nhóm này
2	Member	Tạo nhóm mới, đăng bài viết, đăng và xóa bình luận, chỉnh sửa thông tin cá nhân, tìm kiếm nhóm...
3	Manager	Tạo nhóm mới, quản lý nhóm, quản lý bài viết, bình luận, quản lý thành viên trong nhóm, tìm kiếm nhóm...
4	Owner	Tạo nhóm mới, quản lý nhóm, quản lý bài viết, bình luận, quản lý thành viên trong nhóm, tìm kiếm nhóm, nâng cấp quyền member thành manager...

Bảng 5.2: Danh sách actor

5.5.1.2 Danh sách use case

STT	Actor	Ý nghĩa
1	Đăng nhập	Actor đăng nhập để sử dụng hệ thống
2	Đăng ký tài khoản	Actor đăng ký tài khoản hệ thống
3	Xem nhóm public	Actor xem nhóm public mà không cần đăng nhập vào hệ thống
4	Tìm kiếm	Actor thực hiện việc tìm kiếm nhóm dựa trên thông tin: tên nhóm, mô tả nhóm, tên người tạo nhóm
5	Xem topic	Actor xem bài viết và tất cả bình luận của bài viết đó
6	Đăng bình luận	Actor thực hiện việc đăng bình luận cho bài viết
7	Xóa bình luận	Actor thực hiện việc xóa bình luận
8	Đăng topic	Actor thực hiện việc đăng bài viết mới
9	Xóa topic	Actor thực hiện việc xóa topic với quyền owner hoặc manager

10	Chỉnh sửa thông tin cá nhân	Actor thực hiện việc chỉnh sửa thông tin cá nhân
11	Tạo nhóm mới	Actor thực hiện việc tạo mới một nhóm
12	Chỉnh sửa thông tin nhóm	Actor thực hiện việc thay đổi thông tin của nhóm
13	Thêm thành viên	Actor thực hiện việc thêm thành viên mới với quyền owner hoặc manager
14	Thay đổi quyền	Actor với quyền owner nâng cấp quyền member lên thành manager

Bảng 5.3: Danh sách use case

5.5.1.3 Mô tả Use case

Usecase	Đăng nhập
Mô tả	Cho phép người dùng đăng nhập hệ thống.
Điều kiện tiên quyết	Đã có tài khoản đăng nhập vào hệ thống.
Luồng xử lý chính	Sau khi người dùng nhập tên đăng nhập và mật khẩu vào form đăng nhập và bấm “Login”, hệ thống mã hoá mật khẩu theo chuẩn MD5 rồi kiểm tra tên đăng nhập và mật khẩu có tồn tại trong cơ sở dữ liệu, sau đó thông báo kết quả.
Kết quả sau khi hoàn tất Usecase	<ul style="list-style-type: none"> - Nếu đăng nhập thành công: hiển thị tên người đang đăng nhập và các nhóm mà người này đang tham gia. - Ngược lại: thông báo đăng nhập thất bại, yêu cầu nhập lại tên đăng nhập và mật khẩu.
Luồng xử lý khác/ ngoại lệ	Nếu chưa nhập đủ tên đăng nhập hoặc mật khẩu, hệ thống sẽ yêu cầu nhập đủ thông tin trước khi thực hiện kiểm tra.

Bảng 5.4: Mô tả Usecase đăng nhập

Usecase	Đăng ký
Mô tả	Cho phép người dùng đăng ký một tài khoản của hệ thống.
Điều kiện tiên quyết	
Luồng xử lý chính	<p>Sau khi người dùng nhập đầy đủ thông tin vào form đăng ký và bấm “Register”, hệ thống thực hiện kiểm tra tên đăng nhập đã tồn tại hay chưa:</p> <ul style="list-style-type: none"> - Nếu đã tồn tại: thông báo tên đăng nhập đã tồn tại và yêu cầu người dùng nhập tên khác - Nếu chưa tồn tại: tiếp tục thực hiện quá trình đăng ký. Hệ thống thực hiện mã hoá mật khẩu theo chuẩn MD5 sau đó lưu xuống cơ sở dữ liệu.
Kết quả sau khi hoàn tất Usecase	<ul style="list-style-type: none"> - Nếu đăng ký thành công: thông báo việc đăng ký thành công. - Ngược lại: thông báo lỗi và yêu cầu nhập lại thông tin.
Luồng xử lý khác/ ngoại lệ	<ul style="list-style-type: none"> - Nếu nhập tên đăng nhập đã tồn tại: hệ thống thông báo yêu cầu nhập tên khác. - Nếu mật khẩu lần 2 không khớp với lần 1: thông báo yêu cầu nhập mật khẩu lại. - Nếu nhập email không đúng: yêu cầu người dùng nhập chính xác.

Bảng 5.5: Mô tả Usecase đăng ký

Usecase	Xem Group public
Mô tả	Người dùng chưa đăng nhập có thể xem nhóm được cài đặt public.
Điều kiện tiên quyết	Nhóm mà người dùng muốn xem phải được cài đặt ở dạng public.
Luồng xử lý chính	Thực hiện lấy thông tin nhóm từ cơ sở dữ liệu lên rồi kiểm tra xem có phải nhóm public không. Nếu là nhóm public thì thực hiện hiển thị thông tin các bài đăng ra cho người dùng xem.
Kết quả sau khi hoàn tất Usecase	Người dùng thấy các bài đăng của nhóm . Có thể xem chi tiết các bài đăng và các bình luận.
Luồng xử lý khác/ ngoại lệ	Nếu nhóm được yêu cầu không phải nhóm public thì một thông báo hiện ra yêu cầu người dùng phải tham gia (join) nhóm.

Bảng 5.6: Mô tả Usecase xem Group public

Usecase	Tìm kiếm
Mô tả	Cho phép người dùng tìm kiếm thông tin nhóm.
Điều kiện tiên quyết	Biết tên, thông tin mô tả của ít nhất 1 nhóm.
Luồng xử lý chính	Sau khi người dùng nhập từ tìm kiếm và bấm nút “Search”, hệ thống thực hiện tìm kiếm chính xác theo tên. Nếu tìm thấy kết quả thì trả kết quả về. Ngược lại, tiếp tục tìm kiếm gần đúng theo tên, tìm kiếm theo mô tả (description) của các nhóm và hiển thị kết quả tìm kiếm lên.
Kết quả sau khi hoàn tất Usecase	Các nhóm có tên, mô tả giống/ gần giống với từ tìm kiếm.
Luồng xử lý khác/ ngoại lệ	Không tìm thấy nhóm nào có tên, mô tả giống/ gần giống với từ tìm kiếm. Hệ thống thông báo không tìm thấy kết quả.

Bảng 5.7: Mô tả Usecase tìm kiếm

Usecase	Xem topic
Mô tả	Hiển thị nội dung của một bài đăng và các bình luận của bài đăng này
Điều kiện tiên quyết	Nếu không là group public thì cần đăng nhập hệ thống
Luồng xử lý chính	Hệ thống lấy dữ liệu của bài đăng và các bình luận của nó từ cơ sở dữ liệu lên. Sau đó kiểm tra quyền của người đang đăng nhập: <ul style="list-style-type: none"> - Nếu là member: cho phép đăng trả lời - Nếu là manager/ owner: cho phép đăng và xóa trả lời. - Nếu người dùng chưa đăng nhập hoặc chưa tham gia nhóm: chỉ cho xem nội dung.
Kết quả sau khi hoàn tất Usecase	Người dùng xem được nội dung của một bài đăng cùng với các bình luận.
Luồng xử lý khác/ ngoại lệ	Nếu người dùng chưa đăng nhập hoặc chưa tham gia nhóm này và group không phải group public thì hệ thống không hiển thị nội dung bài đăng, đồng thời yêu cầu người dùng tham gia nhóm.

Bảng 5.8: Mô tả Usecase xem topic

Usecase	Đăng bình luận
Mô tả	Cho phép người dùng đăng bình luận trong một bài đăng.
Điều kiện tiên quyết	Người dùng phải đang đăng nhập và tham gia nhóm.
Luồng xử lý chính	Sau khi người dùng nhập bình luận và bấm “Post”, hệ thống thực hiện lưu bình luận xuống cơ sở dữ liệu đồng thời cập nhật giao diện web.
Kết quả sau khi hoàn tất Usecase	Bình luận được lưu xuống cơ sở dữ liệu và giao diện được cập nhật mới lại.
Luồng xử lý khác/ ngoại lệ	Khi người dùng chưa tham gia nhóm, việc đăng bình luận bị ngưng lại và một thông báo yêu cầu người dùng tham gia vào nhóm trước khi đăng bình luận.

Bảng 5.9: Mô tả Usecase đăng bình luận

Usecase	Xoá bình luận
Mô tả	Cho phép người dùng xoá bình luận trong một bài đăng.
Điều kiện tiên quyết	Người dùng phải đang đăng nhập và tham gia nhóm. Nếu người đang đăng nhập là Member thì chỉ được xoá bình luận của chính họ. Nếu người đang đăng nhập là Manager hoặc Owner thì có quyền xoá bất cứ bình luận.
Luồng xử lý chính	Sau khi người dùng chọn xoá bình luận, hệ thống thực hiện xoá bình luận khỏi cơ sở dữ liệu đồng thời cập nhật lại giao diện web.
Kết quả sau khi hoàn tất Usecase	Bình luận được xoá khỏi cơ sở dữ liệu và giao diện được cập nhật mới lại.
Luồng xử lý khác/ ngoại lệ	

Bảng 5.10: Mô tả Usecase xoá bình luận

Usecase	Đăng Topic
Mô tả	Tạo mới một bài đăng (topic) cho một group.
Điều kiện tiên quyết	Người dùng cần đăng nhập vào hệ thống và đã tham gia group.
Luồng xử lý chính	Người dùng nhập đầy đủ thông tin của topic vào form và nhấn “Post”, hệ thống thực hiện lưu các thông tin sau xuống cơ sở dữ liệu: + Thông tin của Topic. + Thêm thông tin của Topic mà người dùng vừa nhập vào nhóm. Sau đó chuyển về trang chủ.
Kết quả sau khi hoàn tất Usecase	Trang chủ hiển thị tên của topic vừa mới tạo.
Luồng xử lý khác/ ngoại lệ	Khi người dùng chưa tham gia nhóm, việc đăng topic bị ngưng lại và một thông báo yêu cầu người dùng tham gia vào nhóm trước khi đăng topic.

Bảng 5.11: Mô tả Usecase đăng topic

Usecase	Xoá Topic
Mô tả	Cho phép người có quyền “manager” và “owner” xoá các Topic.
Điều kiện tiên quyết	Đã đăng nhập vào hệ thống với quyền “manager” hoặc “owner”.
Luồng xử lý chính	<p>Để xoá Topic, người dùng trở ra trang xem danh sách các Topic và check vào các checkbox trước các Topic cần xoá. Khi đó nút “Delete” được hiển thị lên cho phép người dùng xoá các Topic đã chọn.</p> <p>Khi lệnh xoá Topic được thực hiện, các Topic được chọn sẽ được xoá lần lượt dưới cơ sở dữ liệu như sau:</p> <ul style="list-style-type: none"> + Xoá Topic có ID tương ứng. + Xoá Topic trong danh sách Topic của nhóm. <p>Sau đó thông báo xoá thành công sẽ được hiển thị.</p>
Kết quả sau khi hoàn tất Usecase	Xoá thành công các Topic được chọn. Trên màn hình các topic được chọn bị mất đi.
Luồng xử lý khác	

Bảng 5.12: Mô tả Usecase xoá Topic

Usecase	Chỉnh sửa thông tin cá nhân
Mô tả	Cho phép người dùng chỉnh sửa thông tin của họ trên hệ thống.
Điều kiện tiên quyết	Đã đăng nhập vào hệ thống.
Luồng xử lý chính	<p>Để cập nhật thông tin cá nhân, người dùng cần đi đến trang chỉnh sửa thông tin cá nhân và điền đầy đủ các thông tin mới sau đó nhấn “Save”. Sau đó thông báo cập nhật thành công sẽ được hiển thị.</p>
Kết quả sau khi hoàn tất Usecase	Cập nhật thành công thông tin mới vào cơ sở dữ liệu. Trên màn hình thông báo việc chỉnh sửa thành công.
Luồng xử lý khác/ ngoại lệ	Điền thông tin không đầy đủ và chính xác (email) sẽ bị báo lỗi khi nhấn “Save”.

Bảng 5.13: Mô tả Usecase chỉnh sửa thông tin cá nhân

Usecase	Tạo Group
Mô tả	Tạo một nhóm (group) mới.
Điều kiện tiên quyết	Người dùng cần đăng nhập vào hệ thống.
Luồng xử lý chính	<p>Người dùng nhập đầy đủ thông tin của nhóm vào form và nhấn “Create”, hệ thống thực hiện lưu các thông tin sau xuống cơ sở dữ liệu.</p> <ul style="list-style-type: none"> + Thông tin nhóm mà người dùng vừa nhập. + Thêm thông tin nhóm vào danh sách nhóm của người tạo với quyền là “Owner”. + Thêm thông tin quan hệ giữa nhóm và người tạo với quyền là “Owner”. <p>Sau đó chuyển về trang chi tiết của nhóm.</p>
Kết quả sau khi hoàn tất Usecase	Trang chi tiết của nhóm được hiển thị, chưa có topic nào trong này. Người vừa tạo chính là người sở hữu nhóm (Owner) và có quyền cao nhất.
Luồng xử lý khác/ ngoại lệ	

Bảng 5.14: Mô tả Usecase tạo group

Usecase	Chỉnh sửa thông tin group
Mô tả	Chỉnh sửa thông tin của một group.
Điều kiện tiên quyết	Người dùng cần đăng nhập vào hệ thống với quyền Owner hoặc Manager.
Luồng xử lý chính	Người dùng đi đến trang chỉnh sửa thông tin group và nhập đầy đủ thông tin của nhóm vào form, sau đó nhấn “Save” để tiến hành cập nhật thông tin mới xuống cơ sở dữ liệu. Sau khi cập nhật xong, thông báo lên màn hình quá trình cập nhật đã thành công.
Kết quả sau khi hoàn tất Usecase	Thông tin của group được cập nhật mới.
Luồng xử lý khác/ ngoại lệ	Nhập không đầy đủ thông tin khi đó hệ thống sẽ báo lỗi yêu cầu nhập lại.

Bảng 5.15: Mô tả Usecase chỉnh sửa thông tin group

Usecase	Thêm member
Mô tả	Cho phép một user tham gia group.
Điều kiện tiên quyết	Người đang đăng nhập phải có quyền Manager hoặc Owner của group. Có một hoặc nhiều user yêu cầu tham gia group.
Luồng xử lý chính	Manager hoặc owner của group vào trang quản lý member và lựa chọn user muốn cho phép tham gia group và chọn “Accept”. Hệ thống sẽ cập nhật lại trạng thái và cấp quyền member cho user đó tham gia group.
Kết quả sau khi hoàn tất Usecase	User vừa được cho phép tham gia bởi Manager hoặc Owner trở thành member của group và có quyền đăng bài, đăng bình luận trong group.
Luồng xử lý khác/ ngoại lệ	Khi Manager hoặc Owner không cho phép user tham gia group thì có thể chọn “Reject” để hủy yêu cầu tham gia của user đó.

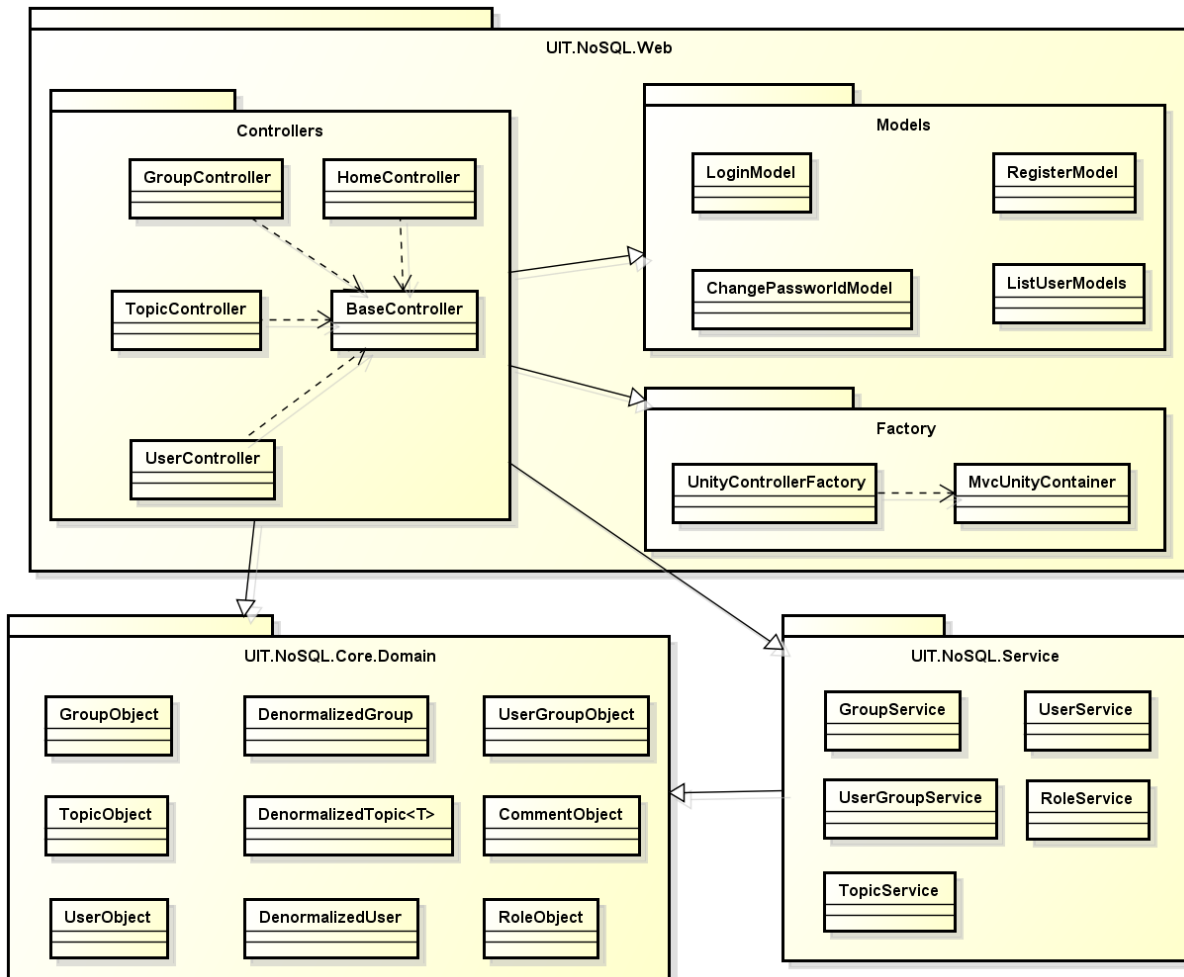
Bảng 5.16: Mô tả Usecase thêm member

Usecase	Thay đổi quyền
Mô tả	Thay đổi quyền từ Member thành Manager và ngược lại
Điều kiện tiên quyết	Owner của group đang đăng nhập
Luồng xử lý chính	Owner của group vào trang quản lý member và thay đổi quyền cho member. Quyền mới vừa được Owner cài đặt sẽ được update xuống cơ sở dữ liệu.
Kết quả sau khi hoàn tất Usecase	Người vừa được thay đổi sẽ có quyền mới đối với group này và hiện thông báo kết quả thay đổi thành công lên màn hình.
Luồng xử lý khác/ ngoại lệ	Không thể thay đổi quyền của Owner.

Bảng 5.17: Mô tả Usecase tạo thay đổi quyền

5.5.2 Class diagram

Mô hình class diagram của hệ thống được mô tả như sau:



Sơ đồ 5.10: Sơ đồ lớp cung cấp các chức năng chính cho website

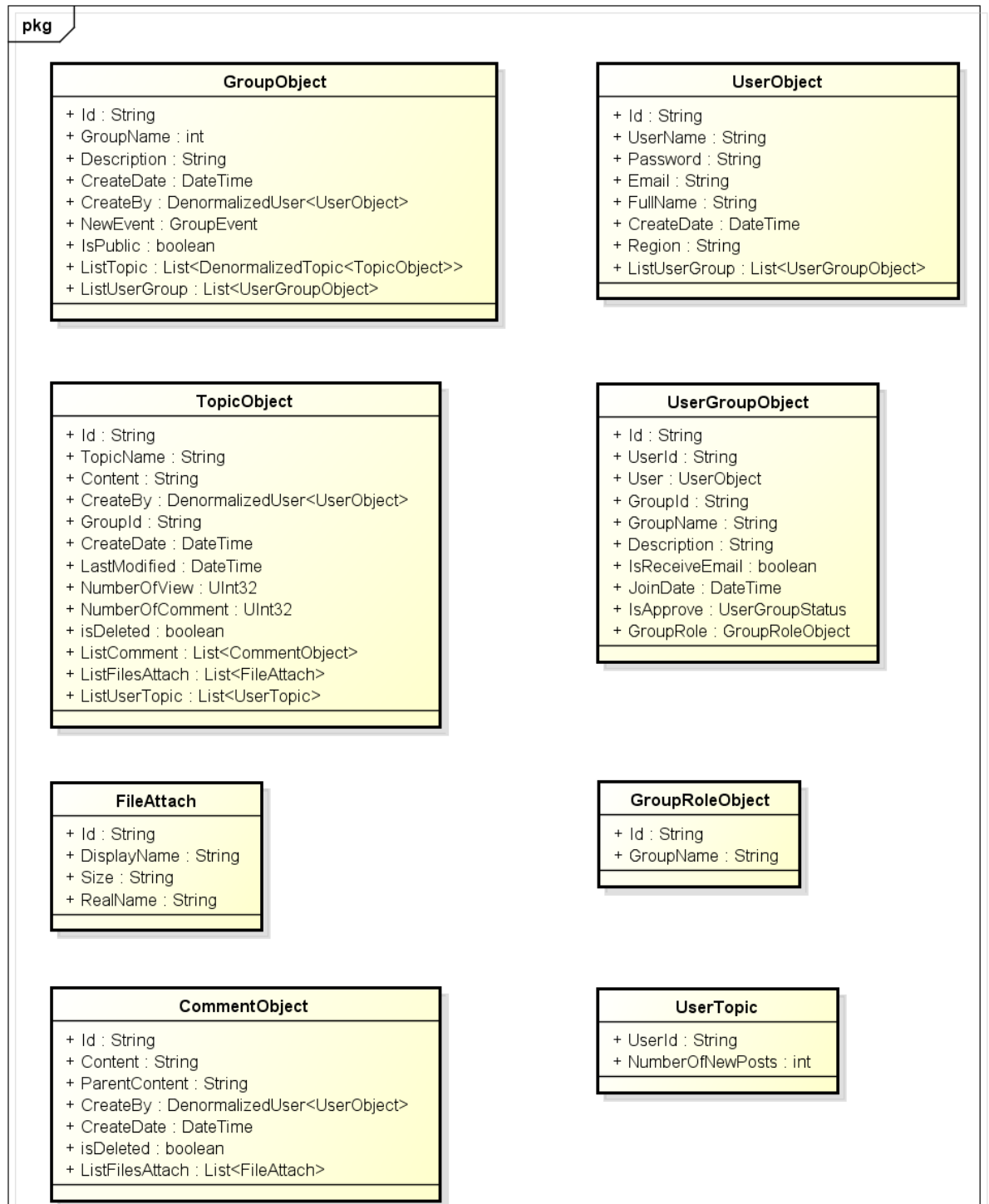
Các class Controller là một thành phần của mô hình MVC làm nhiệm vụ nhận và điều hướng các xử lý từ view xuống các class Service và sau đó trả kết quả lại cho view. Sau đây là mô tả chức năng từng class trong Controller:

STT	Tên class	Chức năng
1	HomeController	Nhận và điều khiển các xử lý cho trang home
2	GroupController	Nhận và điều khiển các xử lý cho trang xem chi tiết group, các chức năng khác như tìm kiếm, manager...
3	TopicController	Nhận và điều khiển các xử lý cho trang xem chi tiết topic, đăng bình luận
4	UserController	Nhận và điều khiển các xử lý liên quan đến user như login, register, profile

Core.Domain chứa dữ liệu của từng object. Chức năng mỗi class như sau:

STT	Tên class	Chức năng
1	GroupRole	Chứa thông tin của một quyền truy cập hệ thống.
2	Group	Chứa thông tin của một group.
3	Topic	Chứa thông tin của một topic.
4	UserGroup	Chứa thông tin đăng ký một user vào một group.
5	User	Chứa thông tin của một user

Cấu trúc của mỗi class trong Core.Domain sẽ được dùng làm cấu trúc lưu xuống database. Cấu trúc chi tiết của từng class được mô tả như sơ đồ sau:



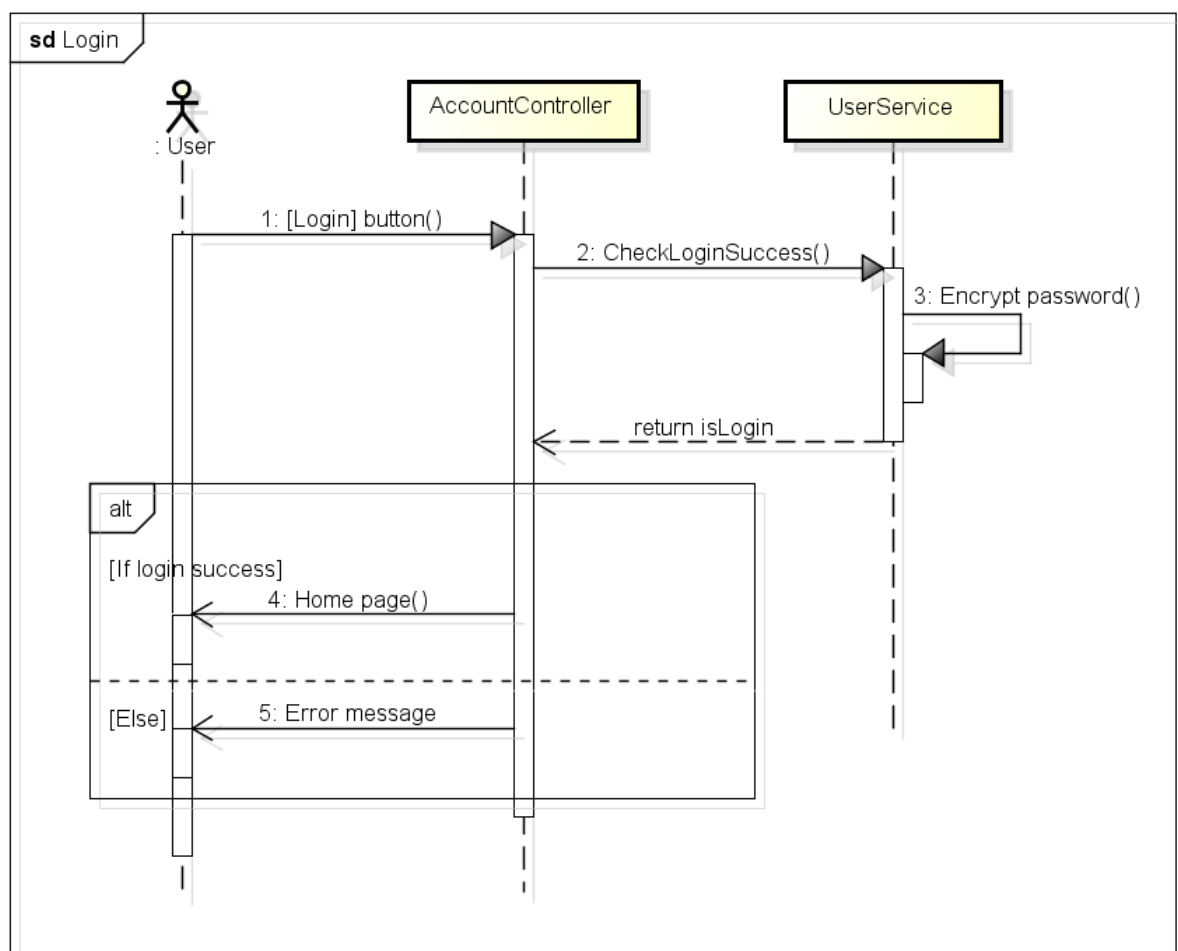
Hình Class diagram của Core.Domain

Core.Service chứa các class xử lý nghiệp vụ, xử lý lưu trữ và truy vấn dữ liệu. Các class này được implement từ Core.IService. Chức năng mỗi class như sau:

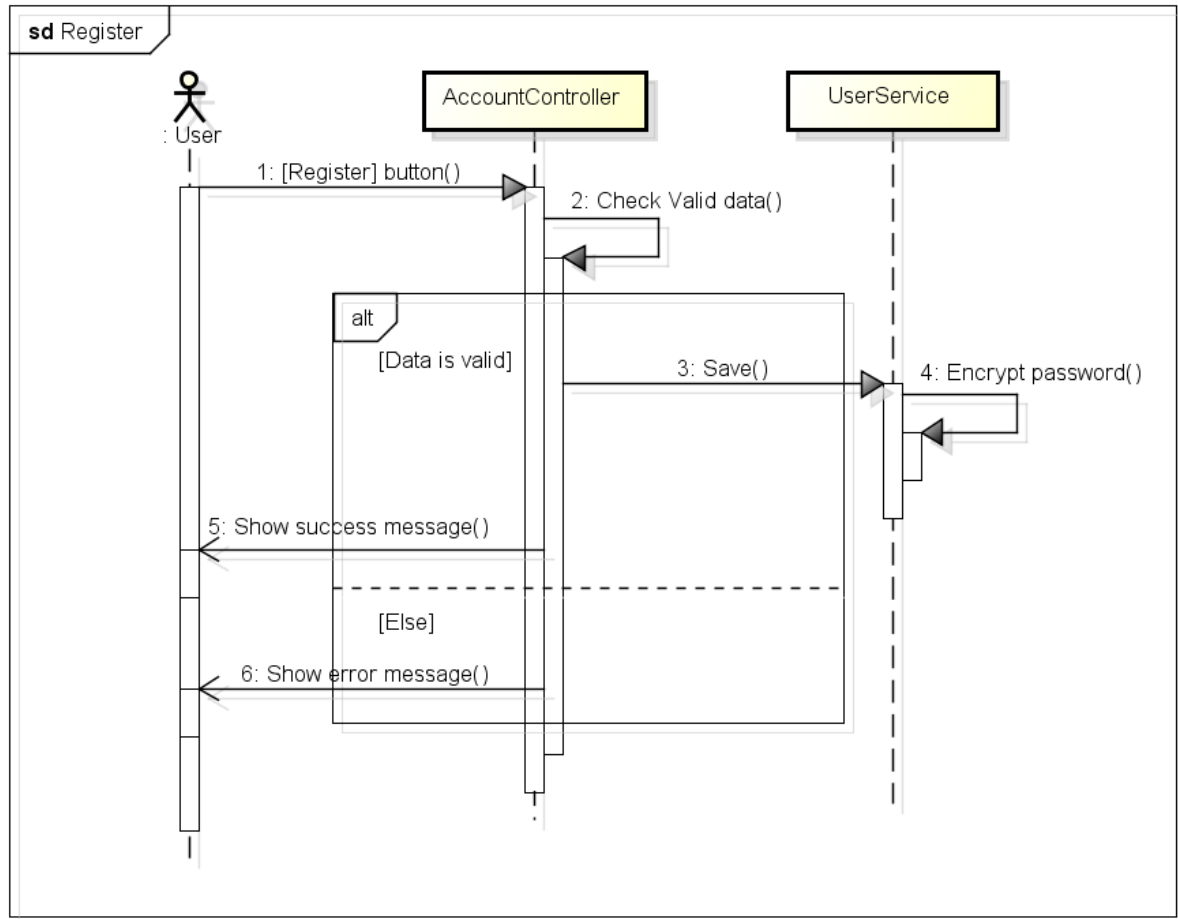
STT	Tên class	Chức năng
1	GroupRoleService	Xử lý, lưu trữ và truy vấn các thông tin của GroupRole
2	GroupService	Xử lý, lưu trữ và truy vấn các thông tin của Group
3	TopicService	Xử lý, lưu trữ và truy vấn các thông tin của Topic
4	UserGroupService	Xử lý, lưu trữ và truy vấn các thông tin của UserGroup
5	UserService	Xử lý, lưu trữ và truy vấn các thông tin của User

5.5.3 Sequence diagram

5.5.3.1 Quản lý User

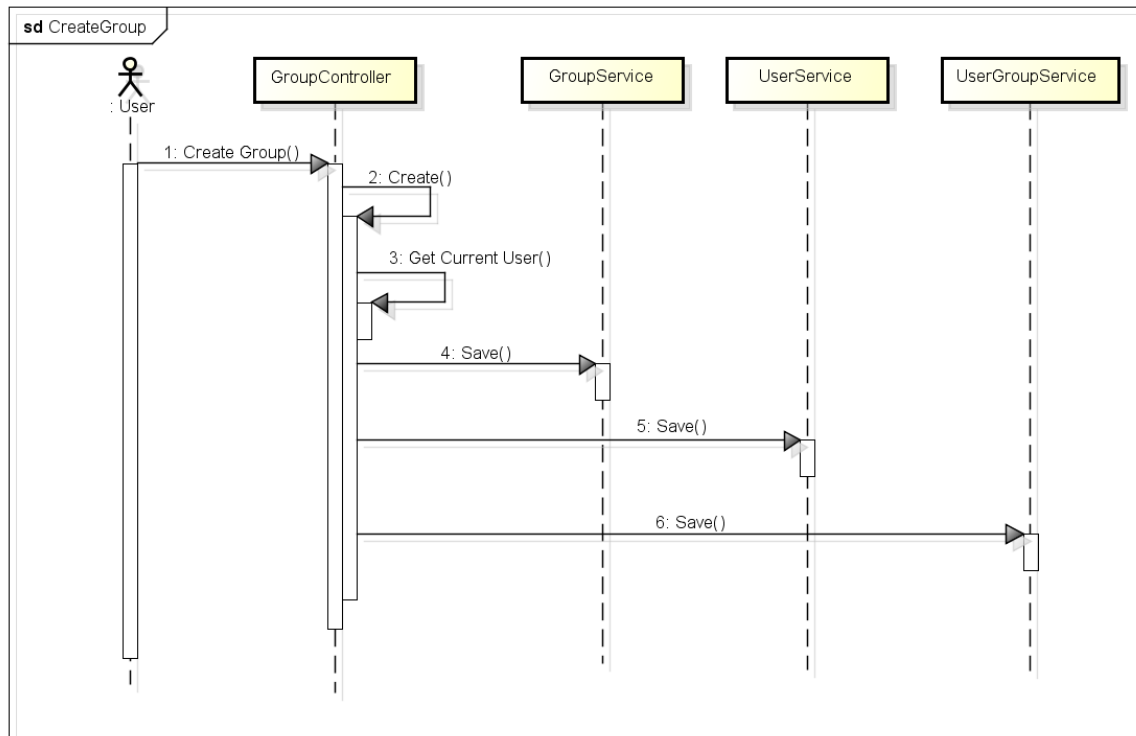


Sơ đồ 5.11: Sequence diagram của chức năng Login

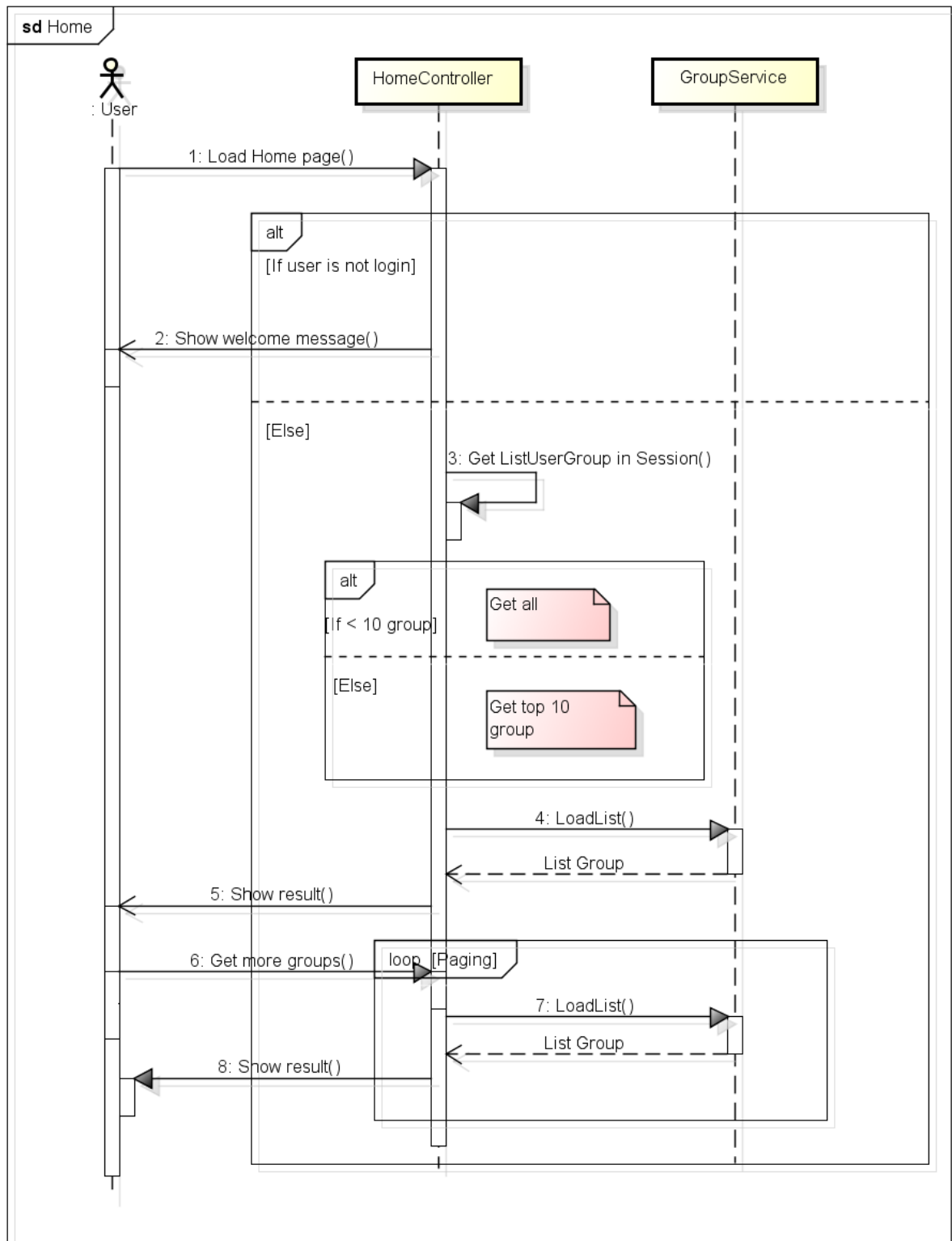


Sơ đồ 5.12: Sequence diagram thực hiện chức năng Register

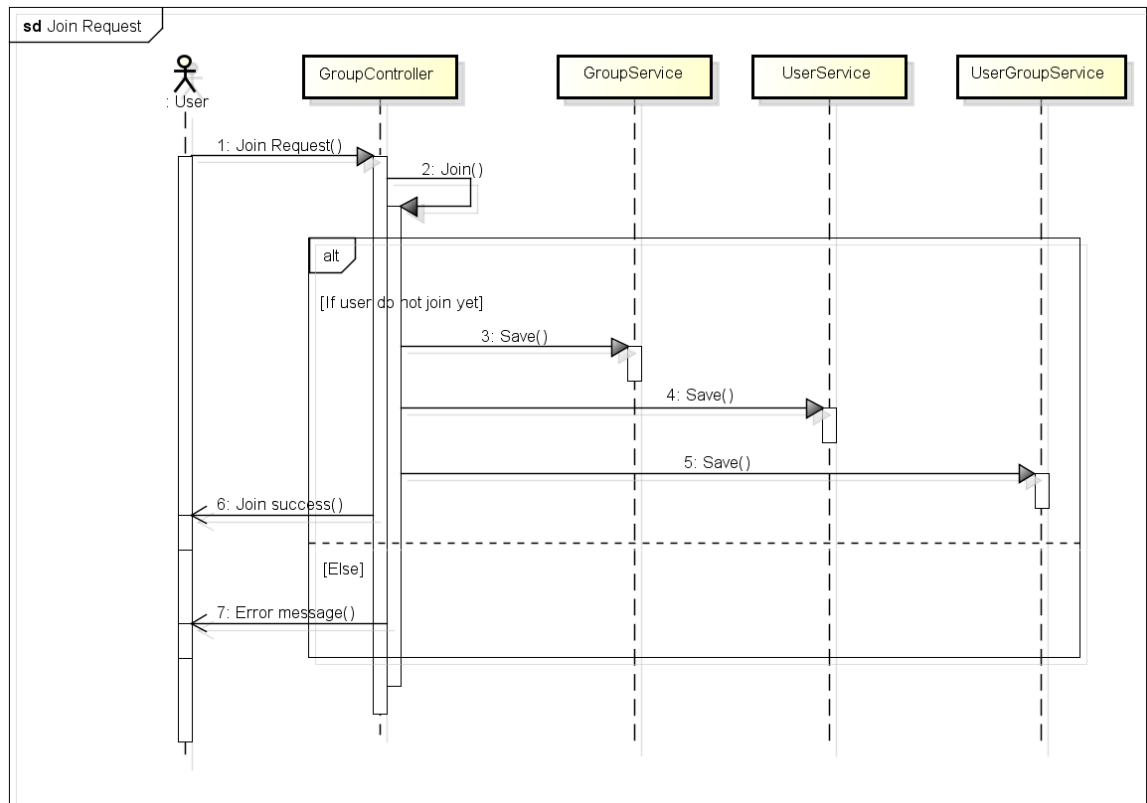
5.5.3.2 Quản lý Group



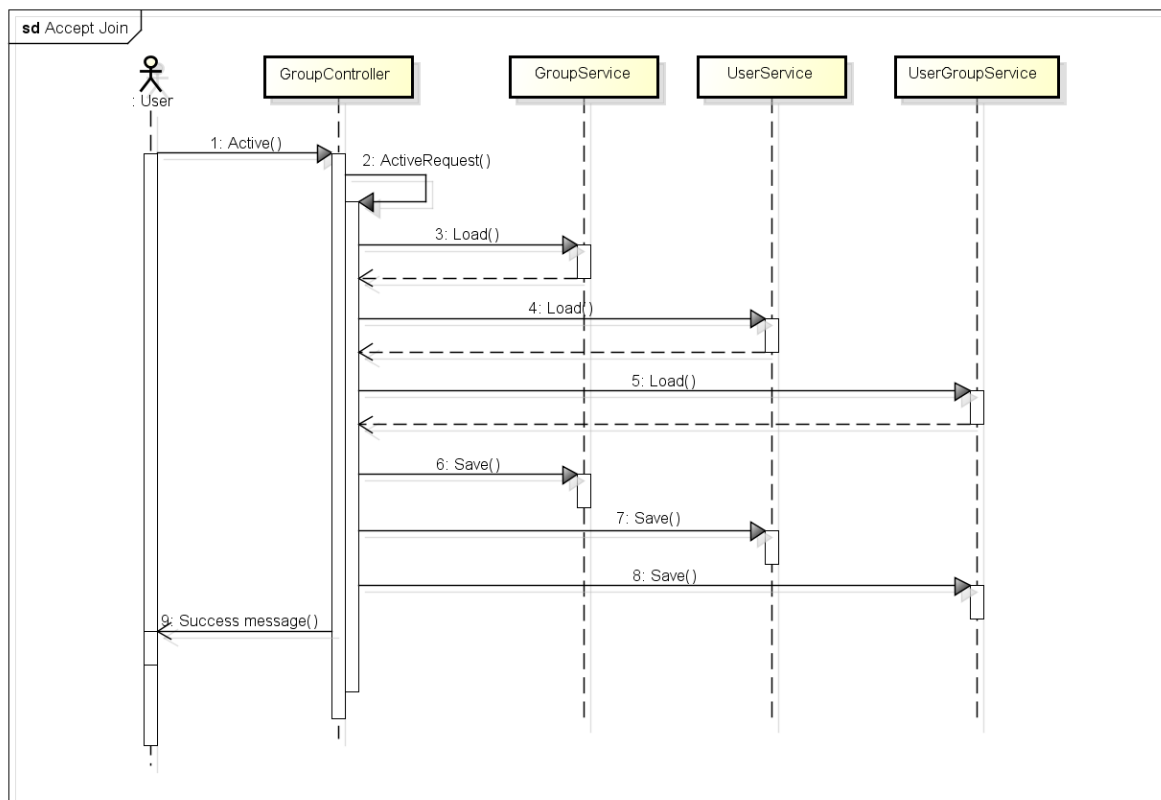
Sơ đồ 5.13: Sequence diagram của chức năng Create Group



Sơ đồ 5.14: Sequence diagram của trang Home

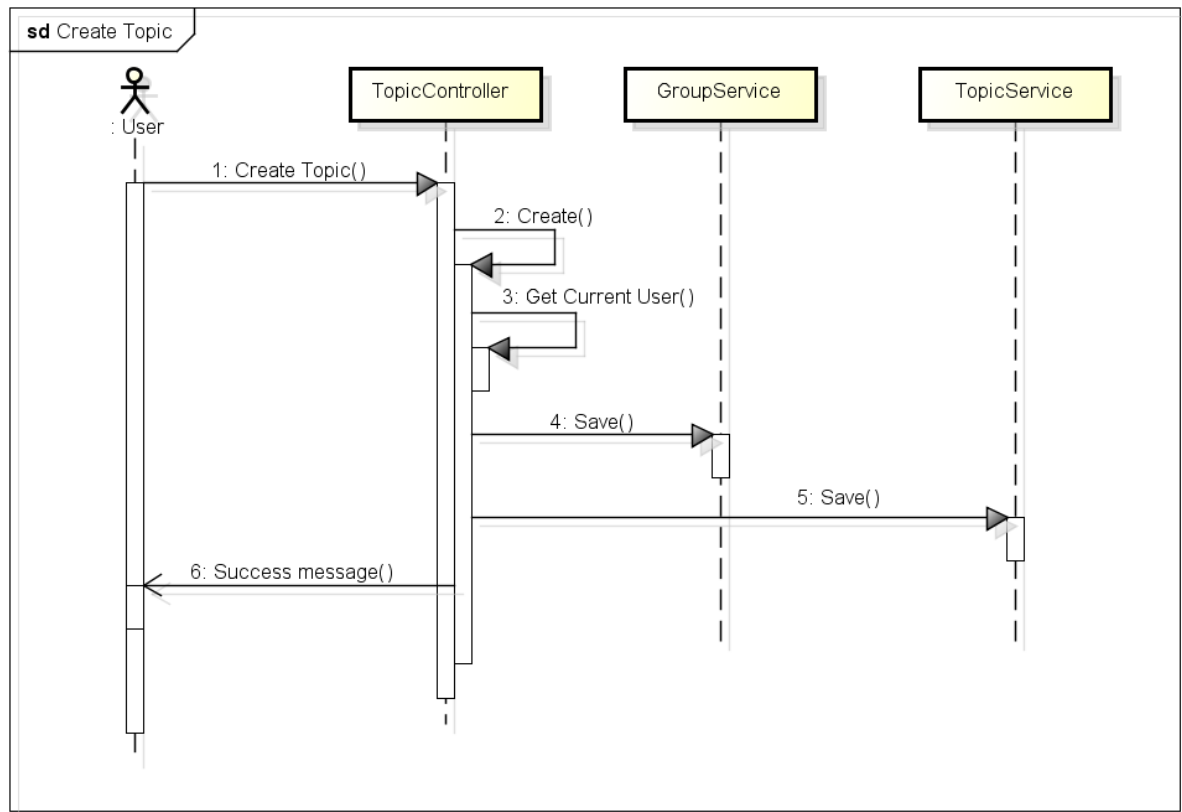


Sơ đồ 5.15: Sequence diagram của chức năng Join Group

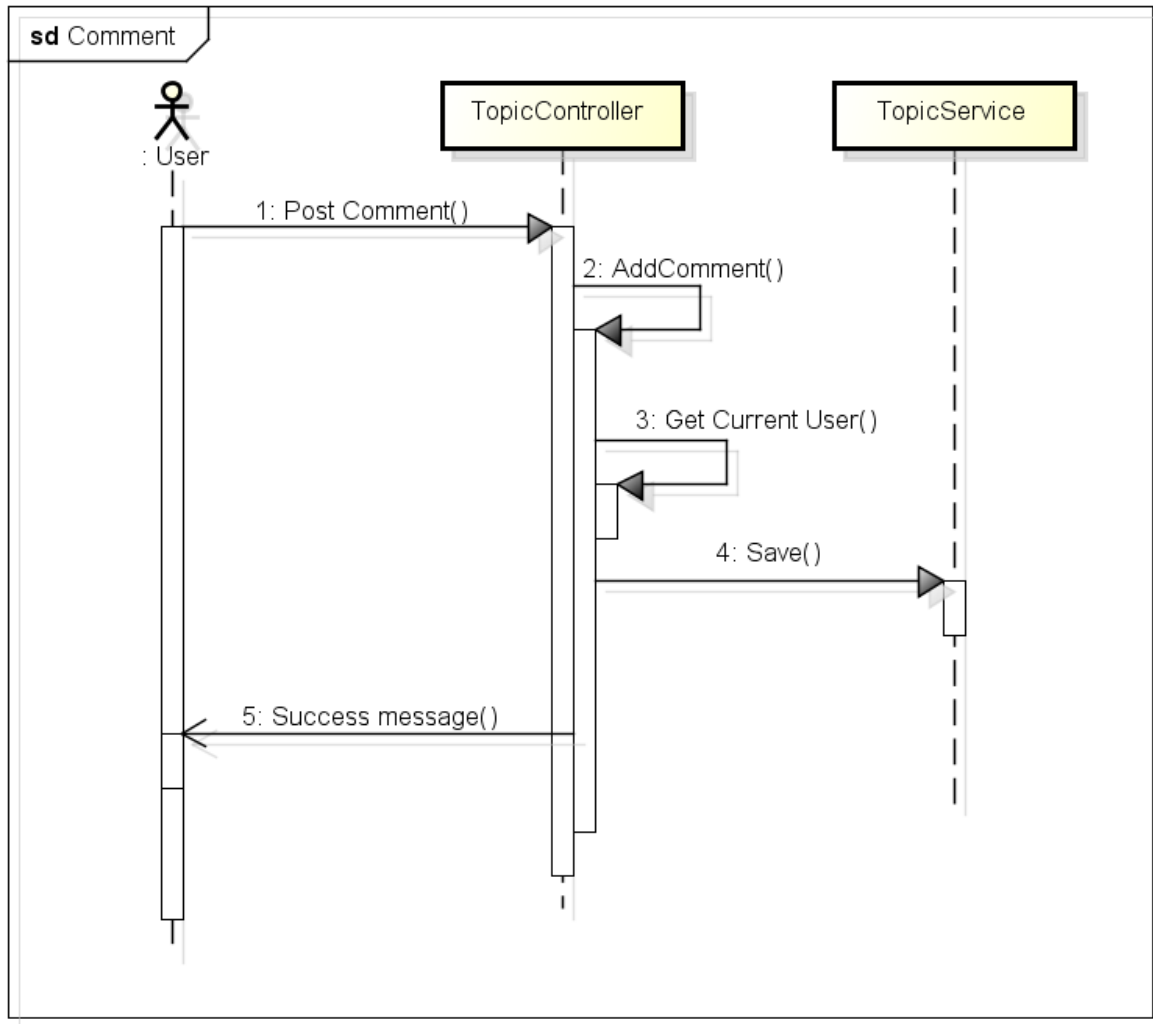


Sơ đồ 5.16: Sequence diagram của chức năng Accept Member

5.5.3.3 Quản lý Topic

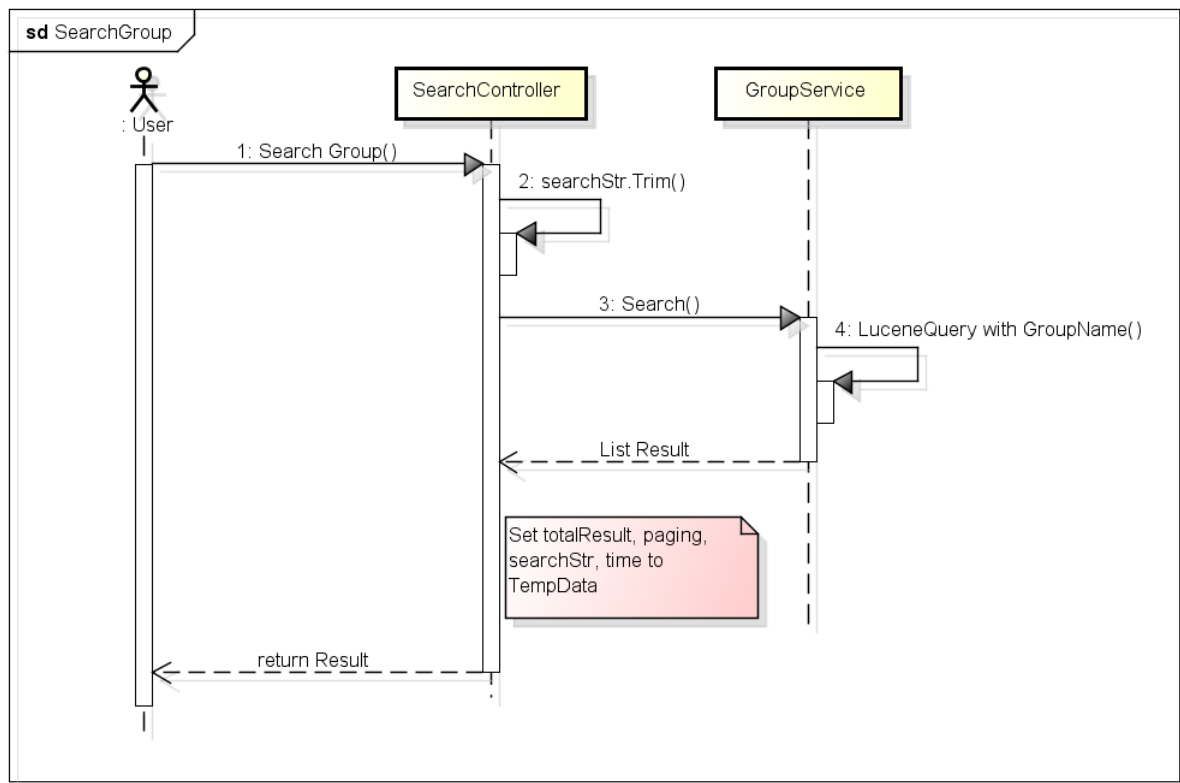


Sơ đồ 5.17: Sequence diagram của chức năng Create Topic



Sơ đồ 5.18: Sequence diagram của chức năng đăng bình luận

5.5.3.4 Tìm kiếm



Sơ đồ 5.19: Sequence diagram của chức năng tìm kiếm

5.6 Thiết kế giao diện

5.6.1 Danh sách màn hình

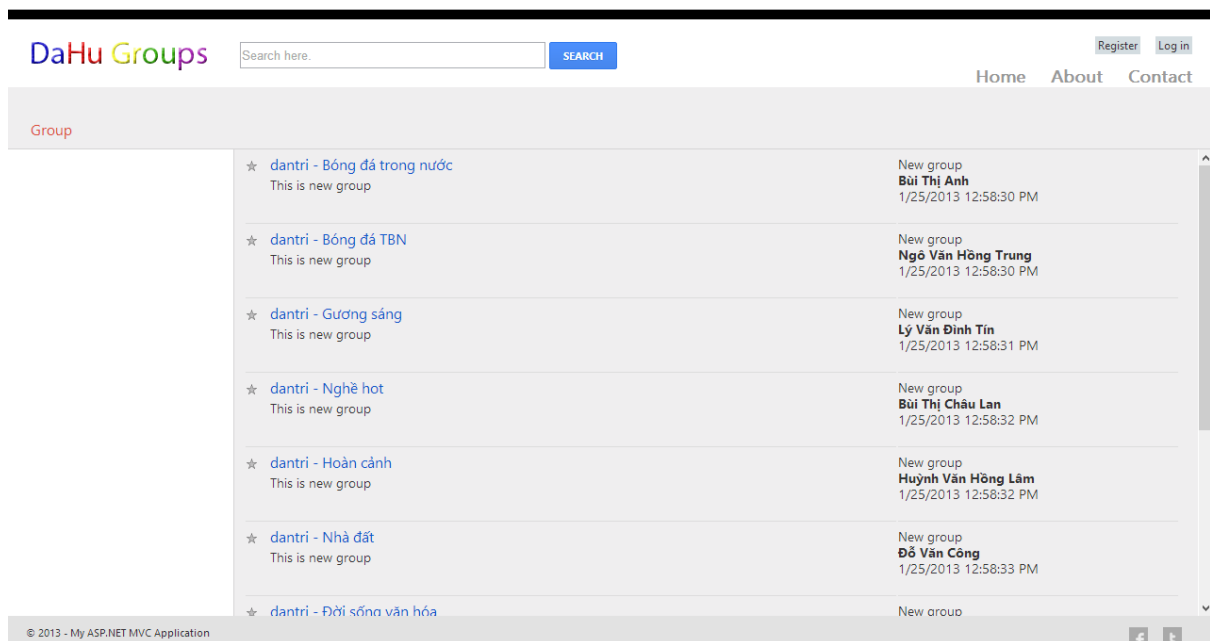
STT	Tên màn hình	Chức năng
1	Màn hình giao diện chính	Cung cấp cho người dùng cái nhìn tổng quan về chương trình. Tại màn hình này, ta có thể thực hiện các thao tác thường dùng như: Login, Register, Create Group...
2	Màn hình đăng nhập	Cho phép người dùng đăng nhập vào hệ thống
3	Màn hình đăng ký tài khoản	Cho phép người dùng mới đăng ký tài khoản
4	Màn hình tạo mới nhóm	Cho phép người dùng tạo mới một nhóm sau khi đăng nhập vào hệ thống
5	Màn hình tạo mới bài viết	Cho phép người dùng tạo mới bài viết
6	Màn hình danh sách bài viết	Hiển thị danh sách bài viết. Có thể quản lý danh sách bài viết nếu vai trò là Owner hay Manager
7	Màn hình bài viết và tất cả bình luận	Hiển thị một bài viết cụ thể và tất cả các bình luận của nó. Cho phép người dùng đăng bình luận
8	Màn hình upload file	Cho phép thành viên upload file khi đăng bài viết mới hoặc đăng bình luận
9	Màn hình cài đặt group	Cung cấp các chức năng cài đặt chính của group
10	Màn hình quản lý user	Hiển thị danh sách user đã tham gia group, có thể xóa thành viên ra khỏi nhóm.
11	Màn hình thêm thành viên nhóm	Owner hoặc manager của nhóm thêm thành viên mới vào nhóm
12	Màn hình tùy chỉnh cài đặt trong nhóm	Cho phép thành viên trong nhóm thay đổi cài đặt trong nhóm đó (như có nhận mail không...)
13	Màn hình tìm kiếm	Cho phép người dùng thực hiện việc tìm kiếm nhóm

14	Màn hình đổi mật khẩu	Thay đổi mật khẩu của user
15	Màn hình thay đổi thông tin cá nhân	Cho phép người dùng thay đổi thông tin cá nhân

Bảng 5.18: Danh sách màn hình

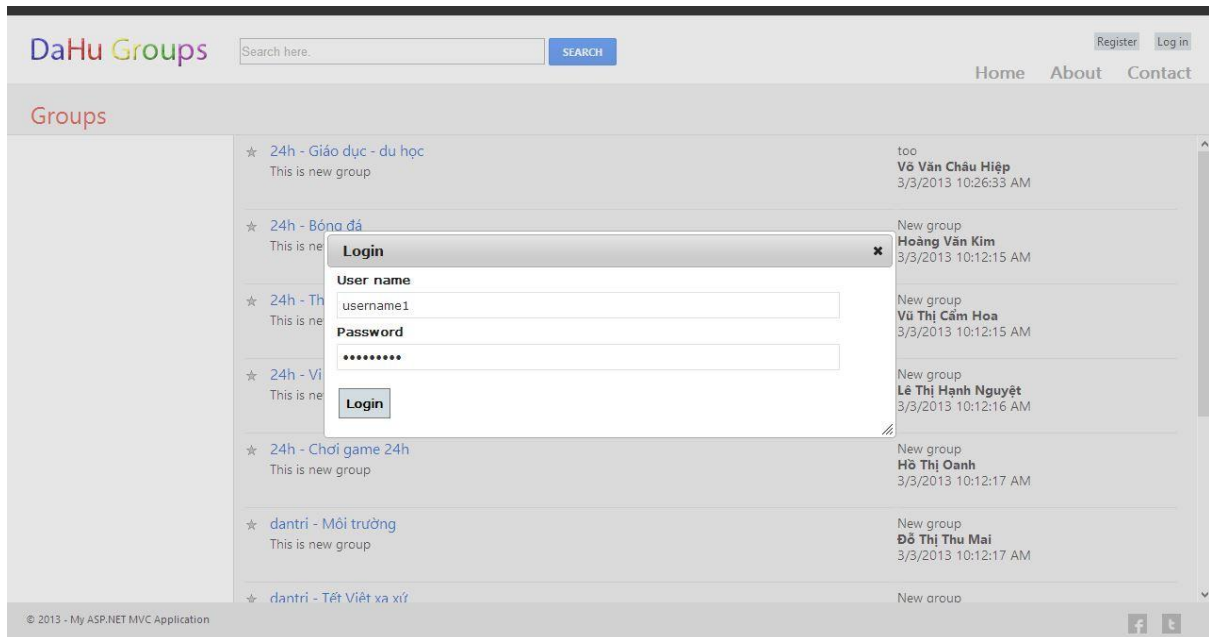
5.6.2 Mô tả giao diện người dùng

5.6.2.1 Màn hình giao diện chính



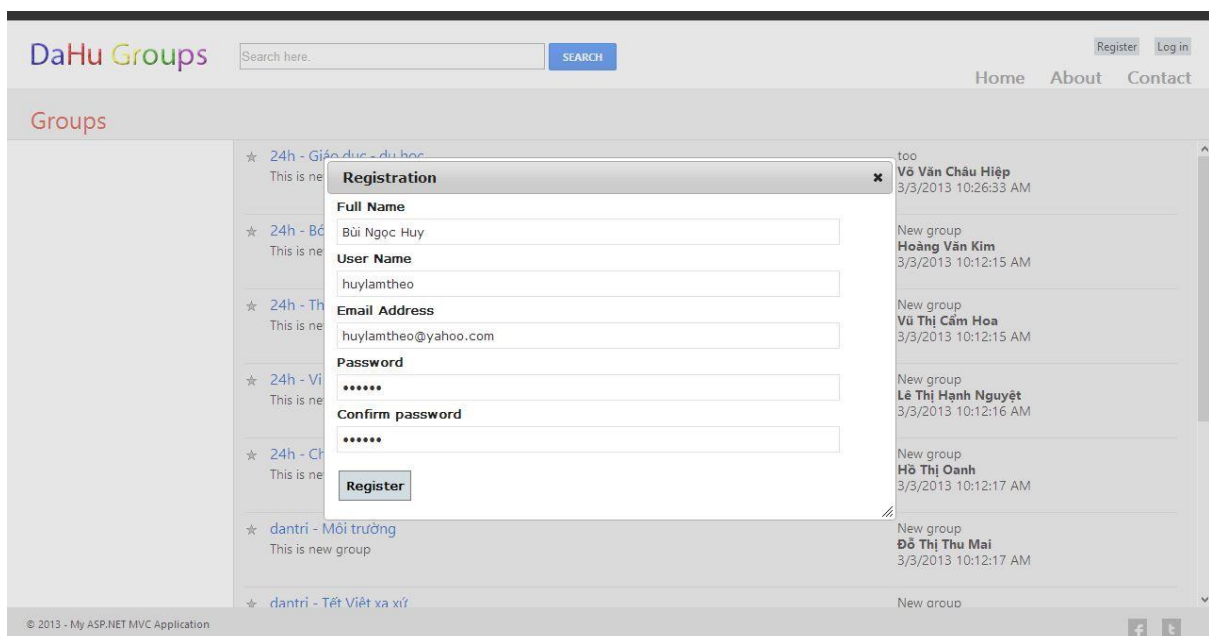
Hình 5.1: Màn hình chính của chương trình

5.6.2.2 Màn hình đăng nhập



Hình 5.2: Màn hình đăng nhập

5.6.2.3 Màn hình đăng ký tài khoản



Hình 5.3: Màn hình đăng ký tài khoản

5.6.2.4 Màn hình tạo mới nhóm

The screenshot shows the 'Create New Group' interface. On the left is a sidebar with navigation links: 'My Group', 'Home', 'Share', '24h - Bóng đá', 'Favorites', and 'Favorites'. The main content area is titled 'Groups' and contains a form with two input fields: 'Group Name' and 'Description'. Below these fields is a 'Create' button. At the top of the page, there is a header with the 'DaHu Groups' logo, a search bar, and a 'SEARCH' button. On the right side of the header, it says 'Hello, Hoàng Văn Kim !' with a 'Log off' link. Below the header are links for 'Home', 'About', and 'Contact'. At the bottom of the page, there is a footer with the text '© 2013 - My ASP.NET MVC Application' and social media icons for Facebook and Twitter.

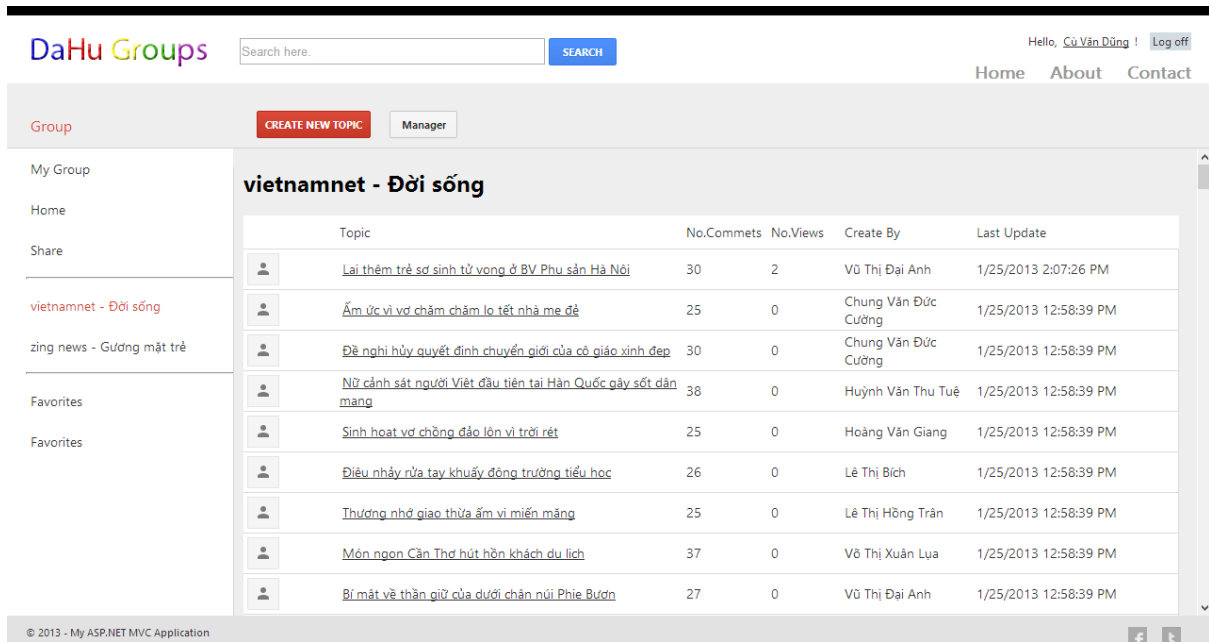
Hình 5.4: Màn hình tạo mới nhóm

5.6.2.5 Màn hình tạo mới bài viết

The screenshot shows the 'Create New Post' interface. At the top, there are buttons for 'POST TOPIC' and 'Cancel'. Below these, the selected group is 'vietnamnet - Đời sống' and the creator is 'Lại Thị Thu Bích'. The form has two main sections: 'Topic Name' with an input field, and 'Content' with a rich text editor. The rich text editor has a toolbar with various formatting options like bold, italic, underline, and link. At the bottom of the form is a 'POST' button. The top navigation bar is identical to the one in Figure 5.4, but the user is now logged in as 'Cù Văn Dũng'. The footer is also the same.

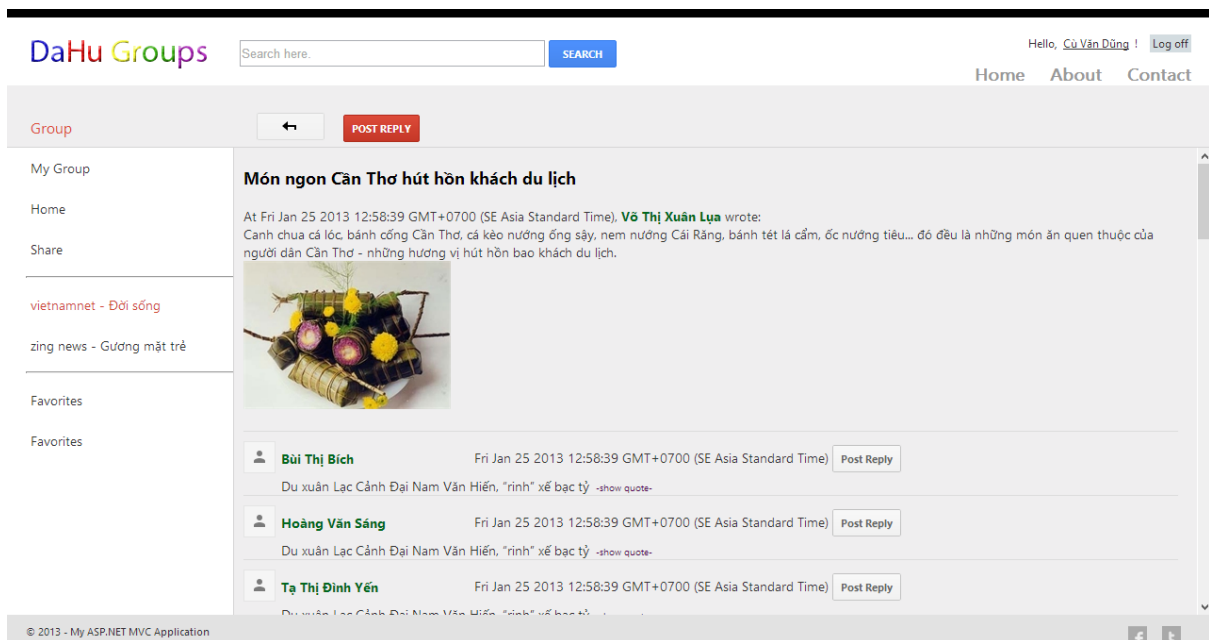
Hình 5.5: Màn hình tạo mới bài viết

5.6.2.6 Màn hình danh sách bài viết



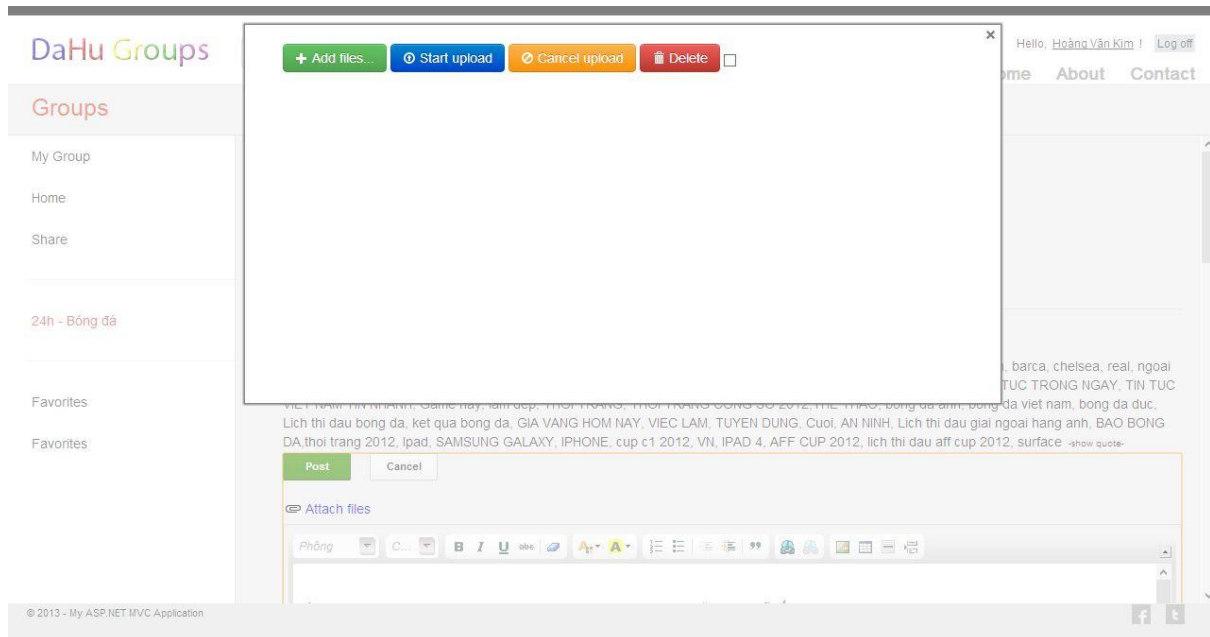
Hình 5.6: Màn hình danh sách bài viết

5.6.2.7 Màn hình bài viết và tất cả bình luận



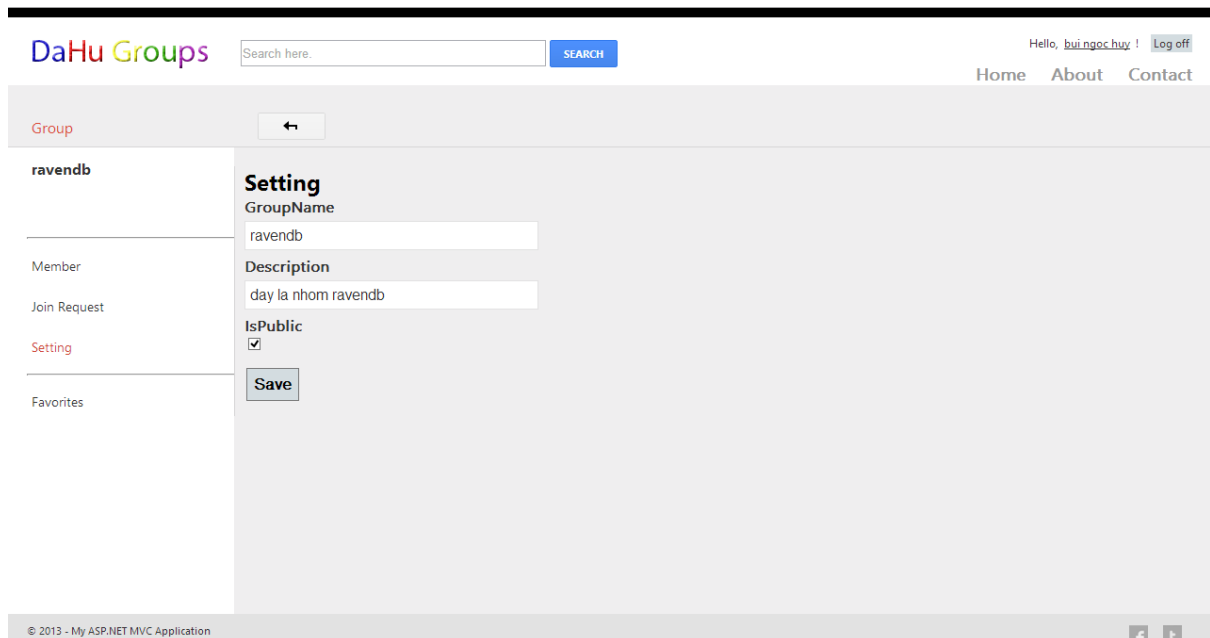
Hình 5.7: Màn hình bài viết và tất cả bình luận

5.6.2.8 Màn hình upload file



Hình 5.8: Màn hình upload file

5.6.2.9 Màn hình cài đặt Group



Hình 5.9: Màn hình cài đặt Group

5.6.2.10 Màn hình quản lý user

DaHu Groups Search here. [SEARCH](#) Hello, [Hoàng Văn Kim](#) ! [Log off](#) [Home](#) [About](#) [Contact](#)

Groups [←](#)

24h - Bóng đá

List Members

User Name	Email Address	Full Name	Role	Remove
username1822	username1822@gmail.com	Hoàng Văn Giang	Member Change	Remove
username217	username217@gmail.com	Ngô Thị Châu Văn	Member Change	Remove
username5653	username5653@gmail.com	Ngô Văn Thông	Member Change	Remove
username5490	username5490@gmail.com	Đỗ Thị Đại Anh	Member Change	Remove
username8431	username8431@gmail.com	Ngô Thị Thu Cúc	Member Change	Remove

© 2013 - My ASP.NET MVC Application [f](#) [t](#)

Hình 5.10: Màn hình quản lý user

5.6.2.11 Màn hình thêm thành viên nhóm

DaHu Groups Search here. [SEARCH](#) Hello, [Hoàng Văn Kim](#) ! [Log off](#) [Home](#) [About](#) [Contact](#)

Groups [←](#)

24h - Bóng đá

Join Request

User Name	Accept Member	Reject
username1	Accept Member	Reject
username2	Accept Member	Reject
username3	Accept Member	Reject

© 2013 - My ASP.NET MVC Application [f](#) [t](#)

Hình 5.11: Màn hình thêm thành viên nhóm

5.6.2.12 Màn hình tùy chỉnh cài đặt trong nhóm

The screenshot shows the 'DaHu Groups' application interface. At the top, there is a search bar and a 'SEARCH' button. The user is logged in as 'Hoàng Văn Kim' and can click 'Log off'. The navigation menu includes 'Home', 'About', and 'Contact'. The main content area is titled 'Groups' and shows a list of groups on the left. The selected group is '24h - Bóng đá'. The 'ManagerUser' settings for this group are displayed on the right. The settings include:

- GroupName:** 24h - Bóng đá
- Description:** This is new group
- JoinDate:** 3/3/2013 10:12:15 AM
- IsReceiveEmail:** ☒
- Save:** A button to save the changes.
- Back to List:** A link to return to the group list.

 The footer shows the copyright notice '© 2013 - My ASP.NET MVC Application' and social media icons for Facebook and Twitter.

Hình 5.12: Màn hình tùy chỉnh cài đặt trong nhóm

5.6.2.13 Màn hình tìm kiếm

The screenshot shows the 'DaHu Groups' application interface with the search bar containing the text '24h'. The search results are displayed in a table. The table has the following columns:

- Group:** The name of the group, preceded by a star icon.
- Description:** A brief description of the group.
- User:** The name of the user who created the group.
- Date:** The date and time when the group was created.

 The search results are as follows:

Group	Description	User	Date
★ 24h - Y tế - thiết bị	This is new group	Lê Văn Hiệp	1/25/2013 12:58:29 PM
★ 24h - Thời trang Hi-tech	This is new group	Lý Văn Châu Lâm	1/25/2013 12:58:28 PM
★ 24h - Cưới suốt 24giờ	This is new group	Đặng Thị Trân	1/25/2013 12:58:29 PM
★ 24h - Phi thường - kỳ quặc	This is new group	Nguyễn Văn Xuân Lễ	1/25/2013 12:58:28 PM
★ 24h - Thị trường - Tiêu dùng	This is new group	Phan Văn Lâm	1/25/2013 12:58:29 PM
★ 24h - Thời trang	This is new group	Lê Thị Ngọc Diễm	1/25/2013 12:58:28 PM

 The footer shows the copyright notice '© 2013 - My ASP.NET MVC Application' and social media icons for Facebook and Twitter.

Hình 5.13: Màn hình tìm kiếm

5.6.2.14 Màn hình thay đổi mật khẩu

DaHu Groups Search here. SEARCH Hello, Võ Văn Châu Hiệp ! Log off Home About Contact

Groups

My Group

Home

Share

24h - Giáo dục - du học

24h - Thị trường - Tiêu dùng

Favorites

Favorites

Change Password. Use the form below to change your password.

New passwords are required to be a minimum of 6 characters in length.

Current password

New password

Confirm new password

Change password

© 2013 - My ASP.NET MVC Application

Hình 5.14: Màn hình thay đổi mật khẩu

5.6.2.15 Màn hình thay đổi thông tin cá nhân

DaHu Groups Search here. SEARCH Hello, Võ Văn Châu Hiệp ! Log off Home About Contact

Groups

My Group

Home

Share

24h - Giáo dục - du học

24h - Thị trường - Tiêu dùng

Favorites

Favorites

Account setting

UserName	username729
Email	username729@gmail.com
FullName	Võ Văn Châu Hiệp
CreateDate	3/3/2013 10:12:15 AM
Region	Asia

[Change Password](#)

© 2013 - My ASP.NET MVC Application

Hình 5.15: Màn hình thay đổi thông tin cá nhân

CHƯƠNG 6 – KẾT LUẬN

6.1 Kết quả đạt được

- Về mặt lý thuyết:

- Tổng hợp và phân tích khá chi tiết về cơ sở dữ liệu NoSQL cùng với những ứng dụng thực tiễn của nó. Qua tài liệu này, người đọc có được cái nhìn bao quát về NoSQL và có thể ứng dụng nó vào các hệ thống cần lưu trữ rất nhiều dữ liệu.
- Trình bày và phân tích những loại khác nhau của NoSQL bao gồm: key-value store, column families, document database và graph database.
- Tìm hiểu về tính năng, đặc điểm và những lợi ích của một số loại document database phổ biến là: MongoDB, CouchDB, RavenDB.
- Đào sâu kiến thức về RavenDB và các triển khai một ứng dụng sử dụng RavenDB .
- Bên cạnh đó, chúng em còn tìm hiểu thêm một số công nghệ phát triển ứng dụng web hiệu quả như: ASP.NET MVC4, LINQ, JQuery.

- Về mặt thực nghiệm: Xây dựng được ứng dụng DaHu Groups (có các chức năng cơ bản giống Google Groups) giúp cho người dùng có thể tạo nhóm và trao đổi các vấn đề liên quan về nhóm. Ứng dụng được xây dựng trên nền web và sử dụng cơ sở dữ liệu RavenDB. Qua quá trình nghiên cứu và phát triển thì ứng dụng của chúng em đã đạt được những kết quả sau:

- Ứng dụng cho phép người dùng tạo nhóm thảo luận, đăng các bài viết liên quan đến nhóm, đăng bình luận cho các bài viết. Ứng dụng cũng cho phép người dùng upload file trong quá trình đăng bài và cho download file khi bài viết hay bình luận được đăng lên nhóm.
- Khi một nhóm có bài đăng mới thì hệ thống sẽ gửi mail thông báo đến tất cả thành viên trong nhóm về bài đăng này gồm nội dung bài đăng, người tạo, ngày tạo. Khi đăng nhập vào hệ thống, người dùng cũng được thông báo về bài viết mới nhất của mỗi nhóm, số lượng bài đăng mới mà người dùng chưa đọc của mỗi topic.

- Người dùng có quyền owner (chủ nhóm) và manager (quản lý nhóm) có thể quản lý các thành viên trong nhóm bao gồm: chấp nhận hoặc từ chối yêu cầu tham gia nhóm của người dùng, có thể xóa thành viên ra khỏi nhóm. Owner có thể chuyển quyền thành viên thành quyền quản lý. Bên cạnh đó, chủ nhóm hoặc quản lý có thể quản lý bài đăng của các thành viên trong nhóm(có thể xóa các bài đăng trong nhóm hoặc xóa các bình luận trong bài viết).
- Hệ thống cho phép người dùng tạo mới tài khoản cũng như cập nhật khi có thông tin thay đổi. Khi người dùng trở thành thành viên của một nhóm nào đó thì có thể thay đổi thiết lập riêng tư cho riêng nhóm đó (như có nhận e-mail từ hệ thống khi có bài đăng mới không...).
- Ứng dụng có giao diện web đơn giản, trực quan, dễ sử dụng.

6.2 Hướng phát triển

Ứng dụng hiện tại đã cung cấp được nhiều tính năng cơ bản của một ứng dụng thảo luận nhóm (giống google groups). Tuy nhiên, còn nhiều chức năng, cài đặt nâng cao mà ứng dụng chưa hoàn thiện được. Do đó, ứng dụng cần được phát triển thêm các chức năng nâng cao như: đánh dấu sao bài đăng, phân loại bài đăng, duyệt nội dung bài viết, cho phép tùy chọn email. Ngoài ra, chúng ta cũng cần nâng cấp hệ thống ở phía máy chủ như:

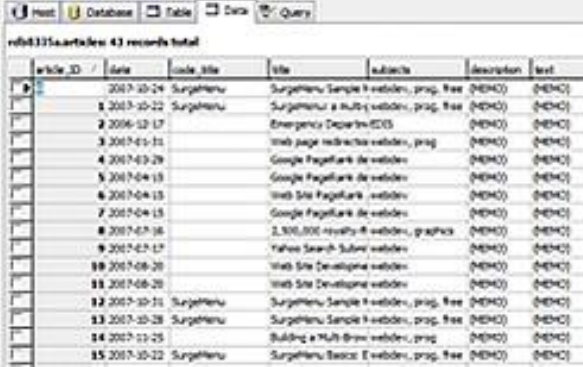
- Tìm hiểu thêm vấn đề phân tán dữ liệu ở nhiều máy chủ.
- Tìm hiểu thêm các vấn đề quản lý transaction. Đặc biệt xử lý tình huống cơ sở dữ liệu phân tán trên nhiều máy chủ.
- Một số vấn đề như bảo mật, config server, backup và restore dữ liệu trên cơ sở dữ liệu NoSQL.

PHỤ LỤC

7.1 Tính năng đầy đủ của RavenDB

- **Safe by default:** RavenDB đảm bảo an toàn cho việc truy cập dữ liệu. Không tiêu tốn tài nguyên mạng và hệ thống. Xây dựng ứng dụng với RavenDB, tốc độ chạy chương trình nhanh và đáng tin cậy.
- **Transactional:** Hỗ trợ đầy đủ ACID transactions (Atomicity, Consistency, Isolation, Durability) ngay cả những node khác nhau trong hệ thống.
- **Scalable:** Hỗ trợ Sharding, Replication, Multi-Tenancy. Scaling out (mở rộng theo chiều ngang) tương đối dễ dàng.
 - Build-in Sharding: phân tán dữ liệu trên nhiều server khác nhau để quản lý việc load dữ liệu tốt hơn.
 - Buil-in Replication: nhân bản dữ liệu trên nhiều server để tăng tính sẵn sàng và lấy dữ liệu nhanh chóng.
 - Mix replication and sharding: Có thể sử dụng kết hợp cả 2 tính năng Replication và Sharding
- **Schema free:** Bỏ qua những khái niệm tables, rows, mappings, complex data-layers. RavenDB là cơ sở dữ liệu hướng tài liệu, vì thế có thể lưu trữ cả đối tượng dữ liệu.
- **Get running in 5 minutes:** Chỉ cần 5 phút là đã có thể sử dụng RavenDB. RavenDB không yêu cầu cài đặt phức tạp, chỉ tải về và chạy. Rất đơn giản.
- **It Just Work:** Thiết kế để làm việc
- **Fast Queries:** RavenDB có thể thực hiện bất kì truy vấn với tốc độ cực nhanh(tốc độ ánh sáng). Tất cả thao tác indexing được thực hiện nền (thực hiện ngầm), không ảnh hưởng đến truy vấn, thao tác đọc viết từ database.
- **Best practices built-in**
 - Unit Of Work: thay đổi dữ liệu bằng cách thay đổi đối tượng nhận được từ Client API.

- Domain Driven Design: mô hình dữ liệu sử dụng khái niệm DDD để thao tác dữ liệu tốt nhất.
 - In-memory DB for testing
 - Automatic-batching: tự tối ưu bằng cách gửi đi một tập lệnh thay vì một lệnh đơn.
- **High performance:** RavenDB lưu trữ rất nhanh tất cả mô hình dữ liệu. Bỏ qua giai đoạn mapping phức tạp hay đa tầng DAL, chỉ đơn giản là lưu trữ những thực thể.
 - **Caching built-in:** Nhiều tầng caches thực hiện tự động trên cả server và client. Caching được cấu hình sẵn và có chế độ nâng cao là Aggressive Caching.
 - **APIs:** Có thể truy cập RavenDB bằng nhiều ngôn ngữ hay công nghệ khác nhau. Giao tiếp Client/Server thông qua REST (HTTP API), .NET client API, Silverlight and Javascript.
 - **Built-in managemet studio:** Dễ dàng quản lý dữ liệu với giao diện đồ họa trực quan.
 - **Carefully designed:** RavenDB được thiết kế rất cẩn thận, tỉ mỉ đảm bảo mọi thứ hoạt động tốt.
 - **Map/Reduce:** Sử dụng indexes, dễ dàng viết các hàm Map/Reduce sử dụng cú pháp Linq. Hỗ trợ khái niệm multi-maps và boosting indexes để viết Map/Reduce đơn giản hơn và thể hiện sức mạnh của nó.
 - **Feature rich and extensible:** Hỗ trợ nhiều tính năng và khả năng mở rộng.
 - **Embededable:** RavenDB có thể nhúng vào bất kỳ ứng dụng .NET, và nó cũng hoàn toàn phù hợp với các ứng dụng desktop.
 - **Bundles:** Nhiều gói dữ liệu hỗ trợ đi kèm với Server-side plugins. Chỉ cần copy file DLL vào thư mục Server.
 - **Index replication to SQL:** sử dụng ưu điểm của công cụ reporting có sẵn từ RDBMS. RavenDB cho phép nhân bản index sang SQL table dễ dàng.



article_ID	date	code	title	subject	description	text
1	2007-10-24	SurgeMenu	SurgeMenu Sample 1 webdev, prog, free	(NEMO)	(NEMO)	
2	2007-10-22	SurgeMenu	SurgeMenu a Multi webdev, prog, free	(NEMO)	(NEMO)	
3	2006-12-17		Emergency Document EODS	(NEMO)	(NEMO)	
4	2007-03-11		Web page redirection webdev, prog	(NEMO)	(NEMO)	
5	2007-03-29		Google PageRank de webdev	(NEMO)	(NEMO)	
6	2007-04-13		Google PageRank de webdev	(NEMO)	(NEMO)	
7	2007-04-15		Web Site PageRank de webdev	(NEMO)	(NEMO)	
8	2007-04-15		Google PageRank de webdev	(NEMO)	(NEMO)	
9	2007-07-16		2,300,000 results de webdev, graphics	(NEMO)	(NEMO)	
10	2007-07-17		Yahoo Search Submit webdev	(NEMO)	(NEMO)	
11	2007-08-20		Web Site Development webdev	(NEMO)	(NEMO)	
12	2007-08-20		Web Site Development webdev	(NEMO)	(NEMO)	
13	2007-10-21	SurgeMenu	SurgeMenu Sample 2 webdev, prog, free	(NEMO)	(NEMO)	
14	2007-10-28	SurgeMenu	SurgeMenu Sample 3 webdev, prog, free	(NEMO)	(NEMO)	
15	2007-11-25		Building a Multi-Branch webdev, prog	(NEMO)	(NEMO)	
16	2007-10-22	SurgeMenu	SurgeMenu Basic E webdev, prog, free	(NEMO)	(NEMO)	

Hình 7.1: Index replication to SQL

- **Full-text Search built-in:** Không cần sử dụng công cụ hỗ trợ tìm kiếm nâng cao bên ngoài, RavenDB hỗ trợ tìm kiếm full-text ở server và client API.
- **Advances search techniques**
- **Geo-spatial search support:** Dễ dàng sử dụng API này.



Hình 7.2: Geo-spatial search support

- **Easy backup:** Việc lưu trữ bất đồng bộ mà không làm ảnh hưởng đến thao tác DB thông thường. Backup và Restore đều được hỗ trợ bởi DB.
- **Multi-tenancy:** Lưu trữ nhiều database trên một RavenDB Server.



Hình 7.3: Multi-tenancy

- **Attachments:** RavenDB hỗ trợ lưu trữ luồng dữ liệu mà không thực sự là dữ liệu như hình ảnh hay dữ liệu nhị phân mà chúng ta không muốn lưu trữ như một document, nhưng vẫn có thể lưu trữ.
- **Online index Rebuild:** Indexes được update ngầm bên dưới mà không cần tác động của người dùng hay bất kì thao tác ACID của cơ sở dữ liệu.
- **Fully async (C# 5 ready):** RavenDB hỗ trợ API bất đồng bộ mới được giới thiệu bởi C#5
- **Community**
- **Cloud hosting available:** Chạy RavenDB trên đám mây với RavenHQ, CloudBird, AppHorbor hoặc Windows Azure.

7.2 Quản lý mối quan hệ giữa các document

Với RavenDB, một trong những nguyên tắc khi thiết kế database là làm cho các documents độc lập nhau, có nghĩa là tất cả thông tin được yêu cầu khi xử lý một document được lưu trữ toàn bộ trong document đó. Tuy nhiên điều đó không có nghĩa là không có các mối quan hệ (relations) giữa các đối tượng. Có những trường hợp mà chúng ta cần phải xác định mối quan hệ giữa các đối tượng và ta sẽ gặp một vấn đề lớn là: bất cứ khi nào chúng ta nạp dữ liệu đối tượng chứa (container object) thì chúng ta cần phải nạp dữ liệu cho những thực thể tham chiếu (referenced entities). So với việc lưu toàn bộ thông tin cần thiết vào một thực thể thì việc tham chiếu đến thực thể có vẻ như đỡ tốn chi phí lúc đầu, nhưng điều này được chứng minh là khá tốn kém về tài nguyên dữ liệu và lưu lượng truy cập mạng.

RavenDB cung cấp 3 phương pháp để giải quyết vấn đề này. Mỗi trường hợp sẽ sử dụng một hoặc nhiều phương pháp. Khi áp dụng một cách chính xác, các phương pháp này sẽ làm cải thiện hiệu suất, giảm băng thông và tăng tốc độ phát triển một cách đáng kể.

7.2.1 Phương pháp 1: Denormalization (Phi chuẩn hóa)

Cách dễ nhất để giải quyết vấn đề này là phi chuẩn hóa dữ liệu (denormalization data) vào trong thực thể chứa. Thực thể chứa sẽ chứa những dữ liệu thay cho dữ liệu tham chiếu tới. Xem một ví dụ về một JSON document:

```

{ // Order document with id: orders/1234
  "Customer": {
    "Name": "Itamar",
    "Id": "customers/2345"
  },
  "Items": [
    {
      "Product": {
        "Id": "products/1234",
        "Name": "Milk",
        "Cost": 2.3
      },
      "Quantity": 3
    }
  ]
}

```

Document Order đã chứa dữ liệu đã được phi chuẩn hóa của 2 documents là Customer và Product. Thông tin đầy đủ của Customer và Product documents được lưu trữ ở một nơi khác. Lưu ý là chúng ta sẽ không copy toàn bộ thuộc tính của Customer vào trong Order, chúng ta chỉ copy những thuộc tính của Customer mà chúng ta quan tâm khi cần hiển thị hay xử lý với các Order. Cách tiếp cận này được gọi là “**denormalized reference**”.

Cách tiếp cận denormalization này giúp chúng ta tránh việc tìm kiếm chéo dữ liệu và chỉ những kết quả cần thiết mới được truyền tải qua mạng nhưng nó lại làm cho một số trường hợp khác trở nên khó khăn. Ví dụ, lúc đầu chúng ta có những thực thể có cấu trúc như bên dưới:

```

public class Order
{
    public string CustomerId { get; set; }
    public string[] SupplierIds { get; set; }
    public Referral Referral { get; set; }
    public LineItem[] LineItems { get; set; }
    public double TotalPrice { get; set; }
}

public class Customer
{

```

```

    public string Name { get; set; }
    public string Address { get; set; }
    public short Age { get; set; }
    public string HashedPassword { get; set; }
}

```

Nếu chúng ta biết rằng khi nào chúng nạp dữ liệu cho Order từ cơ sở dữ liệu, chúng ta cũng cần nạp dữ liệu cho Customer Name và Customer Address, chúng ta có thể tạo ra một trường dữ liệu phi chuẩn hóa Order.Customer và lưu thông tin này trực tiếp vào trong đối tượng Order. Customer Password và những thông tin không cần thiết khác sẽ không được phi chuẩn hóa:

```

public class DenormalizedCustomer
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }
}

```

Đây không phải là tham chiếu trực tiếp giữa Order và Customer. Order sẽ chứa một DenormalizedCustomer (chứa những thông tin cần thiết từ Customer khi chúng ta xử lý đối tượng Order).

Chuyện gì sẽ xảy ra nếu như Customer's Address thay đổi? Chúng ta phải thực hiện một loạt các thao tác để cập nhật lại tất cả các Order mà Customer này đã thực hiện. Và điều gì sẽ xảy ra nếu Customer này có rất nhiều Orders hoặc là địa chỉ của họ thường xuyên thay đổi? Việc giữ cho các thông tin được đồng bộ là yêu cầu cấp thiết trên máy chủ. Điều gì sẽ xảy ra nếu một quá trình làm việc khác cần một tập các thuộc tính khác của Customer ? DenormalizedCustomer cần phải được mở rộng, và như thế thì một số lượng lớn Customer record được nhân bản.

7.2.2 Phương pháp 2: Includes

Tính năng “Include” trong RavenDB nhắm đến sự hạn chế của phương pháp “Denormalization”. Thay vì đối tượng chính copy những thông tin từ những đối tượng khác thì nó chỉ cần giữ tham chiếu đến đối tượng cần quan tâm. Khi đó

RavenDB có thể tải dữ liệu trước (pre-load) cho đối tượng được tham chiếu đến vào thời điểm mà đối tượng chính được nạp dữ liệu. Làm điều này như sau:

```
var order = session.Include<Order>(x => x.CustomerId)
    .Load("orders/1234");
// câu lệnh tiếp theo không yêu cầu truy vấn tới server!
var cust = session.Load<Customer>(order.CustomerId);
```

Với đoạn code trên, chúng ta yêu cầu RavenDB lấy dữ liệu đối tượng Order “order/1234” và cùng lúc đó thì nó sẽ “include” Customer vào Order được tham chiếu bởi thuộc tính Order.CustomerId. Phương thức Load() thứ 2 sẽ được thực hiện hoàn toàn ở phía client (không cần một yêu cầu thứ 2 tới RavenDB) bởi vì đối tượng Customer thích hợp đã được nạp dữ liệu (đây là một đối tượng Customer đầy đủ, không phải là một DenormalizedCustomer). Chúng ta cũng có thể sử dụng “Includes” với truy vấn sau:

```
var orders = session.Query<Order>()
    .Customize(x => x.Include<Order>(o => o.CustomerId))
    .Where(x => x.TotalPrice > 100)
    .ToList();
foreach (var order in orders)
{
    // không yêu cầu truy vấn tới server!
    var cust = session.Load<Customer>(order.CustomerId);
}
```

RavenDB có 2 kênh xuyên suốt mà nó trả về thông tin cho một yêu cầu nạp dữ liệu. Thứ nhất là kênh Results, đối tượng chính sẽ được nạp dữ liệu bởi phương thức Load(). Thứ 2 là kênh Includes, những documents được include sẽ được gửi về phía client. Ở phía Client, những documents được include không được trả về bởi phương thức Load(), bởi vì nó đã được thêm vào session unit of work và những yêu cầu tiếp theo để nạp dữ liệu sẽ được làm bởi session cache, không có bất cứ yêu cầu truy vấn nào gửi đến server nữa.

7.2.3 Live Projections

Sử dụng Include thì rất hữu ích, nhưng nhiều lúc chúng ta muốn làm những thao tác phức tạp hơn. Tính năng Live Projection là duy nhất trong RavenDB và nó có thể được coi là bước thứ 3 trong thao tác Map/Reduce: sau khi mapped (ánh xạ) tất cả dữ liệu và nó đã được reduced (nếu index là Map/Reduce index), RavenDB server có thể chuyển đổi kết quả sang một cấu trúc hoàn toàn khác và trả về kết quả này thay vì kết quả gốc. Sử dụng Live Projection bạn có quyền kiểm soát nhiều hơn đối với những gì nạp vào các thực thể kết quả, và vì nó trả về projection (phép chiếu) của kết quả ban đầu, chúng ta có thể lọc ra những thuộc tính không cần thiết.

Lợi ích chính của việc sử dụng Live Projection là chúng ta không cần phải viết nhiều code, nó được thực thi ở phía server và nó tốn ít băng thông mạng bằng cách trả về những dữ liệu mà chúng ta quan tâm.

Lưu ý: một điểm khác biệt quan trọng là Include hữu dụng trong cả 2 trường hợp nạp dữ liệu bởi id và truy vấn dữ liệu, còn Live Projection chỉ được sử dụng cho truy vấn dữ liệu.

7.2.4 Phương pháp kết hợp

Chúng ta có thể sử dụng kết hợp các phương pháp trên. Sử dụng DenormalizedCustomer ở phần trên và tạo ra Order để sử dụng chúng:

```
public class Order3
{
    public DenormalizedCustomer Customer { get; set; }
    public string[] SupplierIds { get; set; }
    public Referral Referral { get; set; }
    public LineItem[] LineItems { get; set; }
    public double TotalPrice { get; set; }
}
```

Sử dụng Denormalization, đơn giản và nhanh chóng để load dữ liệu của Order và những thông tin cần thiết của Customer được yêu cầu khi xử lý Order. Và cũng dễ dàng và hiệu quả load đầy đủ object Customer:

```

var order = session.Include<Order3, Customer2>(x => x.Customer.Id)
    .Load(orders/1234);
// this will not require querying the server!

var fullCustomer = session.Load<Customer2>(order.Customer.Id);

```

Sự kết hợp giữa Denormalization và Include có thể sử dụng với List các đối tượng Denormalized.

7.2.5 Kết luận

Không có quy luật cụ thể nào cho việc dùng từng phương pháp trên. Chúng ta nên suy nghĩ theo nhiều hướng khác nhau và xem xét cái hay của từng phương pháp. Ví dụ như trong một ứng dụng thương mại điện tử thì ta phi chuẩn hóa Product Name và Product Price vào trong đối tượng Order Line bởi vì ta muốn chắc chắn là Customer sẽ nhìn thấy đúng Product Name và Product Price trong lịch sử mua hàng. Nhưng Customer Name và Customer Address nên được tham chiếu thay vì phi chuẩn hóa trong Order. Trong những trường hợp mà phi chuẩn hóa không phải là sự lựa chọn thì Include sẽ là phương pháp phù hợp. Và bất cứ khi nào một tiến trình quan trọng được yêu cầu sau khi công việc Map/Reduce được hoàn thành hay khi chúng ta cần một cấu trúc thực thể khác được trả về thì hãy dùng Live Projections.

7.3 Index

7.3.1 Static index

RavenDB cũng cho phép chúng ta tự định nghĩa index và truy vấn nó một cách tường minh. Những index người dùng tự tạo được gọi là static index. Một số lý do mà static index hay được sử dụng hơn những index được tạo tự động là:

- Độ trễ thấp: Tạo index không phải là quá trình ít tốn chi phí, mà nó tốn một thời gian để thực hiện. Vì những dynamic index được tạo cùng với truy vấn đầu tiên nên kết quả non-stale cho lần đầu truy vấn sẽ tốn nhiều thời gian trả về. Dynamic index được tạo như là những index tạm thời, điều này dẫn đến hiệu suất khi thực hiện truy vấn lần đầu.
- Linh hoạt: Static index được hỗ trợ thêm nhiều chức năng khác như sorting, boosting, Full text Search, Live Projection, spatial search support ...

Trong khi sử dụng dynamic index thì sẽ dễ dàng cho chúng ta, việc sử dụng static index thì hữu dụng và hiệu quả hơn với dữ liệu thời gian thực. Vì thế, nên sử dụng static index trong hầu hết các thao tác của chương trình hay ít nhất cũng chắc chắn rằng những index tạm thời được tạo từ những dynamic index sẽ được chỉ định là những index thường dùng.

Bất cứ khi nào chúng ta yêu cầu RavenDB truy vấn dữ liệu và đã có static index thích hợp tồn tại, RavenDB sẽ trực tiếp truy vấn sử dụng index đó một cách tự động. Chúng ta cũng có thể chỉ định tên của index mà chúng ta muốn dùng:

```
var results = session.Query<BlogPost>("MyBlogPostsIndex").ToArray();
```

Lưu ý là RavenDb sẽ ném ra lỗi nếu chúng ta chỉ định tên của index được sử dụng mà index này lại không thực sự tồn tại.

Định nghĩa static index:

- Để định nghĩa một index, chúng ta cần một đối tượng IndexDefinition và đưa nó vào cơ sở dữ liệu. Một index có thể được truy vấn ngay lập tức sau khi quá trình tạo index được bắt đầu, nhưng cho đến lúc quá trình này hoàn thành thì kết quả trả về sẽ được đánh dấu là stale. Index sẽ được cập nhật liên tục khi có bất kì thao tác thêm hay sửa dữ liệu nào.
- Lớp IndexDefinition:
 - Một định nghĩa index bao gồm tên index, hàm map/reduce, một hàm tùy chọn TransformResults và một vài tùy chọn khác. Cấu trúc lớp IndexDefinition được thể hiện bên dưới:

```
class IndexDefinition
{
    public string Name { get; set; }
    public string Map { get; set; }
    public string Reduce { get; set; }
    public string TransformResults { get; set; }

    public IDictionary<string, FieldStorage> Stores { get; set; }
    public IDictionary<string, FieldIndexing> Indexes { get; set; }
}
```

```

public IDictionary<string, SortOptions> SortOptions { get; set; }
public IDictionary<string, string> Analyzers { get; set; }
}

```

- Bất kỳ index nào cũng yêu cầu phải có tên và hàm Map. Hàm Map là cách mà chúng ta thông báo cho RavenDB biết làm thế nào tìm được những dữ liệu chúng ta cần đến và những trường dữ liệu nào mà chúng ta sẽ tìm kiếm. Hàm Map được viết theo cú pháp Linq.
 - Hàm Reduce là một tùy chọn, được viết và thực thi giống như hàm Map nhưng được thực thi trên kết quả của hàm Map. Hàm Reduce thực sự là một index thứ hai cho phép chúng ta thực hiện các thao tác tập hợp ít tốn chi phí và trực tiếp từ index.
 - Hàm thứ ba là hàm TransformResults, một tính năng được gọi là Live Projections, sẽ được nói rõ ở phần sau.
 - Những thuộc tính còn lại hữu ích cho việc tận dụng toàn bộ sức mạnh của Lucene bằng cách tùy biến các indexes.
- Tạo mới một index:
- Dùng hàm PutIndex trong đối tượng DocumentCommands để tạo index:

```

// tạo một index mà chúng ta sẽ tìm kiếm dữ trên thuộc tính Post Title
documentStore.DatabaseCommands.PutIndex(BlogPosts/ByTitles,"

    new IndexDefinitionBuilder<BlogPost>{

        Map = posts => from post in posts

            select new { post.Title }

    });

```

- Có thể tạo một index (index class) bằng cách thừa kế từ AbstractIndexCreationTask<T>. Sau đó thông báo cho server tạo ra index thực sự bằng cách thêm lời gọi vào lúc ứng dụng khởi động (các index đã tồn tại vẫn bị ảnh hưởng):

```

IndexCreation.CreateIndexes(typeof(MyIndexClass).Assembly, documentStore);

```

7.3.2 Stale index (index chứa kết quả cũ, chưa cập nhật)

RavenDB thực hiện việc đánh chỉ mục dữ liệu với một tiến trình nền bên dưới chương trình, nó sẽ được thực thi bất cứ khi nào có dữ liệu mới hoặc dữ liệu cũ được chỉnh sửa, cập nhật. Tiến trình chạy nền bên dưới này cho phép server đáp ứng yêu cầu một cách nhanh chóng ngay cả khi một khối lượng lớn dữ liệu bị thay đổi. Tuy nhiên trong trường hợp này, chúng ta sẽ truy vấn với stale index.

Khái niệm “stale index” xuất phát từ sự nhìn nhận sâu sắc về thiết kế của RavenDB. Việc có một kết quả cũ tốt hơn là việc mất kết nối với dữ liệu (it is better to be stale than offline). Và như vậy, nó sẽ trả về kết quả truy vấn ngay cả khi nó biết là không thể cho một kết quả truy vấn tốt nhất (up-to-date). Và quả thực là RavenDB trả về kết quả nhanh chóng cho bất cứ yêu cầu của người dùng, ngay cả khi liên quan đến việc đánh lại chỉ mục của hàng trăm hàng ngàn documents. Yêu cầu thứ nhất sẽ được server đáp ứng rất nhanh, những truy vấn tiếp theo có thể được thực hiện sau đó vài mili giây và kết quả vẫn được trả về, tuy nhiên nó được đánh dấu là Stale.

Kiểm tra kết quả stale:

- Sử dụng đối tượng `RavenQueryStatistics` để kiểm tra kết quả cũ:

```
RavenQueryStatistics stats;
var results = session.Query<Product>()
    .Statistics(out stats)
    .Where(x => x.Price > 10)
    .ToArray();

if (stats.IsStale)
{
    // Những kết quả cũ
}
```

- Khi giá trị `IsStale` là true thì có nghĩa là có thao tác thêm hoặc thay đổi Product và index không có đủ thời gian để cập nhật lại thay đổi trước khi chúng ta truy vấn.

Lấy kết quả mới(non-stale):

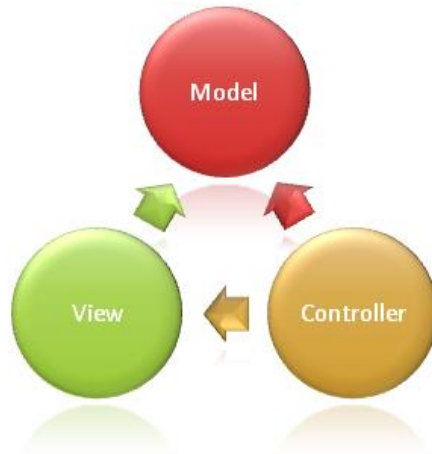
- Với truy vấn yêu cầu lấy kết quả non-stale, ta có thể chỉ định rõ khi truy vấn:

```
RavenQueryStatistics stats;
var results = session.Query<Product>()
    .Statistics(out stats)
    .Where(x => x.Price > 10)
    .Customize(x => x.WaitForNonStaleResults(TimeSpan.FromSeconds(5)))
    .ToArray();
```

- Lưu ý là trên ví dụ trên thì thời gian chờ là 5 giây. Ta có thể yêu cầu RavenDB chờ vô thời hạn cho đến khi nhận được kết quả non-stale, nhưng điều này chỉ nên sử dụng trong unit-testing và không bao giờ dùng trong những ứng dụng thực tế trừ khi chúng ta hiểu 100% về nó hoặc đó là điều chúng ta mong muốn.

7.4 Giới thiệu mô hình ASP.NET MVC4

Mẫu kiến trúc Model – View – Controller được sử dụng nhằm chia ứng dụng thành ba thành phần chính: model, view và controller. Nền tảng ASP.NET MVC giúp cho chúng ta có thể tạo được các ứng dụng web áp dụng mô hình MVC thay vì tạo ứng dụng theo mẫu ASP.NET Web Forms. Nền tảng ASP.NET MVC có đặc điểm nổi bật là nhẹ (lightweight), dễ kiểm thử phần giao diện (so với ứng dụng Web Forms), tích hợp các tính năng có sẵn của ASP.NET. Nền tảng ASP.NET MVC được định nghĩa trong namespace System.Web.Mvc và là một phần của name space System.Web.



Hình 7.4: Mẫu kiến trúc Model – View – Controller

Vài nét về mô hình MVC:

- **Models:** Các đối tượng Models là một phần của ứng dụng, các đối tượng này thiết lập logic của phần dữ liệu của ứng dụng. Thông thường, các đối tượng model lấy và lưu trạng thái của model trong CSDL. Ví dụ như, một đối tượng Product sẽ lấy dữ liệu từ CSDL, thao tác trên dữ liệu và sẽ cập nhật dữ liệu trở lại vào bảng Products ở SQL Server.

Trong các ứng dụng nhỏ, model thường là chỉ là một khái niệm nhằm phân biệt hơn là được cài đặt thực thụ, ví dụ, nếu ứng dụng chỉ đọc dữ liệu từ CSDL và gửi chúng đến view, ứng dụng không cần phải có tầng model và các lớp liên quan. Trong trường hợp này, dữ liệu được lấy như là một đối tượng model (hơn là tầng model).

- **Views:** Views là các thành phần dùng để hiển thị giao diện người dùng (UI). Thông thường, view được tạo dựa vào thông tin dữ liệu model. Ví dụ như, view dùng để cập nhật bảng Products sẽ hiển thị các hộp văn bản, drop-down list, và các check box dựa trên trạng thái hiện tại của một đối tượng Product.
- **Controllers:** Controller là các thành phần dùng để quản lý tương tác người dùng, làm việc với model và chọn view để hiển thị giao diện người dùng. Trong một ứng dụng MVC, view chỉ được dùng để hiển thị thông tin, controller chịu trách nhiệm quản lý và đáp trả nội dung người dùng nhập và

tương tác với người dùng. Ví dụ, controller sẽ quản lý các dữ liệu người dùng gửi lên (query-string values) và gửi các giá trị đó đến model, model sẽ lấy dữ liệu từ CSDL nhờ vào các giá trị này.

Ngày 15-08-2012, Microsoft đã cho ra phiên bản ASP.NET MVC 4 với khá nhiều tính năng mới, giao diện cũng được cải thiện khá nhiều so với phiên bản trước đó.

TÀI LIỆU THAM KHẢO

Tiếng Việt:

1. Ebook NoSQL – Nhữ Đình Thuận
2. Nhung CSDL RavenDB vào ứng dụng ASP.NET MVC 3 – Asp.net.vn
3. Một số bài viết như: Một số thông tin về NoSQL và thị trường database, NoSQL, Nhất quán cuối cùng tại SQL Việt Blog (<http://www.sqlviet.com/blog/>)

Tiếng Anh:

1. Ayende Rahien (Oren Eini), RavenDB Documentation - <http://ravendb.net/docs> (trang tham khảo chính)
2. Ayende Rahien (Oren Eini), RavenDB Mythology Documentation Release 1.0, November 29, 2010
3. Eelco Plugge, Peter Membrey and Tim Hawkins, The Definitive Guide to MongoDB The NoSQL Database for Cloud and Desktop Computing.
4. Adam Freeman and Joseph C.Rattz, Jr. Pro LINQ Language Intergrated Query In C# 2010. Apress, 2010.
5. NoSQL resources - <http://nosql-database.org/>
6. NoSQL in the Enterprise - <http://www.infoq.com/articles/nosql-in-the-enterprise>
7. NoSQL wiki - <http://en.wikipedia.org/wiki/NoSQL>
8. Blog hay về RavenDB: Ayende's Blog (<http://ayende.com/blog>), Phillip Haydon's Blog (<http://www.philliphaydon.com/>), Gregor Suttie's Blog (<http://gregorsuttie.com/>)
9. So sánh RavenDB với CouchDB và MongDB (<http://weblogs.asp.net/britchie/archive/2010/08/17/document-databases-compared-mongodb-couchdb-and-ravendb.aspx>)