BÁO CÁO ĐỒ ÁN CUỐI KÌ MÔN CƠ SỞ DỮ LIỆU NÂNG CAO

Giảng Viên Hướng Dẫn: Lương Trần Hy Hiến

NHÓM 420

Ngô Tấn Trung – 43.01.104.191

Quách Đăng Khoa – 43.01.104.078

Trí Tân – 43.01.104.159

Thùy Vy – 43.01.104.210

Võ Kim Hoa – 43.01.104.054

LỜI MỞ ĐẦU

Ngày nay với kỹ 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. 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.

Hệ thống máy chủ cơ sở dữ liệu đòi hỏi phải rất mạnh mẽ nếu không máy chủ sẽ bị quá tải. 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 ì ạch với hệ thống lớn. 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.

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.

MỤC L

YCHƯƠNG 1 - Tổng quan về cơ sở dữ liệu NoSQL.....6

1. NoSQL là gì ?.....6

2. Ưu nhược điểm của cơ sở dữ liệu NoSQL:.....6

a. Ưu điểm:.....6

b. Nhược điểm:.....7

3. So sánh NoSQL với các loại cơ sở dữ liệu khác.....7

4. Phân loại 8

CHƯƠNG 2- TÌM HIỂU VỀ RAVENDB.....9

1. RavenDB là gì?.....9

2. Giới thiệu về RavenDB.....9

3. Cơ bản về RavenDB.....10

a.RavenDB server.....10

b. Documents, Collections và Document xác định duy nhất.....10

c. The Management Studio.....10

d. Tạo khóa cho các Document.....11

e. Thiết kế cấu trúc cho các Document.....11

4. .NET client API.....12

a. .NET client API là gì ?.....12

b. Nguyên tắc thiết kế .NET client API.....12

c. Kết nối với RavenDB data store.....12

d. Những thao tác cơ bản với cơ sở dữ liệu.....13

Đối tượng Session.....14

Insert document.....14

Load và update document.....15

Delete document.....15

Truy vấn cơ bản trong RavenDB.....16

5. HTTP API.....16

6. Mở rộng hệ thống theo chiều ngang(scaling out hay là scaling horizontally).....17

7. So sánh RavenDB với CouchDB và MongDB.....17

CHƯƠNG 3 – Ứng Dụng Quản Lí Thông Tin Sinh Viên Bằng RAVEN DB và ASP.NET MVC.....18

I . RavenDB:.....18
1. Trang Chủ: Để Tải RavenDB.....18
2. Sau Khi Cài Đặt: Chạy lệnh Run with PowerShell.....18
3. Kết nối thành công sẽ tự hiên lên LocalHost của RavenDB.....19
4. Database của RavenDB.....20
5. Dữ liệu trong DataBase tên “ testDB“:.....21
6. Chi tiết từng dữ liệu:.....21

II . ASP.NET x RAVENDB – Cách Tạo Và Truy Vấn Cơ Sở Dữ Liệu: 22
1. Thêm Nuget Package để truy vấn RavenDB 22
2. Hiển thị dữ liệu ASP.NET từ RAVENDB:.....23
3.Tạo dữ liệu:.....24
4.Chỉnh sữa dữ liệu:.....25
5.Chi tiết dữ liệu:.....27
6.Xoá dữ liệu:.....28
7.Tìm kiếm dữ liệu:.....29

III. DEMO:.....30
1.Trang Chủ:.....30
2.Chính Sách:.....31
3.Quản Lí Thông Tin Nhân Viên:.....32
4.Thêm mới nhân viên:.....33
5.Chi Tiết:.....34
6.Tìm Kiếm Thông Tin.....35
7.Chỉnh Sửa:.....35

CHƯƠNG4-TỐNGKẾT.....36

Nội dung báo cáo

Nội dung đề tài được tổ chức thành 4 chương:

Chương 1 – 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 2 – Tìm hiểu về RavenDB: Chương này chúng em sẽ tìm hiểu kĩ về kĩ thuật, cách áp dụng của một cơ sở dữ liệu thuộc loại document database đó là RavenDB.

Chương 3 – Xây dựng ứng dụng sử dụng RavenDB: Quản lý thông tin nhân viên có sử dụng RavenDB làm cơ sở dữ liệu.

Chương 4 – 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 1 - TỔNG QUAN VỀ CƠ SỞ DỮ LIỆU NOSQL

1. 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.

Một số đặc điểm nhận dạng cho thế 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…

2. Ưu nhược điểm của cơ sở dữ liệu NoSQL:

a. Ưu điểm:

Open source: hầu hết các sản phẩm nguồn mở đưa ra cho những người phát triển với nhiều lợi ích to lớn, đặc biết là việc sử dụng miễn phí.

Khả năng mở rộng linh hoạt: do không bị ràng buộc chặt về các mối quan hệ, cấu trúc lưu trữ nên khả năng mở rộng của NoSQL rất linh động.

Các CSDL NoSQL khác nhau cho những dự án khác nhau: mỗi loại CSDL NoSQL cụ thể sẽ là giải pháp phục vụ cho một hoặc một vài vấn đề cụ thể.

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: Những yêu cầu về lưu trữ của công nghệ đám mây với NoSQL là một sự trùng khớp tự nhiên.

b. Nhược điểm:

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 RMBMs 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 không thể mong đợi được cung cấp sự hỗ trợ có thể so sánh được.

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" muồi và các vấn đề về tính không ổn định, trong khi đó tính chín muồi, 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?

Thiếu sự tinh thông: tính mới mẻ của NoSQL có nghĩa là không có nhiều lập trình viên và người quản trị biết công nghệ này. Như vậy sẽ rất khó khăn cho các công ty tìm người có hiểu biết phù hợp.

Những vấn đề về tính tương thích: mỗi CSDL NoSQL có các giao diện lập trình ứng dụng API riêng, các giao diện truy vấn riêng… Sự thiếu hụt các tiêu chuẩn sẽ gây ra rất nhiều khó khăn khi chuyển từ một nhà cung cấp này sang một nhà cung cấp khác nếu có nhu cầu.

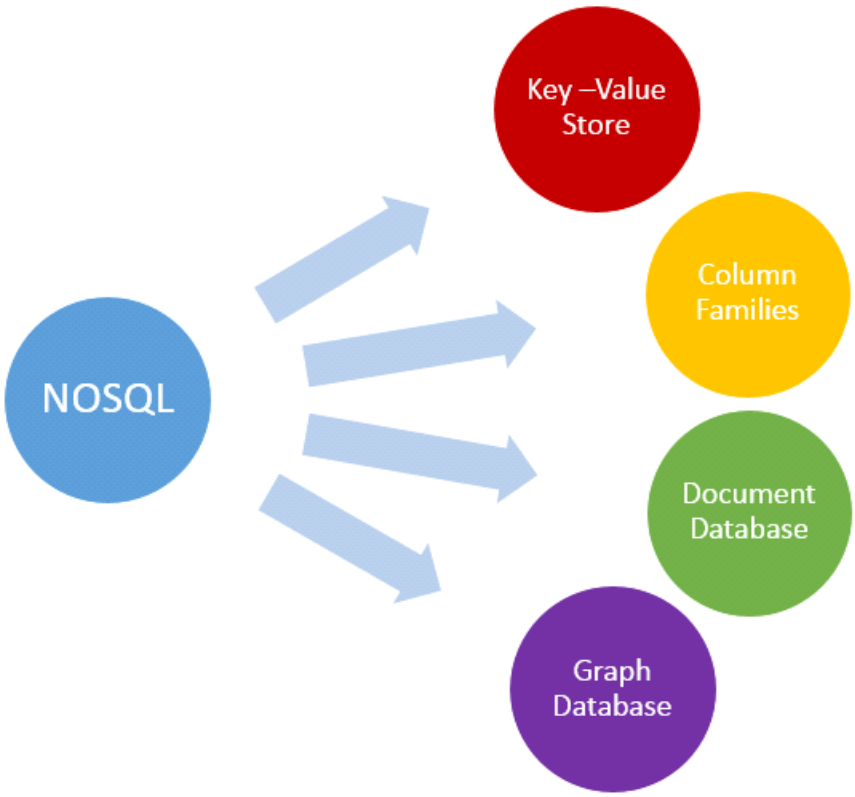
3. So sánh NoSQL với các loại cơ sở dữ liệu khác

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

4. Phân loại

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



Key-Value Database: Dữ liệu được lưu trữ trong database dưới dạng key-value

Document Database: Mỗi object sẽ được lưu trữ trong database dưới dạng một document. Dữ liệu sẽ được lưu trữ dưới dạng BSON/JSON/XML dưới database. Dữ liệu không schema cứng như SQL, do đó ta có thể thêm/sửa field, thay đổi table, … rất nhanh và đơn giản.

Column-Family Database: Dữ liệu được lưu trong database dưới dạng các cột, thay vì các hàng như SQL. Mỗi hàng sẽ có một key/id riêng. Điểm đặt biệt là các hàng trong một bảng sẽ có số lượng cột khác nhau.

Graph Database: Dữ liệu trong graph database được lưu dưới dạng các

node. Mỗi node sẽ có 1 label, 1 số properties như một row trong SQL.

Các node này được kết nối với nhau bằng các relationship.

CHƯƠNG 2- TÌM HIỂU VỀ RAVENDB

1. RavenDB là gì?

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 thế 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 documents, 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.

2. Giới thiệu về 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 có thể được gửi tới máy chủ này từ một hoặc nhiều máy người dùng khác nhau.

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. Cơ bản về RavenDB

a.RavenDB server

Một số cách để chạy RavenDB server:

- Chạy ứng dụng console Raven.Server.exe (tại thư mục /Server/ trong gói sản phẩm)
- 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

b. Documents, Collections và Document xác định duy nhất

- Một thực thể dữ liệu duy nhất trong RavenDB được gọi là một document và tất cả các tài liệu được lưu trữ trong RavenDB như các tài liệu JSON. Các định dạng JSON đã được lựa chọn vì nó có thể lưu trữ phân cấp, con người có thể đọc được. Mọi document đều có siêu dữ liệu gắn liền với nó, theo mặc định nó chỉ chứa dữ liệu được sử dụng trong nội bộ của RavenDB .

- Collections là một tập hợp các tài liệu chia sẻ cùng một loại thực thể RavenDB. Nó không phải là một "bảng cơ sở dữ liệu"(database table), mà là một cách nghĩ của các nhóm tài 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 cố gắng lưu trữ hai thực thể khác nhau theo cùng một id – bản ghi thứ hai sẽ ghi đè lên bản ghi đầu tiên mà không có cảnh báo nào. Quy ước trong RavenDB: documentID được kết hợp từ tên bộ sưu tập(collection name) và id duy nhất của tài liệu trong bộ sưu tập. Tuy nhiên, đó chỉ là một quy ước. Document ID thì không phụ thuộc vào

loại thực thể, do đó không bắt buộc phải chứa tên của bộ sưu tập chứa nó.

c. The Management Studio

Tất cả các thực thể máy chủ có thể quản lý thông qua một ứng dụng Silverlight truy cập từ xa - Management Studio. Nó có thể được truy cập bằng cách trỏ trình duyệt của bạn đến địa chỉ và cổng máy chủ lắng nghe (mặc định là http://localhost:8080).

d. Tạo khóa cho các Document

RavenDB tự động tạo khóa: Khi chúng ta không chỉ định khóa cho các document, RavenDB sẽ tự động tạo mới khóa cho các document. Raven sử dụng các GUID liên tiếp để tạo các khóa. Các GUID liên tiếp này là duy nhất và có lợi trong việc sắp xếp các indexing. Cách này thường được dùng nếu chúng ta không quan tâm tới việc tạo khóa cho các document như là lưu lại các log hay là khi mà người dùng không bao giờ hiển thị dữ liệu các khóa này.

Tự tạo khóa cho các document: Chúng ta có thể gán khóa cho các document trước khi lưu các document này xuống cơ sở dữ liệu. Thường sử dụng trong các trường hợp như chúng ta muốn tạo khóa cho tập hợp người dùng trong hệ thống, ví dụ như: “user/nguyenvana”

Khóa xác định: Raven xác định REST như là khóa, ví dụ “posts/1234”. Nếu bạn lưu document với khóa kết thúc bằng “/”, Raven sẽ tự động theo dấu các số xác minh cho tiền tố nếu nó không tồn tại và sẽ nối thêm các số xác minh vào khóa. Cách này được dùng hầu hết cho các trường hợp vì nó tạo ra khóa mà con người có thể đọc được.

e. Thiết kế cấu trúc cho các Document

RavenDB lưu trữ dữ liệu không theo một lược đồ cố định, nó có lược đồ tùy ý tùy biến. Tuy nhiên, chúng ta vẫn nên dành nhiều thời gian xem xét làm thế nào để thiết kế các document nhằm đảm bảo rằng chúng ta có thể truy cập tất cả dữ liệu chúng ta cần phục vụ 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. Bởi vì RavenDB lưu trữ dữ liệu phi quan hệ nên thiết kế cấu trúc document theo cách riêng của document database sẽ đem lại lợi ích to lớn. Nhờ đó mà chúng ta sẽ tận dụng những điểm mạnh của cơ sở dữ liệu hướng document như là RavenDB.

4. .NET client API

a. .NET client API là gì ?

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)...

b. 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.

c. Kết nối với 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. Khi ứng dụng kết thúc, document store nên được giải phóng và xóa sạch một cách hợp lý.

- Chạy ở server mode

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

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

- Chạy ở embedded mode

```
var documentStore = new EmbeddableDocumentStore { DataDirectory = "path/to/database/directory" };
documentStore.Initialize();
```

- Hỗ trợ Silverlight

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

- Sử dụng chuỗi kết nối (connection string)

```
new DocumentStore { ConnectionStringName = "MyRavenConStr"}
```

Định nghĩa trong file app.config

```
<connectionStrings>
    <add name="Local" connectionString="DataDir = ~\Data"/>
        <add name="Server" connectionString="Url = http://localhost:8080"/>
    <add name="Secure" connectionString="Url = http://localhost:8080;user=beam;password=up;ResourceManagerId=d5723e19-92ad-4531-adad-8611e6e05c8a"/>
</connectionStrings>
```

d. Những thao tác cơ bản với cơ sở dữ liệu

Chúng ta có một số lớp cơ bản dưới đây để thực hiện thao tác với cơ sở dữ liệu:

```
public class BlogPost
```

```
{
        public string Id { get; set; }
        public string Title { get; set; }
        public string Category { get; set; }
        public string Content { get; set; }
        public DateTime PublishedAt { get; set; }
        public string[] Tags { get; set; }
        public BlogComment[] Comments { get; set; }
}
public class BlogComment
{
        public string Title { get; set; }
        public string Content { get; set; }
}
```

- 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())
{
        // Sử dụng session để thao tác với cơ sở dữ liệu
        var entity = new BlogComment{ Title = "Title", Content
        = “Content” };
        session.Store(entity);
        session.SaveChanges();
}
```

Trong ngữ cảnh này, chúng ta có thể nghĩ rằng session quản lý tất cả thay đổi nội tại và SaveChanges sẽ gửi tất cả thay đổi đó tới

RavenDB server. Tất cả các thao tác dữ liệu trong lời gọi SaveChanges sẽ được thực hiện (hoặc là tất cả cùng thành công, hoặc là cùng thất bại).

- Insert document

Để lưu một bài viết xuống cơ sở dữ liệu, ta sẽ tạo một mới một thực thể bài viết:

```
// tạo thực thể mới của lớp BlogPost
BlogPost post = new BlogPost()
{
        Title = "Hello RavenDB",
        Category = "RavenDB",
        Content = "This is a blog about RavenDB",
        Comments = new BlogComment[]
        {
          new BlogComment() {Title = "Unrealistic", Content = "This example is unrealistic"},
          new BlogComment() {Title = "Nice", Content = "This example is nice"}
        }
};
```

Lưu trữ bài viết vừa tạo bằng cách gọi hàm Store() và SaveChanges()

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

- Load và update document

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

```
// BlogPosts/1 là một thực thể của collection BlogPost với Id là 1
BlogPost existingBlogPost = session.Load<BlogPost>("BlogPosts/1");
```

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

```
existingBlogPost.Title = "Some new title";
```

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.
```

- Delete document

Xóa bằng cách tham chiếu đến đối tượng:

- Khi ta lấy được document thông qua hàm load() thì chúng ta có thể xóa được document thông qua hàm delete():

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

Xóa dựa vào khóa:

- Dùng lệnh Defer của tính năng Advanced session

```
session.Advanced.Defer(new DeleteCommandData {Key = "posts/1234"});
```

- Dùng DatabaseCommands:

```
session.Advanced.DatabaseCommands.Delete("posts/1234"
, null);
```

- Truy vấn cơ bản trong RavenDB

Chúng ta sử dụng Linq để truy vấn dữ liệu. Ví dụ như chúng ta cần truy vấn tất cả các bài viết blog theo danh mục xác định

```
var results = from blog in session.Query<BlogPost>()
                where blog.Category == "RavenDB"
```

```
                select blog;
```

Hoặc là với cú pháp khác, ta có thể lấy những bài viết có ít nhất 10 comments:

```
var results = session.Query<BlogPost>()
                .Where(x => x.Comments.Length >= 10)
                .ToList();
```

5. 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 (tạm dịch là 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.

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 trình bày dễ dàng để sử dụng những tính năng mạnh mẽ như batching và multi-document transactions.

6. Mở rộng hệ thống theo chiều ngang(scaling out hay là scaling horizontally)

RavenDB hỗ trợ sẵn 2 gói mở rộng là nhân bản(Replication) và phân tán dữ liệu trên nhiều máy chủ (sharding). Cả hai tính năng này trực giao, có nghĩa là chúng có thể được sử dụng kết hợp với nhau. Sharding là một tính năng phía máy khách, có nghĩa là toàn bộ quyết định được thực hiện phía máy khách. Nhân bản được thực hiện giữa 2 điểm đầu, các máy chủ tự thực hiện việc nhân bản và cũng cần nhận

biết các trường hợp khi một trong các nút (node) bị hư hỏng và khi một nút báo cáo lỗi xung đột khi nhân bản.

7. So sánh RavenDB với CouchDB và MongDB

- 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.

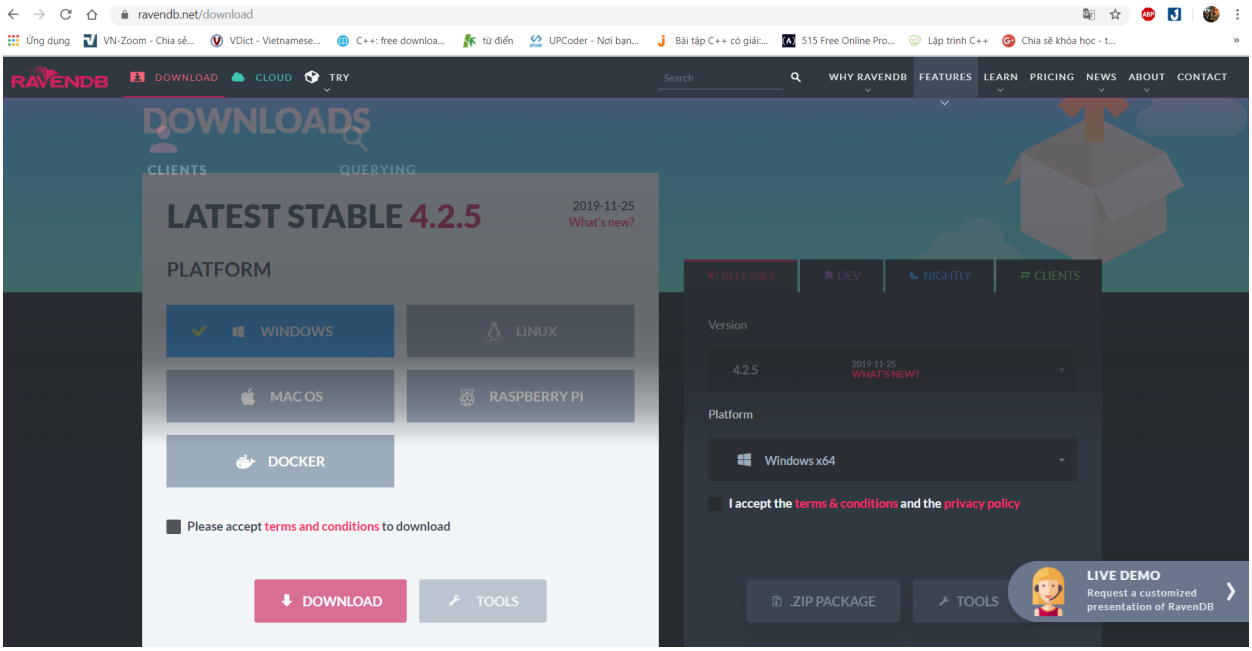
Ngoài những tính năng trên thì RavenDB còn có một số đặc điểm mà khi sử dụng tôi thấy thích thú, đó là:

- 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, …

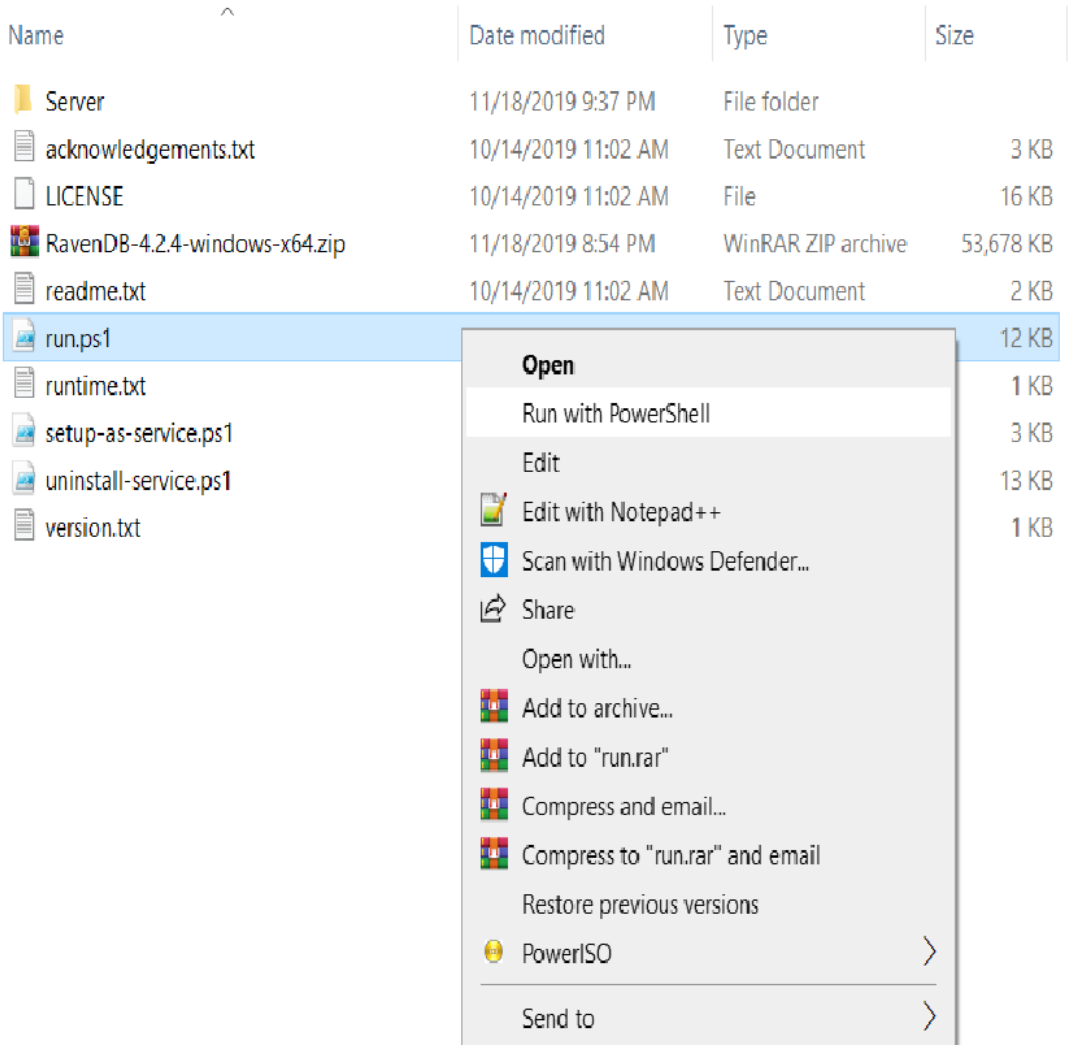
CHƯƠNG 3 – ỨNG DỤNG QUẢN LÍ THÔNG TIN NHÂN VIÊN BẰNG RAVEN DB VÀ ASP.NET MVC

I. RavenDB:

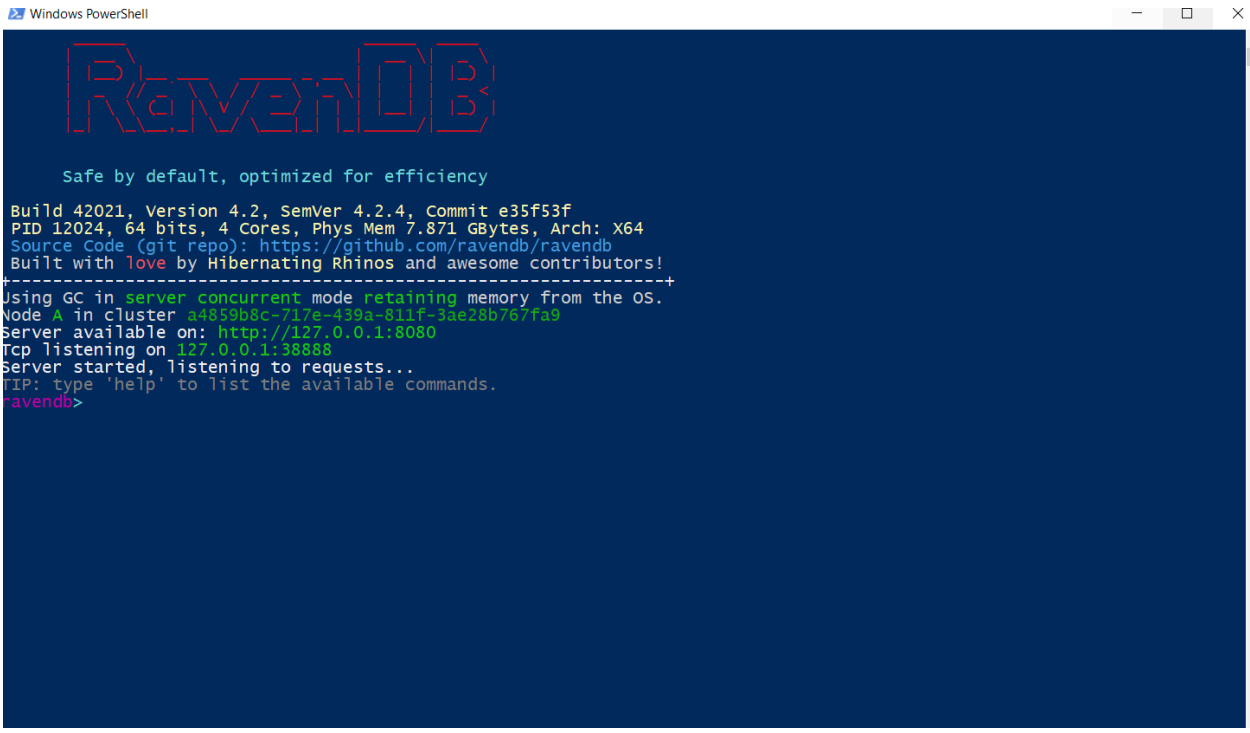
1. Trang Chủ: Để Tải RavenDB

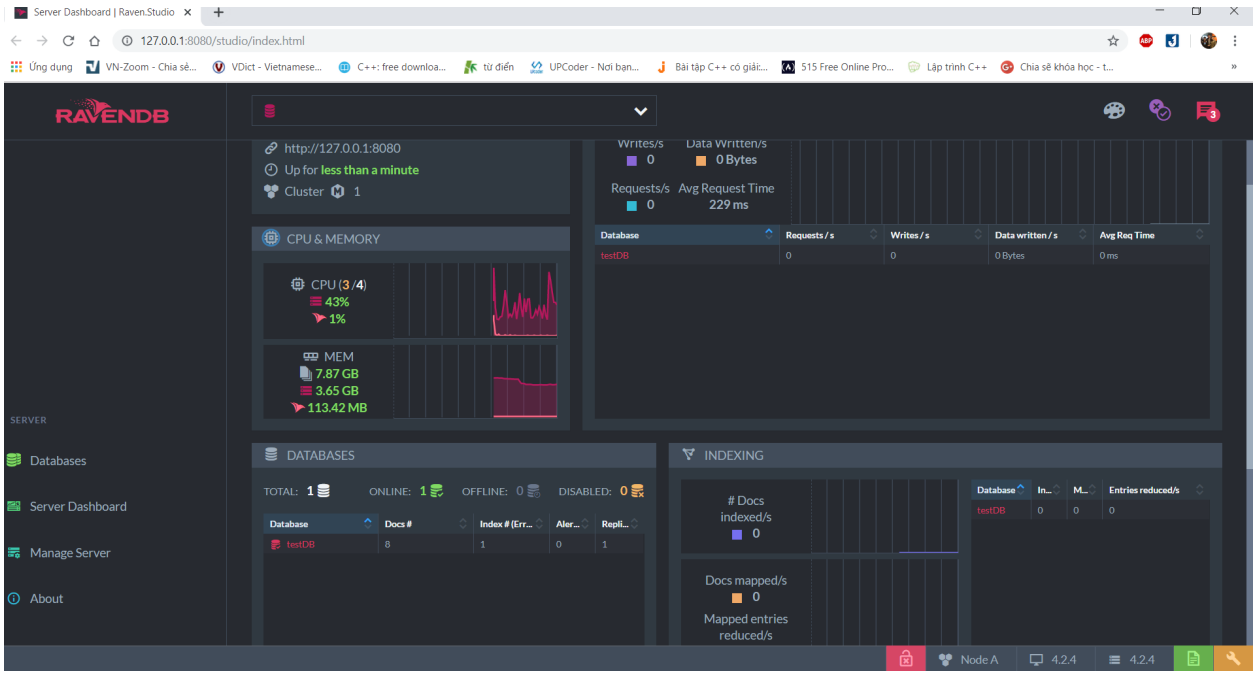


2. Sau Khi Cài Đặt: Chạy lệnh Run with PowerShell

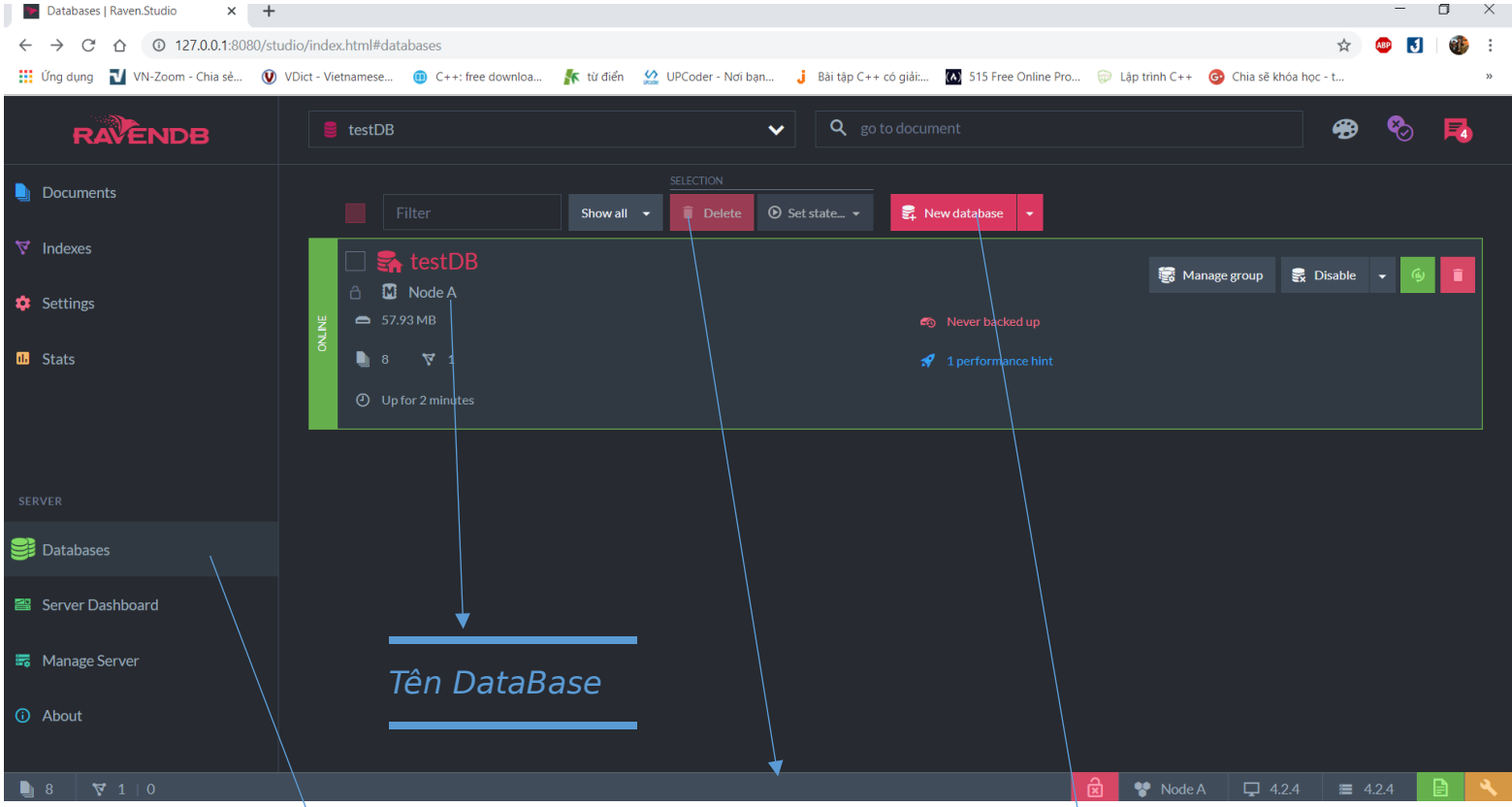


3. Kết nối thành công sẽ tự hiên lên LocalHost của RavenDB

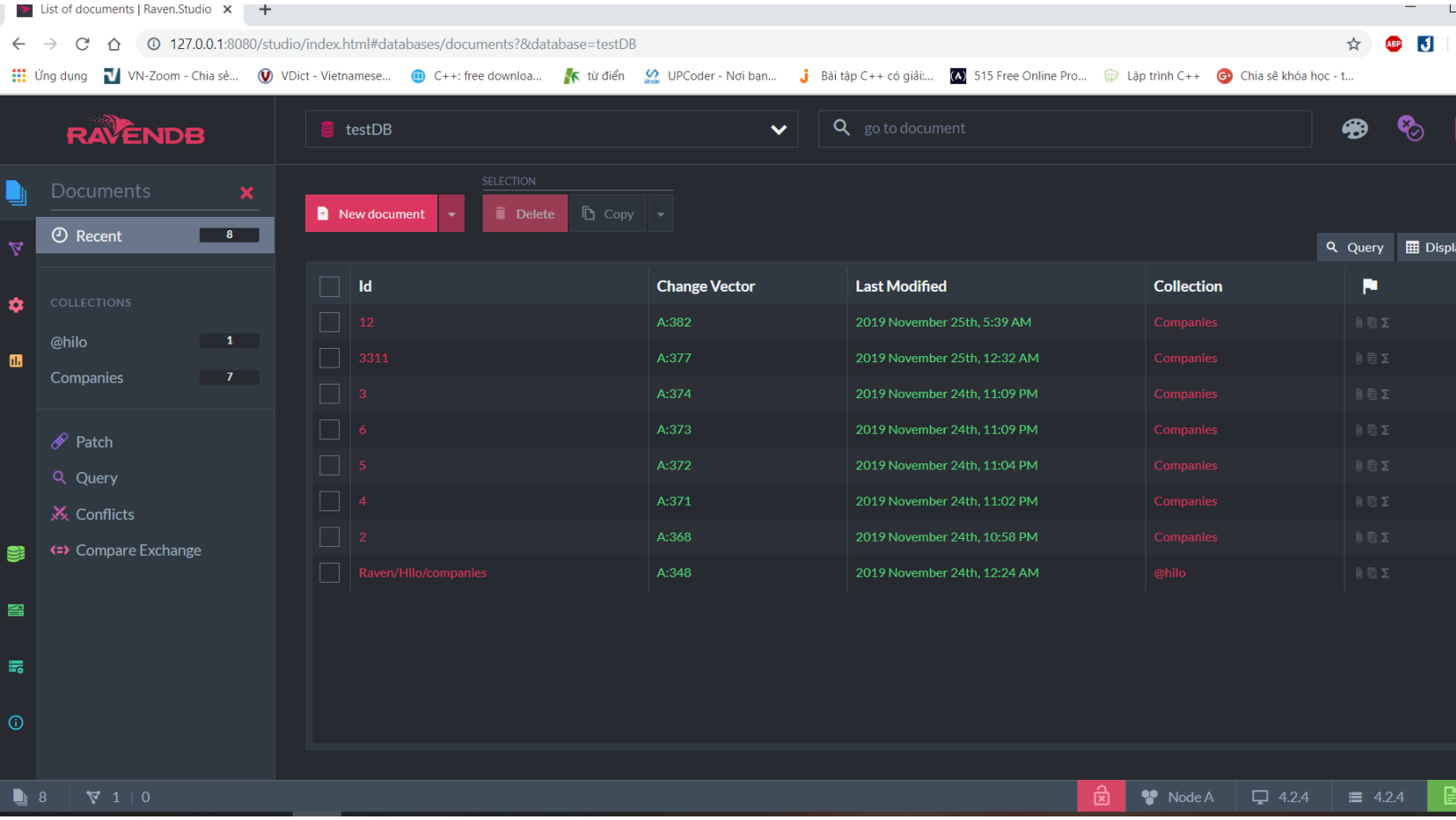




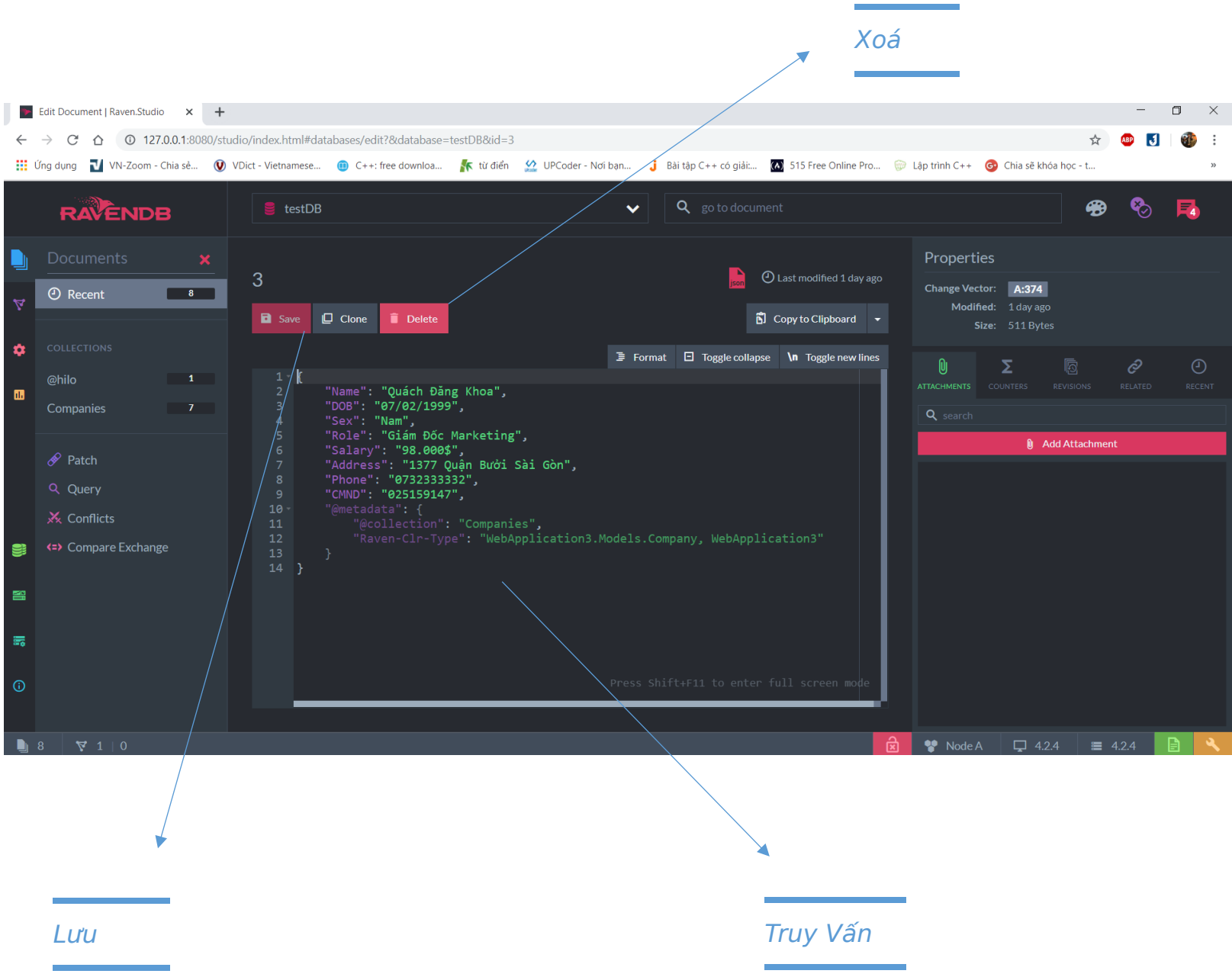
4. Database của RavenDB



5. Dữ liệu trong DataBase tên " testDB"

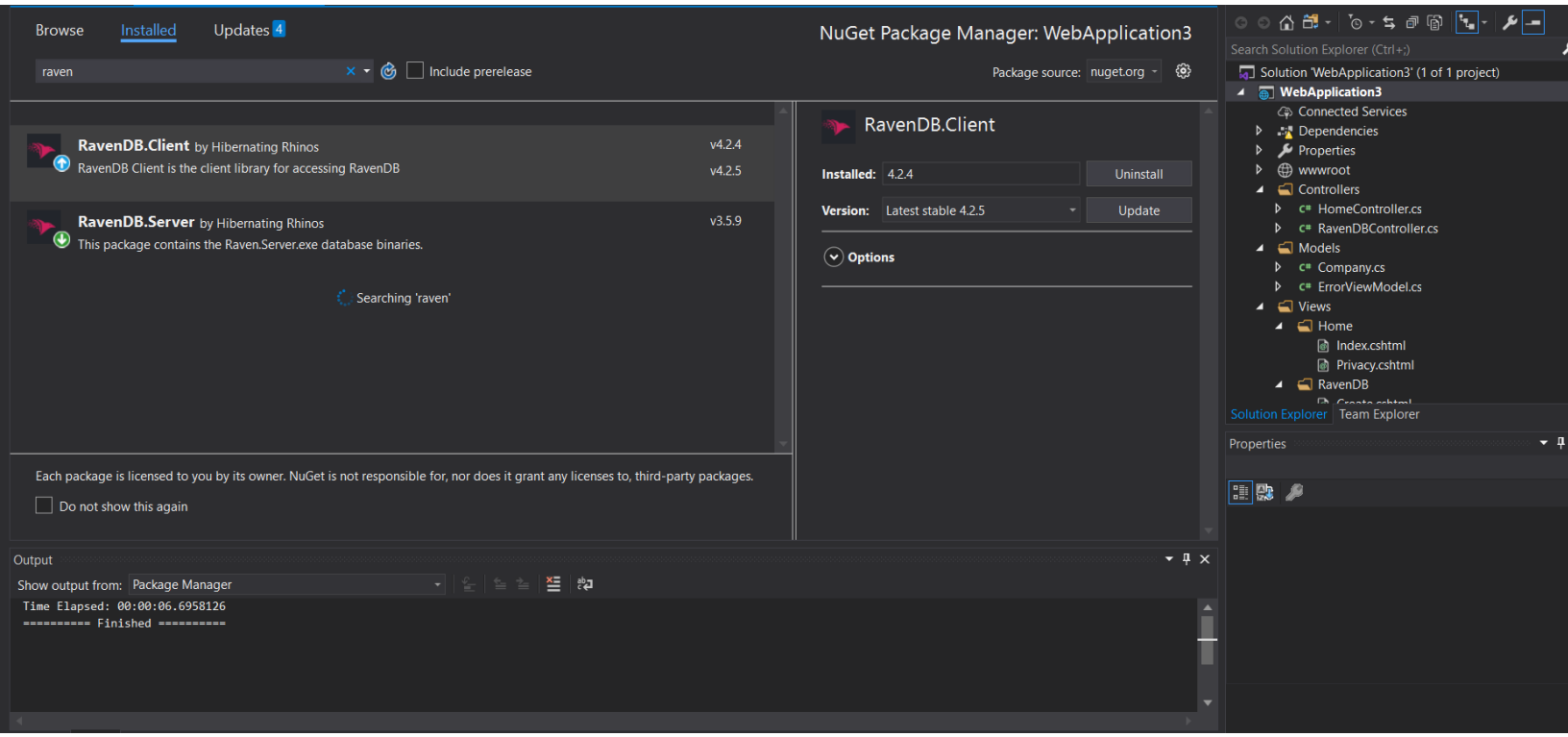


6. Chi tiết từng dữ liệu



I. ASP.NET x RAVENDB – Cách Tạo Và Truy Vấn Cơ Sở Dữ Liệu:

1. Thêm Nuget Package để truy vấn RavenDB



2. Hiển thị dữ liệu ASP.NET từ RAVENDB

```
//Lấy Dữ Liệu Từ DataBase hiển thị ra màn hình
0 references
public ActionResult Index()
{
    //kết nối DB
    List<Company> model = new List<Company>();
    using (var documentstore = new DocumentStore
    {
        Urls = new[] { "http://localhost:8080" },
        Database = "testDB"
    })
    {
        documentstore.Initialize();
        using (var session = documentstore.OpenSession())
        {
            model = session.Query<Company>().ToList();
        }

    }
        return View(model);
}
```

3. Tạo dữ liệu

```
0 references
public ActionResult Create()
{
    return View();
}
[HttpPost]
// Gửi Dữ Liệu Tạo Thành Viên Mới
0 references
public ActionResult Create(Company model)
{
    using (var documentstore = new DocumentStore
    {
        Urls = new[] { "http://localhost:8080" },
        Database = "testDB"
    })
    {
        documentstore.Initialize();
        using (var session = documentstore.OpenSession())
        {
            session.Store(model);
            session.SaveChanges();

        }

    }
    return View(model);
}
```

4. **Chỉnh sữa dữ liệu**

```
//Lấy Dữ Liệu Để Chỉnh Sữa
[HttpGet]
0 references
public ActionResult Edit(int Id)
{
    Company company = new Company();
    using (var documentstore = new DocumentStore
    {
        Urls = new[] { "http://localhost:8080" },
        Database = "testDB"
    })
    {
        documentstore.Initialize();
        using (var session = documentstore.OpenSession())
        {
            company = session.Load<Company>(Id.ToString());
        }
    }
    return View(company);
}
```

```
//Chỉnh Sữa Dữ Liệu
[HttpPost]
0 references
public ActionResult Edit(Company model)
{
    Company result = new Company();
    using (var documentstore = new DocumentStore
    {
        Urls = new[] { "http://localhost:8080" },
        Database = "testDB"
    })
    {
        documentstore.Initialize();
        using (var session = documentstore.OpenSession())
        {
            result = session.Load<Company>(model.Id);
            if (result != null)
            {
                result.Id = model.Id;
                result.Name = model.Name;
                result.DOB = model.DOB;
                result.Sex = model.Sex;
```

```
                result.Sex = model.Sex;
                result.Role = model.Role;
                result.Salary = model.Salary;
                result.Address = model.Address;
                result.Phone = model.Phone;
                result.CMND = model.CMND;
                session.SaveChanges();
            }
        }
    }
    return Redirect("/RavenDB/index");
}
```

5. Chi tiết dữ liệu:

```
//Chi Tiết Thông Tin Nhân Viên
[HttpGet]
0 references
public ActionResult Details(int Id)
{
    Company company = new Company();
    using (var documentstore = new DocumentStore
    {
        Urls = new[] { "http://localhost:8080" },
        Database = "testDB"
    })
    {
        documentstore.Initialize();
        using (var session = documentstore.OpenSession())
        {
            company = session.Load<Company>(Id.ToString());
        }
    }
        return View(company);

}
```

6. Xoá dữ liệu:

```
//Xoá Dữ Liệu
0 references
public ActionResult Delete(int Id)
{
    Company company = new Company();
    using (var documentstore = new DocumentStore
    {
        Urls = new[] { "http://localhost:8080" },
        Database = "testDB"
    })
    {
        documentstore.Initialize();
        using (var session = documentstore.OpenSession())
        {
            company = session.Load<Company>(Id.ToString());
            session.Delete<Company>(company);
            session.SaveChanges();
        }
    }
    return Redirect("/RavenDB/index");
}
```

7. Tìm kiếm dữ liệu

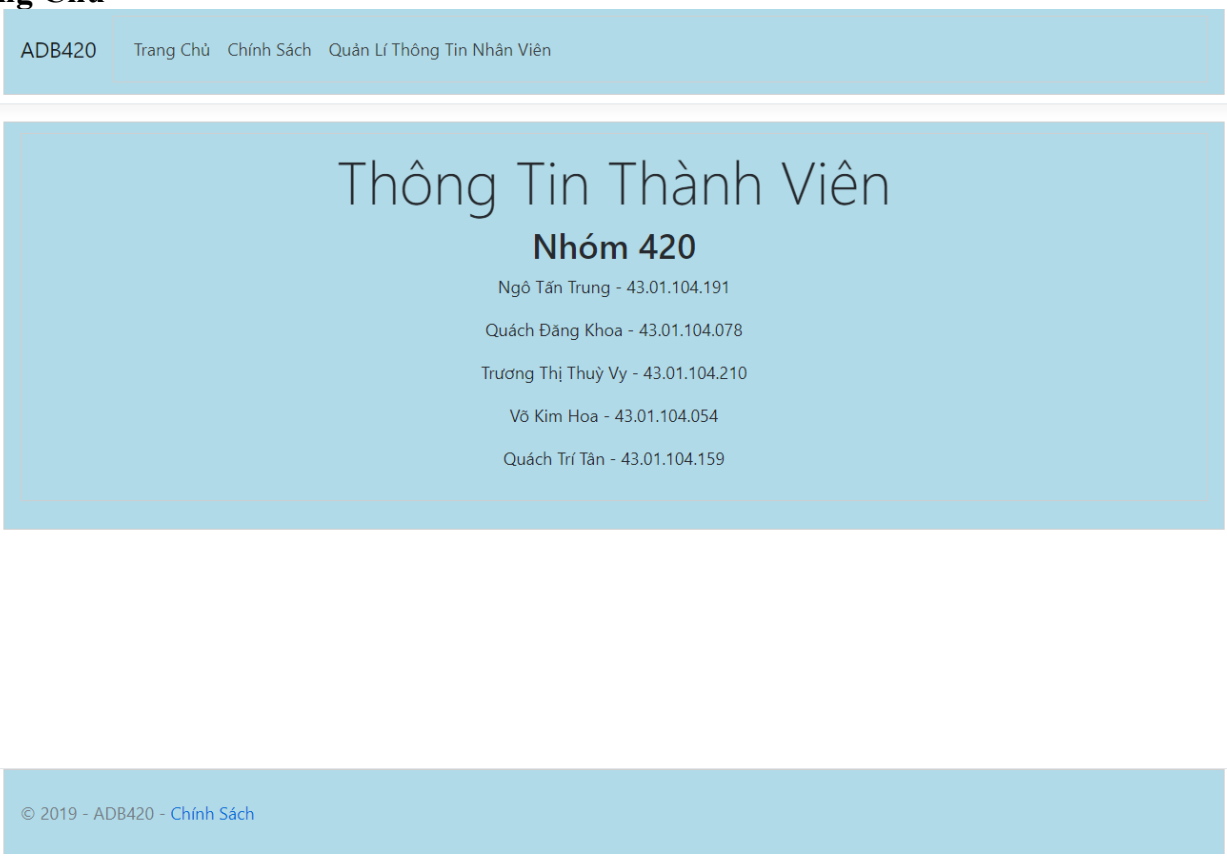
```
//Tìm Kiếm Dữ Liệu từ DataBase
0 references
public ActionResult List(string searchBy,string search)
{

    using (var documentstore = new DocumentStore
    {
        Urls = new[] { "http://localhost:8080" },
        Database = "testDB"
    })
    //Lưu trữ Document/Data
    {
        documentstore.Initialize();

        using (var session = documentstore.OpenSession())
        {
            if (searchBy == "Name")
            {
                return View(session.Query<Company>().Where(x => x.Name.StartsWith(search) || search == null).ToList());
            }
            else
            {
                return View(session.Query<Company>().Where(x => x.Id.StartsWith(search)|| search == null).ToList());
            }
        }
    }
```

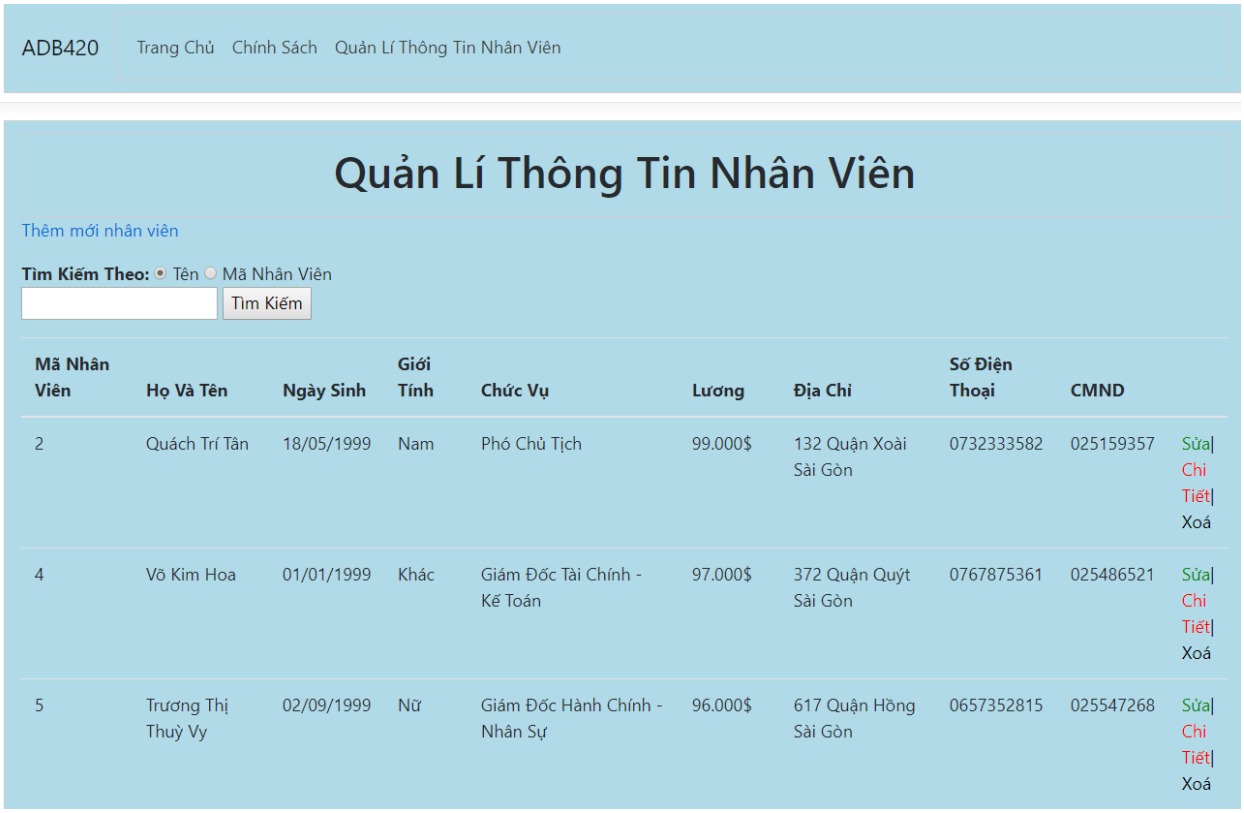
II. DEMO:

1. Trang Chủ

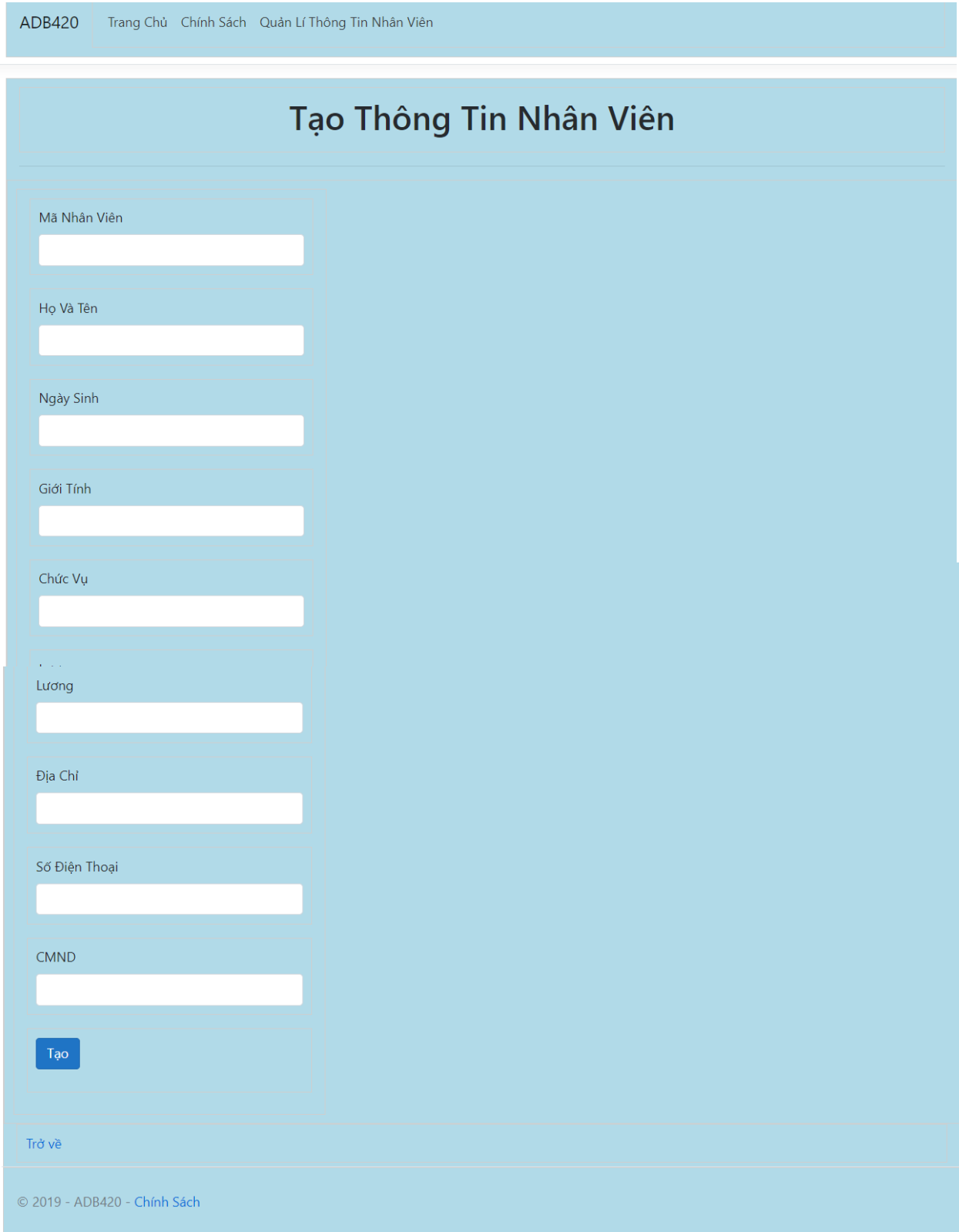


2. Chính Sách

3. Quản Lí Thông Tin Nhân Viên


ADB420
Trang Chủ
Chính Sách
Quản Lí Thông Tin Nhân Viên
Quản Lí Thông Tin Nhân Viên
Thêm mới nhân viên
Tìm Kiếm Theo: Tên Mã Nhân Viên
Tìm Kiếm
Mã Nhân Viên
Họ Và Tên
Ngày Sinh
Giới Tính
Chức Vụ
Lương
Địa Chỉ
Số Điện Thoại
CMND
2 Quách Trí Tân 18/05/1999 Nam Phó Chủ Tịch 99.000$ 132 Quận Xoài Sài Gòn 0732333582 025159357 Sửa| Chi Tiết| Xoá
4 Võ Kim Hoa 01/01/1999 Khác Giám Đốc Tài Chính - Kế Toán 97.000$ 372 Quận Quýt Sài Gòn 0767875361 025486521 Sửa| Chi Tiết| Xoá
5 Trương Thị Thuỳ Vy 02/09/1999 Nữ Giám Đốc Hành Chính - Nhân Sự 96.000$ 617 Quận Hồng Sài Gòn 0657352815 025547268 Sửa| Chi Tiết| Xoá

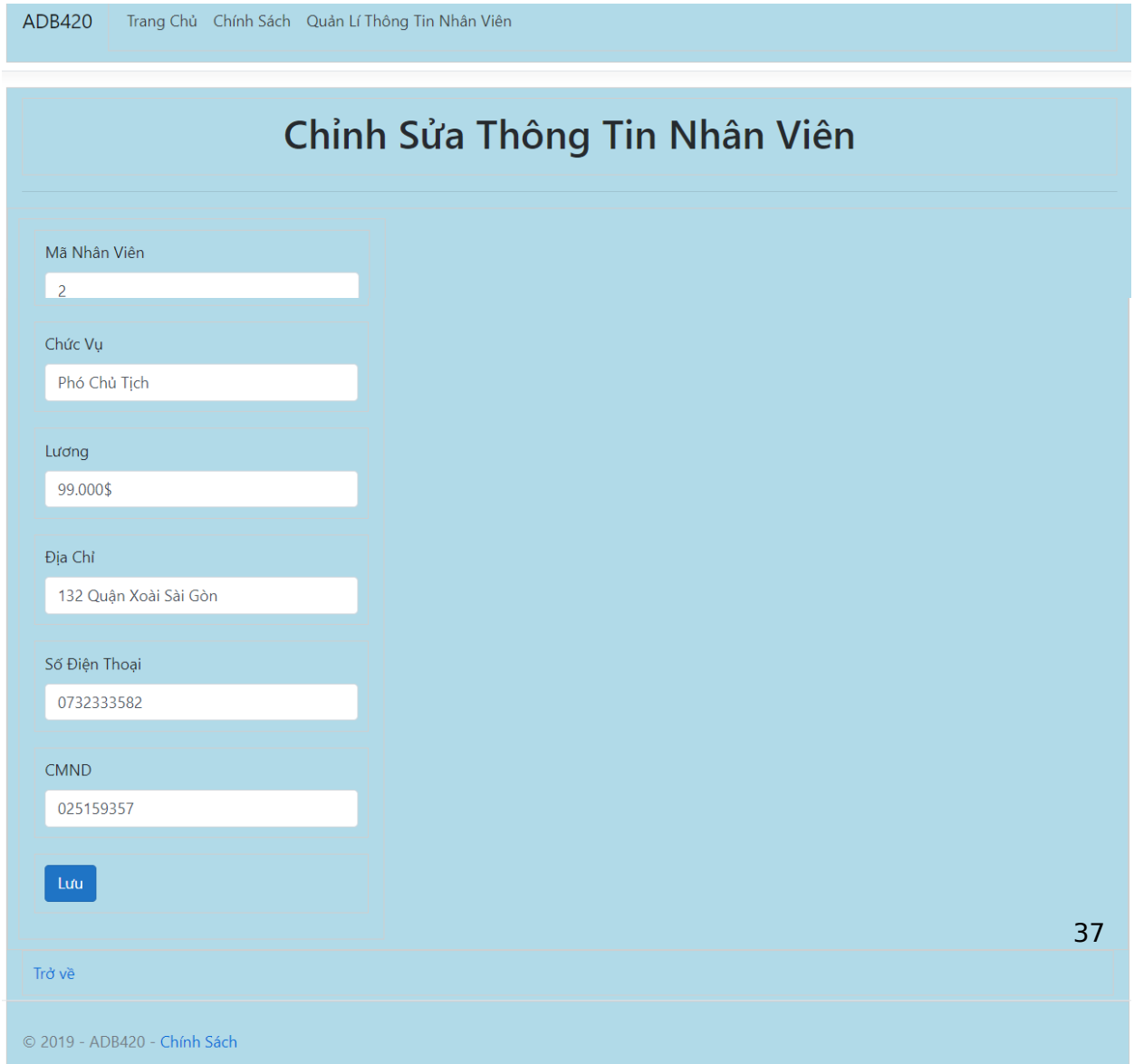
4. Thêm mới nhân viên



5. Chi Tiết

6. Tìm Kiếm Thông Tin

7. Chỉnh Sửa


ADB420
Trang Chủ
Chính Sách
Quản Lí Thông Tin Nhân Viên
Chỉnh Sửa Thông Tin Nhân Viên
Mã Nhân Viên
2
Chức Vụ
Phó Chủ Tịch
Lương
99.000$
Địa Chỉ
132 Quận Xoài Sài Gòn
Số Điện Thoại
0732333582
CMND
025159357
Lưu
Trở về
© 2019 - ADB420 - Chính Sách

CHƯƠNG 4 – TỔNG KẾT

1. Kết quả đạt được:

● Về mặt lý thuyết:

● Tổng hợp và phân tích về cơ sở dữ liệu NoSQL . Qua tài liệu này, người đọc có được cái nhìn khái quát về NoSQL và có thể ứng dụng nó vào các hệ thống cần lưu trữ rất nhiều dữ liệu

. ● Tìm hiểu những kiến thức về RavenDB và các triển khai một ứng dụng sử dụng RavenDB . ● Về mặt thực nghiệm: Xây dựng được một ứng dụng quản lý thông tin nhân viên sử dụng cơ sở dữ liệu RavenDB trên nền Web. Ứng dụng tuy không quá quy mô nhưng cũng đã áp dụng được những kĩ thuật cơ bản và nâng cao của RavenDB và ứng dụng đã thể hiện tốc độ vượt trội khi hoạt động với một lượng lớn dữ liệu, đáp ứng được yêu cầu đề ra.

2. Hướng phát triển:

● 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.