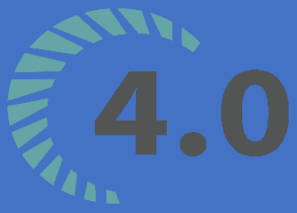
BỘ MÔN HỆ THỐNG THÔNG TIN – KHOA CÔNG NGHỆ THÔNG TIN

ĐẠI HỌC KHOA HỌC TỰ NHIÊN THÀNH PHỐ HỒ CHÍ MINH, ĐẠI HỌC QUỐC GIA TP HCM

QUẢN TRỊ CSDL HIỆN ĐẠI



Sinh viên thực hiện:

19120464	Phạm Ngọc Cường
19120583	Lê Thái Bình Minh
19120590	Huỳnh Thanh Mỹ
19120529	Nguyễn Phước Huy

GV phụ trách: cô Nguyễn Trần Minh Thư

QUẢN TRỊ CƠ SỞ DỮ LIỆU HIỆN ĐẠI

HỌC KỲ II – NĂM HỌC 2022 - 2023

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
227 Nguyễn Văn Cừ, Phường 4, Quận 5, TP.HCM
Điện Thoại: (08) 38.354.266 - Fax:(08) 38.350.096

BẢNG THÔNG TIN CHI TIẾT NHÓM

Mã nhóm:	05	
Số lượng:	**4**	
MSSV	**Họ tên**	**Email**
19120464	Phạm Ngọc Cường	19120464@student.hcmus.edu.vn
19120583	Lê Thái Bình Minh	19120583@student.hcmus.edu.vn
19120590	Huỳnh Thanh Mỹ	19120590@student.hcmus.edu.vn
19120529	Nguyễn Phước Huy	19120529@student.hcmus.edu.vn

Bảng phân công & đánh giá hoàn thành công việc

Công việc thực hiện	Người thực hiện	Mức độ hoàn thành	Đánh giá của nhóm
• Xây dựng demo Tìm & chọn phòng. • Xây dựng template code.	19120464 - Phạm Ngọc Cường	100%	10/10
• Xây dựng demo Chọn phòng.	19120590 – Huỳnh Thanh Mỹ	100%	10/10
• Xây dựng demo Sign in/Sign up + Query. • Thử nghiệm truy vấn với tần suất lớn, tối ưu truy vấn.	19120529 - Nguyễn Phước Huy	100%	10/10
• Mô tả quy trình nghiệp vụ, vẽ Usecase. • Thiết kế dữ liệu, tạo CSDL mẫu. • Làm báo cáo • Phân công công việc	19120583- Lê Thái Bình Minh	100%	10/10

Mục Lục

Kết quả thực hiện

I. So sánh Redis, MongoDB, Cassandra và Neo4J

1. Tổng quan

Redis, Mongo, Cassandra và Neo4j đều là 4 HQT CSDL nổi tiếng hiện nay, sau đây là những ưu điểm và khuyết điểm nổi bật của chúng:

	MongoDB	**Redis**	**Cassandra**	**Neo4J**
Ưu điểm	- Tích hợp tốt với các ứng dụng web và mobile.	- Hiệu suất cao trong việc lưu trữ và truy xuất dữ liệu.	- Khả năng mở rộng tuyến tính với số lượng node.	- Cấu trúc dữ liệu đồ thị mạnh mẽ cho việc tìm kiếm và phân tích dữ liệu liên quan đến quan hệ giữa các đối tượng.
	- Hỗ trợ tính năng sharding cho khả năng mở rộng dữ liệu.	- Hỗ trợ nhiều kiểu dữ liệu và tính năng caching.	- Tính khả dụng cao, có khả năng xử lý các vấn đề về đồng bộ hóa dữ liệu.	- Có tính năng tìm kiếm toàn văn và định tuyến động.
	- Hỗ trợ các tính năng phức tạp như MapReduce và aggregation.	- Dễ dàng triển khai và sử dụng.	- Có khả năng xử lý các bản ghi hàng tỷ.	- Có khả năng tìm kiếm và truy vấn dữ liệu đồ thị rất nhanh.
	- Hỗ trợ các tính năng ACID cho các giao dịch trong cùng một document.	- Hỗ trợ tính năng publish/subscribe cho việc xử lý các	- Hỗ trợ các tính năng ACID và khả năng xử lý các nút bị lỗi.	- Có khả năng xử lý các truy vấn phức tạp với dữ liệu đồ thị.

	MongoDB	**Redis**	**Cassandra**	**Neo4J**
		thông báo realtime.		
	- Cung cấp tính năng tìm kiếm toàn văn và tìm kiếm theo truy vấn.	- Có khả năng xử lý nhiều truy vấn cùng lúc.	- Cung cấp tính năng tìm kiếm toàn văn và tìm kiếm theo truy vấn.	- Có khả năng tối ưu hóa truy vấn bằng cách sử dụng các chỉ mục đồ thị.
Nhược điểm	- Không hỗ trợ transaction qua các document khác nhau.	- Không hỗ trợ tính năng query phức tạp.	- Không hỗ trợ các tính năng phức tạp như MapReduce và aggregation.	- Không phù hợp cho các ứng dụng có dữ liệu lớn và cần khả năng mở rộng tuyến tính.
	- Có thể dễ bị lỗi đồng bộ hóa khi có nhiều phiên bản của cùng một document.	- Có thể dễ bị mất dữ liệu trong trường hợp xảy ra sự cố.	- Khó khăn trong việc triển khai các hệ thống phân tán.	- Không hỗ trợ tính năng ACID.
	- Có thể dễ bị tấn công và xâm nhập dữ liệu khi không được cấu hình đúng cách.	- Không phù hợp cho việc lưu trữ các dữ liệu quan trọng.	- Khôngcó tính năng tìm kiếm toàn văn mạnh mẽ và định tuyến động.	- Cần phải có kiến thức về lập trình đồ thị để có thể sử dụng hiệu quả.

2. Chi tiết

Sau khi tìm hiểu, nhóm đã lập bảng chi tiết để so sánh 4 CSDL như sau:

Tiêu chí	Redis	MongoDB	Cassandra	Neo4j
Kiểu dữ liệu	Nhiều kiểu dữ liệu có kích thước lên đến 512MB	Document JSON	Hỗ trợ nhiều kiểu dữ liệu khác nhau	Hỗ trợ nhiều kiểu dữ liệu
Khả năng tương thích	Cao	Cao	Cao	Cao(ngôn ngữ truy vấn Cypher)
Hiệu suất	Cao	Cao	Cao	Cao
Cộng đồng	Lớn	Lớn	Lớn	Đông đảo
Tính nhất quán	Cao	Cao	Cao	Cao
Tính bảo mật	Tính bảo mật không cao	Cao	Cao	Cao
Tính sẵn sàng	Cao	Cao	Cao	Cao
Tính mở rộng	Chiều ngang và dọc	Theo chiều ngang và dọc	cao	Cao
Chi phí	Miễn phí	Miễn phí	Miễn phí	Có 2 phiên bản miễn phí và trả phí

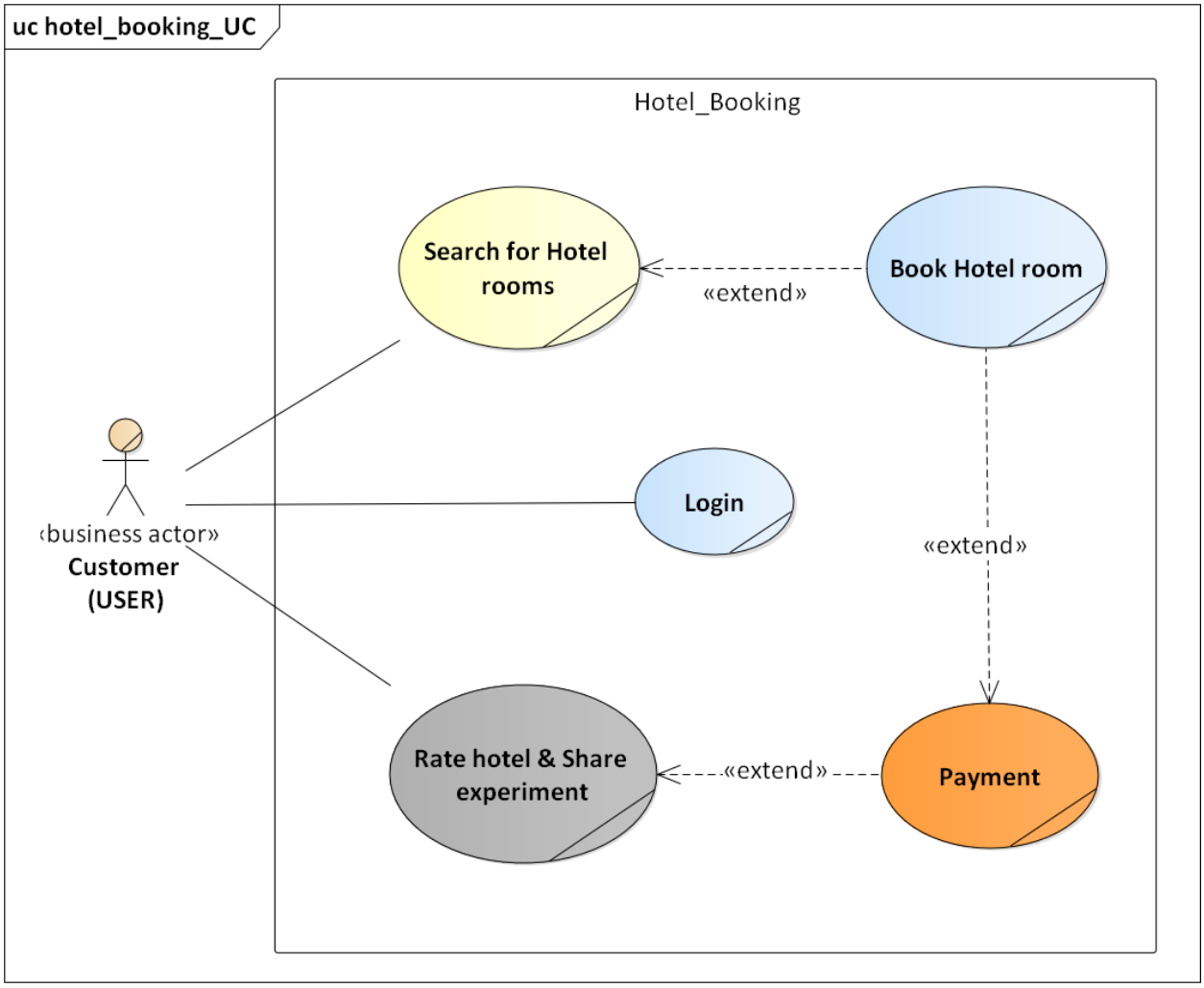
II. Mô tả lại các quy trình nghiệp vụ đặt khách sạn của hệ thống Agoda

Hệ thống Agoda là một trang web đặt phòng khách sạn trực tuyến, cho phép người dùng đặt phòng và thanh toán trực tuyến. Các quy trình nghiệp vụ đặt khách sạn của hệ thống Agoda có thể được mô tả như sau:

- Tìm kiếm khách sạn: Người dùng truy cập vào trang web Agoda và tìm kiếm khách sạn bằng cách nhập địa điểm, ngày nhận phòng và ngày trả phòng.
- Lọc kết quả: Hệ thống Agoda sẽ hiển thị kết quả tìm kiếm và cho phép người dùng lọc kết quả bằng nhiều tiêu chí khác nhau như giá, đánh giá của khách hàng, tiện nghi và vị trí.
- Chọn phòng: Người dùng chọn phòng thích hợp và xác nhận thông tin đặt phòng như số lượng phòng, số lượng khách, thông tin liên lạc và thời gian nhận phòng và trả phòng.
- Thanh toán: Sau khi xác nhận thông tin đặt phòng, người dùng thanh toán bằng các phương thức thanh toán trực tuyến như ví điện tử, thẻ tín dụng hoặc chuyển khoản ngân hàng.
- Xác nhận đặt phòng: Hệ thống Agoda gửi email xác nhận đặt phòng cho người dùng, bao gồm thông tin chi tiết về đặt phòng và số tiền đã thanh toán.
- Nhận phòng: Khi đến khách sạn, người dùng cần xuất trình email xác nhận đặt phòng và giấy tờ tùy thân để nhận phòng.
- Trả phòng: Khi hết thời gian đặt phòng, người dùng trả phòng và thanh toán các dịch vụ sử dụng thêm nếu có.
- Đánh giá: Sau khi trải nghiệm dịch vụ, người dùng có thể đánh giá khách sạn và chia sẻ trải nghiệm của mình trên trang web Agoda, giúp người dùng khác có thêm thông tin khi đặt phòng.

Trên đây là các quy trình nghiệp vụ đặt khách sạn của hệ thống Agoda. Quy trình này đảm bảo tính chính xác và đáng tin cậy của thông tin đặt phòng, giúp người dùng tiết kiệm thời gian và nâng cao trải nghiệm khi sử dụng dịch vụ.

1. Mô hình use-case nghiệp vụ



- Đặc tả:

Tên Use Case	**Search for hotel rooms**
Mô tả	- UC bắt đầu khi có khách hàng truy cập sử dụng dịch vụ tìm khách sạn.
Dòng cơ bản	1) Khách hàng chọn địa điểm muốn tìm khách sạn, chọn ngày nhận phòng, ngày trả phòng và số lượng phòng cần đặt. 2) Hệ thống ghi nhận thông tin và bắt đầu tìm kiếm khách sạn theo thông tin đã nhập. 3) Hệ thống hiển thị các khách sạn thoả mãn yêu cầu của khách hàng.
Dòng thay thế	- A1: Tại bước 1, nếu thông tin khách hàng nhập chưa đầy đủ, hệ thống hiển thị thông báo yêu cầu nhập lại thông tin. - A3: Tại bước 3, nếu hệ thống không tìm thấy bất cứ khách sạn nào thoả mãn, hiển thị thông báo không tìm thấy khách sạn.

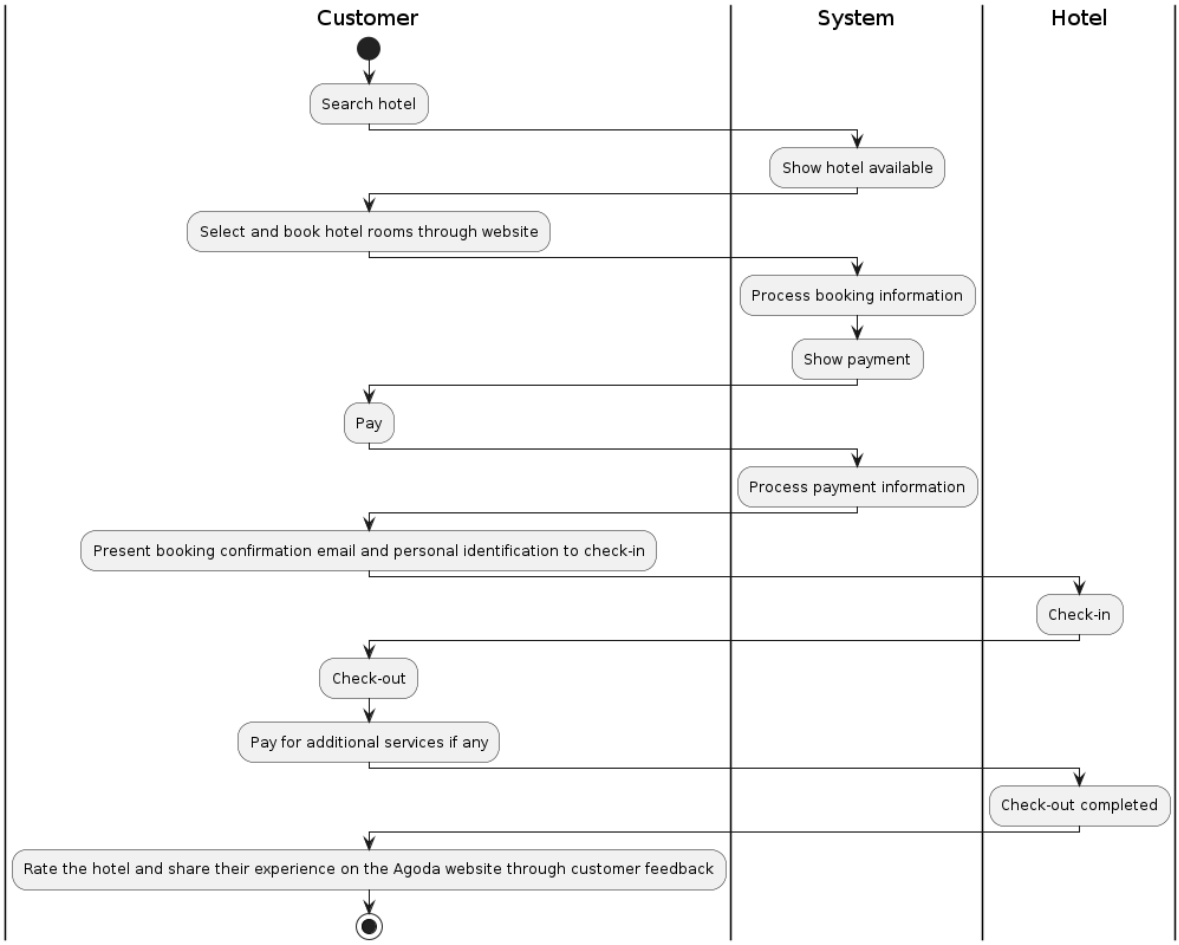
Tên Use Case	**Book hotel rooms**
Mô tả	- UC bắt đầu khi có khách hàng thực hiện thành công UC Search for hotel rooms
Dòng cơ bản	1) Khách hàng chọn khách sạn phù hợp với mong muốn trong danh sách khách sạn hệ thống hiển thị để xem thông tin chi tiết. 2) Hệ thống hiển thị thông tin chi tiết của khách sạn khách hàng đã chọn. 3) Khách hàng click “Đặt phòng” để hoàn tất. 4) Hệ thống ghi nhận thông tin và hiển thị thông báo đặt phòng thành công

Tên Use Case	**Payment**
Mô tả	- UC bắt đầu khi khách hàng thực hiện đặt phòng thành công
Dòng cơ bản	1) Hệ thống hiển thị thông tin giá phòng và các phương thức thanh toán. 2) Khách hàng chọn phương thức thanh toán. 3) Sau khi thực hiện đầy đủ các bước thanh toán, khách hàng click "Thanh toán" để hoàn tất 4) Hệ thống ghi nhận thông tin và hiển thị thông báo Thanh toán thành công
Dòng thay thế	- A3: Tại bước 3, nếu thông tin khách hàng nhập chưa đầy đủ, hệ thống hiển thị thông báo yêu cầu nhập lại thông tin. - A4: Tại bước 4, nếu hệ thống không ghi nhận được thông tin, hiển thị thông báo lỗi.

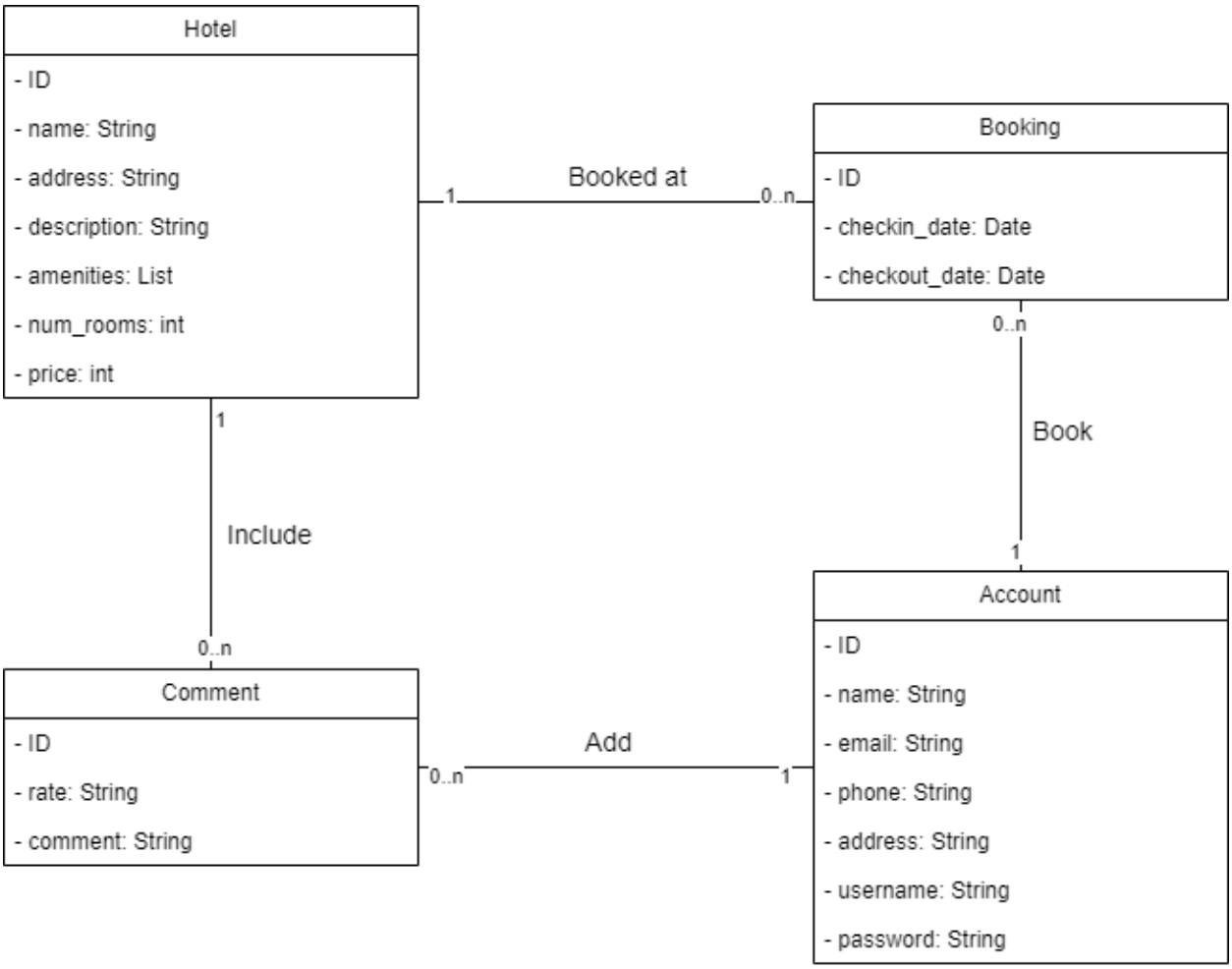
Tên Use Case	**Rate hotel & Share experiment**
Mô tả	- UC bắt đầu khi khách hàng đã thanh toán và trải qua kỳ nghỉ tại khách sạn đó hoặc khi khách hàng bắt đầu xem thông tin chi tiết của từng khách sạn
Dòng cơ bản	1) Khách hàng chọn khách sạn muốn xem đánh giá. 2) Hệ thống hiển thị thông tin các đánh giá thuộc về khách sạn đã chọn. 3) Khách hàng nhập thông tin đánh giá mong muốn của bản thân và click Đánh giá để hoàn tất. 4) Hệ thống ghi nhận thông tin và hiển thị thông báo Đánh giá thành công
Dòng thay thế	- A3: Tại bước 3, nếu khách hàng không có nhu cầu đánh giá. Bỏ qua các bước còn lại. - A4: Tại bước 4, nếu hệ thống không ghi nhận được thông tin, hiển thị thông báo lỗi.

Tên Use Case	Login
Mô tả	- UC bắt đầu khi khách hàng bắt đầu sử dụng dịch vụ
Dòng cơ bản	1) Khách hàng chọn chức năng đăng nhập. 2) Khách hàng nhập thông tin username và password. 3) Hệ thống ghi nhận thông tin và hiển thị thông báo Đăng nhập thành công
Dòng thay thế	- A3.1: Tại bước 3, nếu khách hàng không nhập đủ thông tin, hiển thị thông báo yêu cầu nhập lại. - A3.2: Tại bước 3, nếu hệ thống ghi nhận các thông tin đã nhập không đúng, hoặc không tồn tại, hiển thị thông báo lỗi.

- Sơ đồ hoạt động:

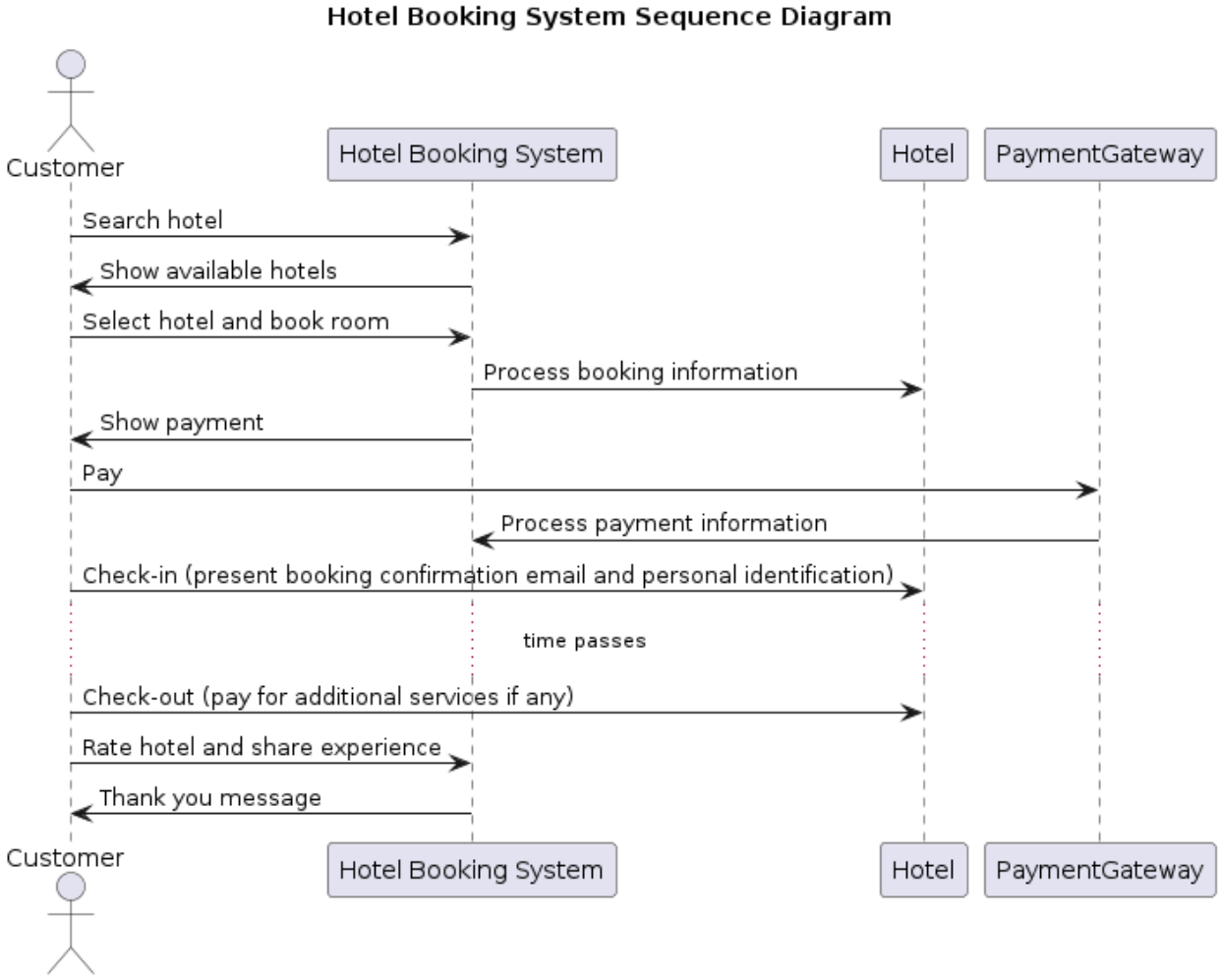


- Sơ đồ lớp mức phân tích:

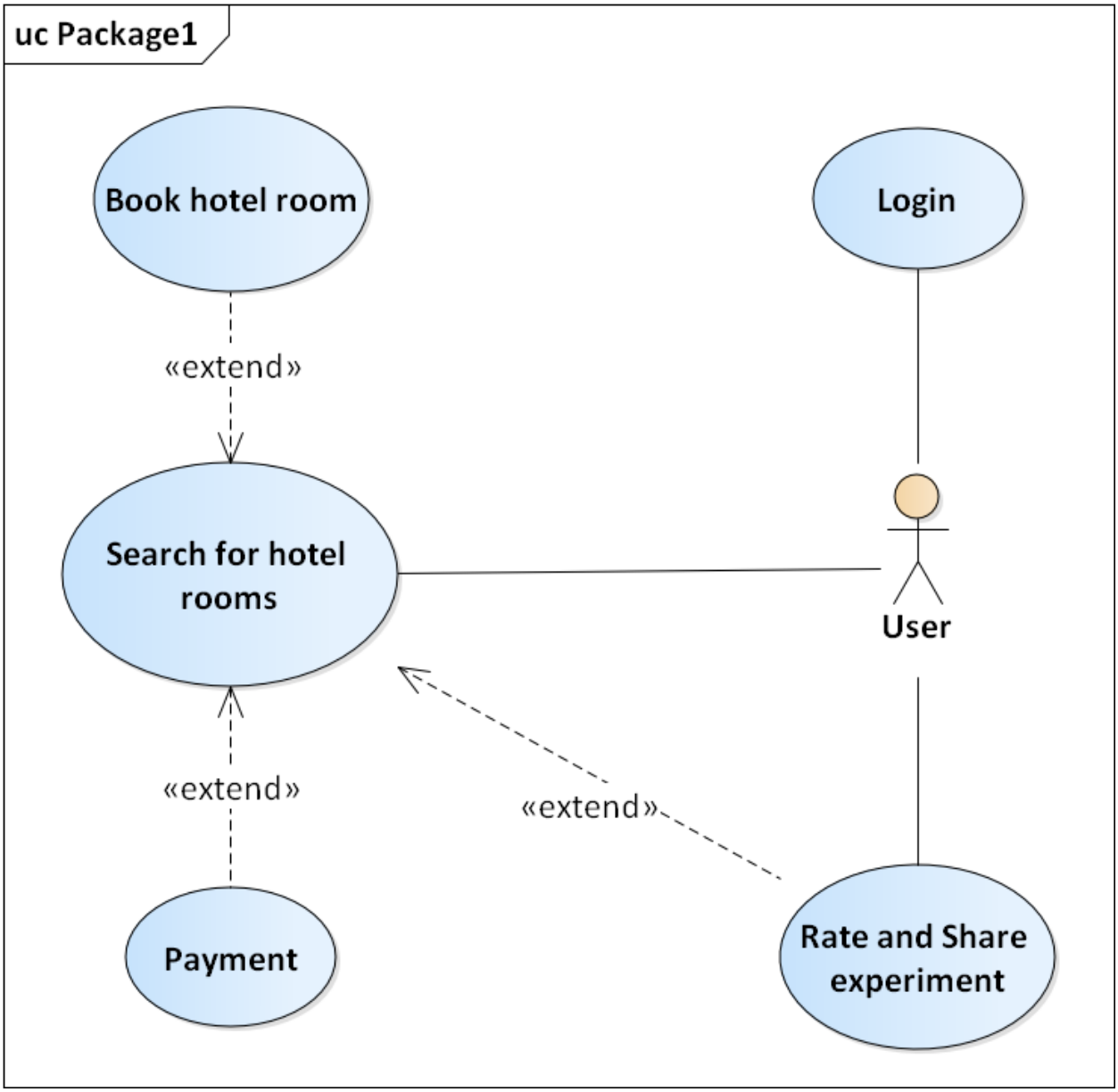


2. Mô hình Use-case xác định các yêu cầu tự động hoá

- Sequence Diagram:



- Mô hình UC chức năng:



- Đặc tả UC hệ thống:

Tên use case	**Search for hotel room**
Tóm tắt	Khách hàng tìm kiếm khách sạn.
Tác nhân	Khách hàng
Use case liên quan	Search for hotel room
Dòng sự kiện chính	1. Khách hàng nhập thông tin địa điểm muốn đến, ngày nhận phòng, ngày trả phòng và số phòng muốn đặt. 2. Khách hàng click nút "Tìm phòng". 3. Hệ thống hiển thị danh sách các khách sạn phù hợp với thông tin khách hàng đã nhập.
Dòng sự kiện phụ	A3. Tại bước 3 nếu không có thông tin nào được trả về, hệ thống thông báo "Không có kết quả".
Điều kiện tiên quyết	Nhân viên phải đăng nhập và chọn chức năng "Tìm phòng khách sạn".
Hậu điều kiện	Hệ thống hiện danh sách các khách sạn phù hợp

Tên use case	**Login**
Tóm tắt	Khách hàng đăng nhập để sử dụng dịch vụ
Tác nhân	Khách hàng
Use case liên quan	Login
Dòng sự kiện chính	1. Khách hàng nhập thông tin cá nhân và username, password mong muốn. 2. Khách hàng nhấn nút "đăng ký".

	3. Khách hàng nhập thông tin username và password đã có. 4. Khách hàng nhấn nút "đăng nhập". 5. Hệ thống hiển thị thông báo đăng nhập thành công và điều hướng đến trang chủ.
Dòng sự kiện phụ	A1. Tại bước 1, nếu khách hàng đã có tài khoản, bỏ qua bước 1 và 2. A5.1: Tại bước 5, nếu khách hàng chưa nhập đủ thông tin, hiển thị thông báo yêu cầu nhập lại. A5.2: Tại bước 5, nếu không có thông tin nào được trả về, hệ thống thông báo "Kiểm tra lại thông tin hoặc Đăng ký nếu chưa có tài khoản".
Điều kiện tiên quyết	
Hậu điều kiện	Hệ thống hiện thông báo đăng nhập thành công.

Tên use case	**Rate and Share experiment**
Tóm tắt	Khách hàng xem hoặc thêm đánh giá
Tác nhân	Khách hàng
Use case liên quan	Rate and Share experiment
Dòng sự kiện chính	1. Khách hàng chọn khách sạn muốn xem hoặc thêm đánh giá. 2. Hệ thống hiển thị danh sách các đánh giá của khách sạn mà khách hàng đã chọn. 3. Khách hàng nhập thông tin đánh giá và nhấn nút "đánh giá" để hoàn tất. 4. Hệ thống hiển thị thông báo Đánh giá thành công.

Dòng sự kiện phụ	A3. Tại bước 3 nếu khách hàng không có nhu cầu thêm đánh giá, bỏ qua các bước còn lại. A4. Tại bước 4, nếu khách hàng chưa nhập đủ thông tin, hiện thông báo yêu cầu nhập lại.
Điều kiện tiên quyết	Nhân viên phải đăng nhập và đã thực hiện thành công UC search for hotel rooms
Hậu điều kiện	Hệ thống hiện danh sách các đánh giá của khách sạn mong muốn.

Tên use case	**Payment**
Tóm tắt	Khách hàng thanh toán
Tác nhân	Khách hàng
Use case liên quan	Payment
Dòng sự kiện chính	1. Hệ thống hiển thị tổng giá tiền cần phải thanh toán. 2. Khách hàng chọn phương thức thanh toán. 3. Khách hàng tiến hành thanh toán. 4. Hệ thống hiển thị thông báo Thanh toán thành công.
Dòng sự kiện phụ	A4. Tại bước 4, nếu khách hàng thanh toán không thành công, hiện thông báo lỗi.
Điều kiện tiên quyết	Nhân viên phải đăng nhập và đã thực hiện thành công UC Book hotel room.
Hậu điều kiện	Hệ thống hiện thông báo thanh toán thành công

Tên use case	**Book hotel room**
Tóm tắt	Khách hàng chọn phòng mong muốn
Tác nhân	Khách hàng
Use case liên quan	Book hotel room
Dòng sự kiện chính	1. Khách hàng chọn khách sạn đặt phòng. 2. Hệ thống hiển thị các thông tin chi tiết của khách sạn khách hàng đã chọn. 3. Khách hàng nhấn nút "Đặt phòng" để hoàn tất. 4. Hệ thống hiển thị thông báo Đặt phòng thành công.
Dòng sự kiện phụ	
Điều kiện tiên quyết	Nhân viên phải đăng nhập và đã thực hiện thành công UC search for hotel rooms
Hậu điều kiện	Hệ thống hiện thông báo đặt phòng thành công.

III. Phân tích các yêu cầu lưu trữ, khả năng mở rộng và hiệu suất truy xuất khi sử dụng từng loại NoSQL cho hệ thống đặt phòng khách sạn Agoda

	Yêu cầu lưu trữ	Khả năng mở rộng	Hiệu suất truy xuất	Ứng dụng
Redis	Redis là một cơ sở dữ liệu key-value, phù hợp cho các ứng dụng yêu cầu lưu trữ dữ liệu nhanh và đơn giản. Redis hỗ trợ lưu trữ dữ liệu dạng key-value, list, set, sorted set và hash.	Redis hỗ trợ horizontal scaling với tính năng Redis Cluster, cho phép phân tán dữ liệu trên nhiều node để tăng khả năng mở rộng	Redis có hiệu suất truy xuất nhanh, hỗ trợ caching dữ liệu trong bộ nhớ và có khả năng xử lý hàng triệu truy vấn mỗi giây.	Khi khách hàng tìm kiếm phòng khách sạn trên Agoda, hệ thống sử dụng Redis để lưu trữ cache kết quả tìm kiếm của khách hàng. Khi khách hàng tiếp tục tìm kiếm hoặc truy cập lại trang web Agoda, hệ thống sẽ sử dụng cache được lưu trữ trên Redis thay vì thực hiện lại truy vấn tìm kiếm, giúp tăng tốc độ truy xuất và giảm tải cho cơ sở dữ liệu chính.
MongoDB	MongoDB là một cơ sở dữ liệu phi cấu trúc (document-oriented), phù hợp cho các ứng dụng yêu cầu lưu trữ dữ liệu phức tạp và đa dạng. MongoDB hỗ trợ lưu trữ dữ liệu dạng BSON (Binary JSON)	MongoDB có tính mở rộng ngang (horizontal scaling) tốt, cho phép phân tán dữ liệu trên nhiều node để tăng khả năng mở rộng	MongoDB có hiệu suất truy xuất nhanh và có thể xử lý hàng triệu truy vấn mỗi giây.	Khi khách hàng đặt phòng khách sạn trên Agoda, hệ thống sử dụng MongoDB để lưu trữ các thông tin về thanh toán và các dịch vụ sử dụng thêm của khách hàng sau khi kết thúc thời gian đặt phòng. MongoDB cho phép lưu trữ dữ liệu phi cấu trúc (document-oriented) và tính mở rộng ngang (horizontal scaling) tốt, giúp hệ thống xử lý các truy vấn đọc/ghi trên nhiều node đồng thời và đảm bảo tính sẵn sàng và độ tin cậy của hệ thống.

Cassandra	Cassandra là một cơ sở dữ liệu phân tán, phù hợp cho các ứng dụng yêu cầu lưu trữ dữ liệu lớn và có tính mở rộng cao. Cassandra hỗ trợ lưu trữ dữ liệu dạng column family.	Cassandra có tính mở rộng ngang (horizontal scaling) rất tốt, cho phép phân tán dữ liệu trên nhiều node và tự động điều chỉnh khi có thêm node mới hoặc node cũ bị lỗi.	Cassandra có khả năng xử lý hàng triệu truy vấn mỗi giây và có thể xử lý các truy vấn đọc/ghi trên nhiều node đồng thời	Khi khách hàng đặt phòng khách sạn trên Agoda, hệ thống sử dụng Cassandra để lưu trữ thông tin đặt phòng và thông tin liên quan như thông tin khách hàng, thông tin phòng, thời gian đến/đi. Cassandra cho phép lưu trữ dữ liệu phân tán trên nhiều node, đảm bảo tính sẵn sàng và độ tin cậy cao của hệ thống. Hệ thống cũng sử dụng Cassandra để lưu trữ lịch sử đặt phòng của khách hàng và phân tích dữ liệu đặt phòng để cải thiện trải nghiệm khách hàng.
Neo4j	Neo4j là một cơ sở dữ liệu đồ thị, phù hợp cho các ứng dụng yêu cầu truy vấn dữ liệu phức tạp và phân tích mối quan hệ giữa các đối tượng.	Neo4j có tính mở rộng ngang (horizontal scaling) hạn chế, vì vậy không phù hợp cho các ứng dụng yêu cầu mở rộng quy mô lớn.	Neo4j có hiệu suất truy xuất tốt cho các truy vấn đồ thị phức tạp, nhưng không phù hợp cho các ứng dụng yêu cầu xử lý lượng dữ liệu lớn.	Khi khách hàng tìm kiếm phòng khách sạn trên Agoda, hệ thống sử dụng Neo4j để xây dựng đồ thị mối quan hệ giữa các khách sạn, các phòng và các tiện nghi để giúp khách hàng tìm kiếm và so sánh thông tin phòng khách sạn dễ dàng hơn. Neo4j cũng được sử dụng để lưu trữ các đánh giá và phản hồi của khách hàng về các khách sạn, giúp cải thiện trải nghiệm khách hàng và tăng tính tương tác của hệ thống.

- Các yêu cầu cụ thể để nhóm áp dụng các CSDL noSQL:
  - Chức năng đăng ký tài khoản, đăng nhập: ta cần lưu trữ username và password:
    - Redis: tạo lưu dữ liệu dưới dạng key/value, theo cách này thì khả năng mở rộng thấp tuy nhiên hiệu suất truy vấn cực kỳ nhanh vì dữ liệu In-memory.
    - MongoDB: tạo collection account để lưu trữ, khả năng mở rộng cao và hiệu suất truy vấn nhanh.
    - Cassandra: ta tạo table account để lưu trữ thông tin, khả năng mở rộng cao và hiệu suất truy vấn nhanh.
    - Neo4j: Tạo các node account để lưu trữ thông tin, khả năng mở rộng cao và hiệu suất truy vấn nhanh.

      ➔ Nhóm chọn Redis cho chức năng đăng nhập để tối ưu hiệu suất và dễ tích hợp nhất. Thông tin username và password ít bị thay đổi nên sử dụng Redis để cache. Bên cạnh đó, sử dụng MongoDB để lưu trữ các thông tin account (do data lưu ở Redis rất khó để backup/restore khi có lỗi xảy ra vì lưu in-memory) để cải thiện khả năng mở rộng.
  - Chức năng tìm và lọc khách sạn:
    - Redis: tạo lưu dữ liệu dưới dạng key/value, theo cách này thì khả năng mở rộng thấp, một value sẽ rất nhiều data vì một khách sạn cần lưu rất nhiều data.
    - MongoDB: tạo collection hotel để lưu trữ, khả năng mở rộng cao và hiệu suất truy vấn nhanh vì một khách sạn có rất nhiều thông tin liên quan, có thể nhúng document để tối ưu.
    - Cassandra: ta tạo table hotel để lưu trữ 2 thông tin này, khả năng mở rộng cao và hiệu suất truy vấn nhanh.
    - Neo4j: Tạo các node account để lưu trữ 2 thông tin này, khả năng mở rộng cao và hiệu suất truy vấn nhanh.

➔ Nhóm chọn MongoDB cho chức năng tìm và lọc để tối ưu hiệu suất, dễ tích hợp nhất, và do không cần phải xác định schema trước nên dễ mở rộng nhất.

- Chức năng chọn phòng khách sạn:
  - Redis: tạo lưu dữ liệu dưới dạng key/value, theo cách này thì khả năng mở rộng thấp, khó truy vấn và một value sẽ rất nhiều data vì một khách sạn và khách hàng cần lưu rất nhiều data.
  - MongoDB: tạo collection booking để lưu trữ, khả năng mở rộng cao và hiệu suất truy vấn nhanh vì một khách sạn có rất nhiều thông tin liên quan, có thể nhúng document để tối ưu.
  - Cassandra: ta tạo table booking để lưu trữ, khả năng mở rộng cao và hiệu suất truy vấn nhanh.
  - Neo4j: Tạo các node booking để lưu trữ, khả năng mở rộng cao và hiệu suất truy vấn nhanh.

    ➔ Nhóm chọn MongoDB cho chức năng chọn để tối ưu hiệu suất và dễ tích hợp nhất, và do không cần phải xác định schema trước nên dễ mở rộng nhất.

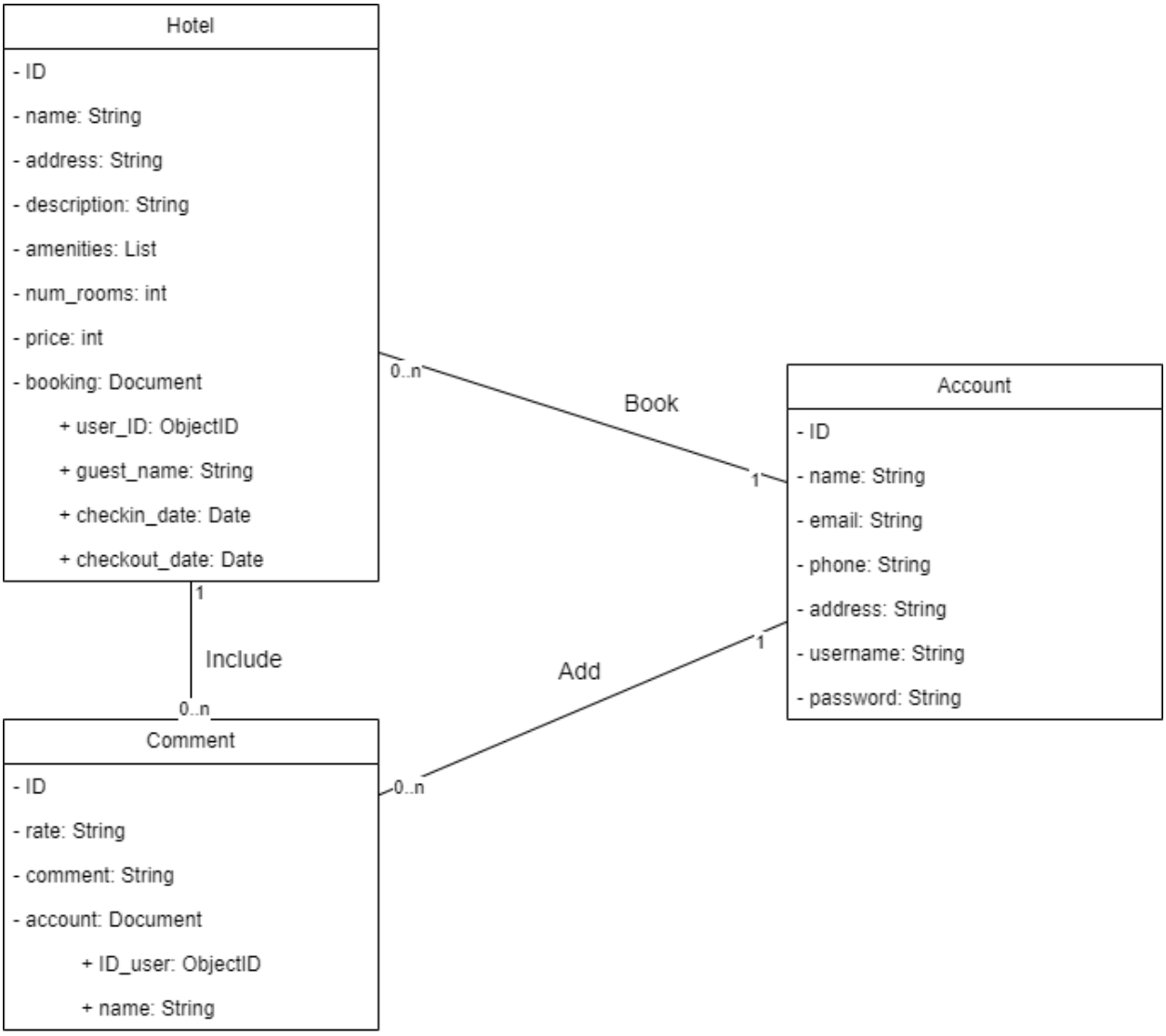
Tóm lại, nhóm chọn MongoDB để sử dụng nhiều nhất với lý do:

- Dễ mở rộng do không cần phải xác định schema.
- Cho phép nhúng document, không cần phải join bảng khi truy vấn.
- Hỗ trợ nhiều ngôn ngữ lập trình, dễ sử dụng cho người mới bắt đầu, phù hợp với team members.

IV. Lựa chọn và thiết kế dữ liệu phù hợp với các quy trình đã phân tích

1. Bảng thiết kế cơ sở dữ liệu

Với MongoDB được lưu dữ liệu dưới dạng file JSON, tuy nhiên CSDL có thể minh hoạ như sau:



2. Đề xuất về cải thiện hiệu quả truy vấn dựa trên thiết kế đề xuất

Theo nhóm thấy thì việc tìm khách sạn theo giá rất thường xuyên xảy ra với tần suất lớn. Vì vậy, nhóm đã quyết định cải thiện câu truy vấn này.

Đầu tiên, nhóm đã generate data cho collection hotels với số lượng 5000 dòng.

Sau đó, nhóm viết 1 API để lấy những hotels có giá từ 150000 đến 2000000.

Nhóm quyết định sử dụng K6 để fake request, và kiểm tra độ chịu tải của ứng dụng.

Mô tả quá trình kiểm tra hiệu suất câu truy vấn

Dùng K6 để fake request gọi câu truy vấn với số lượng lớn

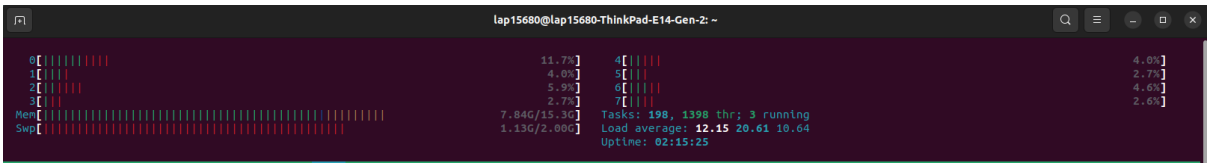
- Kịch bản: constant-arrival-rate với rate = [200, 500, 1000] (reqs/s)
- Thời gian: 2 m

```
export let options = {
   stages: [
    { duration: '0.5m', target: 200 },
    { duration: '0.5m', target: 500 },
    { duration: '0.5m', target: 1000 },
   { duration: '0.5m', target: 0 },
    ],
};

export default function () {
   group('Find hotels', () => {
       const url = "http://localhost:8080/api/v1/hotels?min_price=1500000&max_price=2000000";

       const response = http.get(url);
       check(response, {
           "status code should be 200": res => res.status === 200,
       });
   });
};
```

Chúng ta bắt đầu fake request với tình trạng RAM, CPU như sau:



Kết quả:

```
     ✓ status code should be 200

     checks.........................: 100.00% ✓ 137872      ✗ 0
     data_received..................: 976 MB  8.1 MB/s
     data_sent......................: 18 MB   148 kB/s
     group_duration.................: avg=371.56ms min=2.95ms  med=335.89ms max=1.48s    p(90)=763.8ms  p(95)=830.31ms
     http_req_blocked...............: avg=37.46µs  min=551ns   med=1.36µs   max=153.04ms p(90)=3.25µs   p(95)=4.29µs
     http_req_connecting............: avg=30.29µs  min=0s      med=0s       max=94.19ms  p(90)=0s       p(95)=0s
     http_req_duration..............: avg=371.37ms min=2.88ms  med=335.7ms  max=1.48s    p(90)=763.64ms p(95)=830.2ms
       { expected_response:true }...: avg=371.37ms min=2.88ms  med=335.7ms  max=1.48s    p(90)=763.64ms p(95)=830.2ms
     http_req_failed................: 0.00%   ✓ 0           ✗ 137872
     http_req_receiving.............: avg=235.91µs min=12.06µs med=27µs     max=175.44ms p(90)=77.39µs  p(95)=155.45µs
     http_req_sending...............: avg=27.16µs  min=2.88µs  med=7.24µs   max=146.14ms p(90)=14.69µs  p(95)=20.73µs
     http_req_tls_handshaking.......: avg=0s       min=0s      med=0s       max=0s       p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=371.11ms min=2.82ms  med=335.02ms max=1.48s    p(90)=763.49ms p(95)=830.04ms
     http_reqs......................: 137872  1148.907061/s
     iteration_duration.............: avg=371.59ms min=2.97ms  med=335.93ms max=1.48s    p(90)=763.82ms p(95)=830.35ms
     iterations.....................: 137872  1148.907061/s
     vus............................: 4       min=4         max=997
     vus_max........................: 1000    min=1000      max=1000


running (2m00.0s), 0000/1000 VUs, 137872 complete and 0 interrupted iterations
default ✓ [======================================] 0000/1000 VUs  2m0s
```

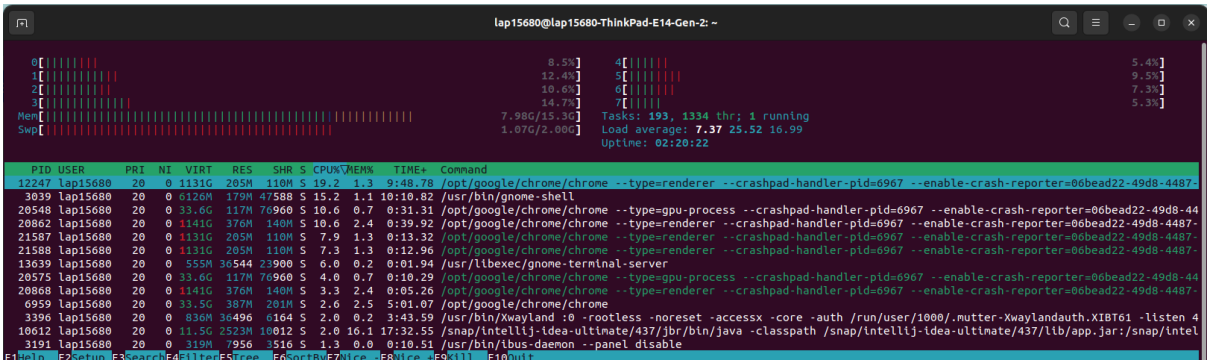
Nhận xét:

Tổng số request thực hiện được trong 2p	137872 request
Thời gian trung bình 1 request	371ms
Thời gian lâu nhất của 1 request	1.48s
Thời gian nhanh nhất của 1 request	2.88ms

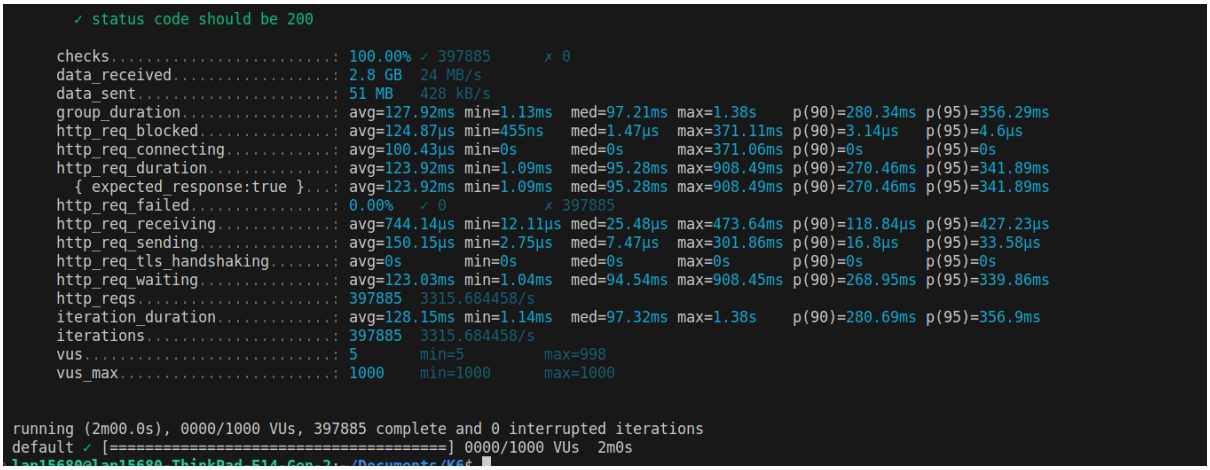
➔ Tối ưu câu truy vấn, ta sử dụng index trên field price

```
4    db.hotels.createIndex({price: 1})
```

Sau khi đánh index, ta tiến hành thực hiện lại thao tác trên



Kết quả fake request



Nhận xét sau khi đánh index.

Tổng số request thực hiện được trong 2p	397885
Thời gian trung bình 1 request	123ms
Thời gian lâu nhất của 1 request	908ms
Thời gian nhanh nhất của 1 request	1.09ms

➔Ta thấy, sau khi dùng index cho “price" thì tốc độ của câu query nhanh hơn 50% so với trước khi dùng index.

➔ Dùng index để cải thiện câu query là điều cần thiết.

3. Source code

Link Github: [cuongjackky/MDM19_HotelManagement (github.com)]