BỘ CÔNG THƯƠNG

**TRƯỜNG ĐẠI HỌC CÔNG THƯƠNG TP.HCM**

**KHOA: CÔNG NGHỆ THÔNG TIN**

A blue and black logo

AI-generated content may be incorrect.

**ĐỒ ÁN CUỐI KỲ**

**DỮ LIỆU NOSQL**

**ĐỀ TÀI: XÂY DỰNG HỆ THỐNG QUẢN LÝ THÔNG TIN TUYỂN DỤNG**

|  |  |  |
| --- | --- | --- |
| **GDHD** | **Nguyễn Thị Thu Tâm** | |
| **NHÓM** | **11** | |
| **THÀNH VIÊN** | **Đặng Trường Vũ** | **2001225908** |
|  | **Đoàn Duy Hiếu** | **2001221914** |
|  | **Lê Trương Công Hiếu** | **2001221414** |

**THÀNH PHỐ HỒ CHÍ MINH, THÁNG 11 NĂM 2025**

A blue and black logo

AI-generated content may be incorrect.

**ĐỒ ÁN CUỐI KỲ**

**DỮ LIỆU NOSQL**

**ĐỀ TÀI: XÂY DỰNG HỆ THỐNG QUẢN LÝ THÔNG TIN TUYỂN DỤNG**

|  |  |  |
| --- | --- | --- |
| **GDHD** | **Nguyễn Thị Thu Tâm** | |
| **NHÓM** | **11** | |
| **THÀNH VIÊN** | **Đặng Trường Vũ** | **2001225908** |
|  | **Đoàn Duy Hiếu** | **2001221914** |
|  | **Lê Trương Công Hiếu** | **2001221414** |

BỘ CÔNG THƯƠNG

**TRƯỜNG ĐẠI HỌC CÔNG THƯƠNG TP.HCM**

**KHOA: CÔNG NGHỆ THÔNG TIN**

**THÀNH PHỐ HỒ CHÍ MINH, THÁNG 11 NĂM 2025**

BẢNG PHÂN CÔNG CÔNG VIỆC

|  |  |  |  |
| --- | --- | --- | --- |
| MSSV | Họ và tên | Công việc | Tiến độ hoàn thành |
| 2001225908 | Đặng Trường Vũ | Xây dựng API  Word  Powerpoint | 100% |
| 2001221914 | Đoàn Duy Hiếu | FE | 100% |
| 2001221414 | Lê Trương Công Hiếu | FE | 100% |

MỤC LỤC

[BẢNG PHÂN CÔNG CÔNG VIỆC i](#_Toc213325916)

[MỤC LỤC ii](#_Toc213325917)

[MỤC LỤC ẢNH vii](#_Toc213325918)

[MỤC LỤC BẢNG x](#_Toc213325919)

[DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT xi](#_Toc213325920)

[CHƯƠNG 1: TỔNG QUAN VỀ REDIS 1](#_Toc213325921)

[1.1 Giới thiệu Redis 1](#_Toc213325922)

[1.1.1 Redis là gì? 1](#_Toc213325923)

[1.1.2 Các tính năng nổi bật của Redis 1](#_Toc213325924)

[1.1.3 Khi nào nên sử dụng Redis? 1](#_Toc213325925)

[1.1.4 Redis so với Cơ sở dữ liệu truyền thống 2](#_Toc213325926)

[1.2 Kiến trúc 3](#_Toc213325927)

[1.2.1 Thành phần Client-Server 3](#_Toc213325928)

[1.2.2 Persistence (Tính bền vững) 4](#_Toc213325929)

[1.2.3 Replication (Nhân bản) 6](#_Toc213325930)

[1.2.4 Cluster 8](#_Toc213325931)

[1.2.5 Sentinel 9](#_Toc213325932)

[1.2.6 Memory Management (Quản lý bộ nhớ) 11](#_Toc213325933)

[1.3 Hướng dẫn cài đặt 12](#_Toc213325934)

[1.3.1 Cài đặt Redis trên Windows 12](#_Toc213325935)

[1.3.2 Cài đặt Redis thông qua WSL (Windows Subsystem for Linux) 16](#_Toc213325936)

[1.3.3 Các công cụ GUI tiêu biểu cho Redis 18](#_Toc213325937)

[1.4 Cấu trúc dữ liệu trong Redis 19](#_Toc213325938)

[1.4.1 Strings 19](#_Toc213325939)

[1.4.2 Lists 21](#_Toc213325940)

[1.4.3 Sets 23](#_Toc213325941)

[1.4.4 Hashes 25](#_Toc213325942)

[1.4.5 Sorted Sets 27](#_Toc213325943)

[1.4.6 Bitmap 28](#_Toc213325944)

[1.4.7 HyperLogLog 30](#_Toc213325945)

[1.4.8 Streams 31](#_Toc213325946)

[1.4.9 Geo 33](#_Toc213325947)

[1.4.10 JSON 35](#_Toc213325948)

[1.4.11 TimeSeries 36](#_Toc213325949)

[1.5 Truy vấn Full-Text Search (FT) trong Redis 38](#_Toc213325950)

[1.5.1 Giới thiệu 38](#_Toc213325951)

[1.5.2 Lệnh cơ bản 38](#_Toc213325952)

[1.5.3 Ứng dụng thực tế 39](#_Toc213325953)

[1.5.4 Cách sử dụng FT 40](#_Toc213325954)

[1.5.5 Ví dụ minh họa 41](#_Toc213325955)

[1.6 Một số thao tác quản trị cơ bản 42](#_Toc213325956)

[1.6.1 Kiểm tra thông tin server 42](#_Toc213325957)

[1.6.2 Quản lý cơ sở dữ liệu 43](#_Toc213325958)

[1.6.3 Xóa dữ liệu 44](#_Toc213325959)

[1.6.4 Kiểm tra key và bộ nhớ 45](#_Toc213325960)

[1.7 Kết luận 46](#_Toc213325961)

[CHƯƠNG 2: PHÂN TÍCH VÀ THIẾT KẾ CƠ SỞ DỮ LIỆU NoSQL 47](#_Toc213325962)

[2.1 Bối cảnh 47](#_Toc213325963)

[2.2 Phân tích thị trường hiện tại 47](#_Toc213325964)

[2.3 Giải pháp đề xuất 47](#_Toc213325965)

[2.4 Phân tích yêu cầu hệ thống 48](#_Toc213325966)

[2.4.1 Yêu cầu chức năng 48](#_Toc213325967)

[2.4.2 Yêu cầu phi chức năng 48](#_Toc213325968)

[2.5 Lựa chọn công nghệ: So sánh SQL và NoSQL 49](#_Toc213325969)

[2.5.1 Phân tích đặc thù của hệ thống tuyển dụng 49](#_Toc213325970)

[2.5.2 So sánh SQL và NoSQL trong bối cảnh tuyển dụng 50](#_Toc213325971)

[2.5.3 Hạn chế của NoSQL và cách khắc phục 51](#_Toc213325972)

[2.6 Thiết kế cơ sở dữ liệu NoSQL 52](#_Toc213325973)

[2.6.1 Triết lý thiết kế: Embedded vs Reference 52](#_Toc213325974)

[2.6.2 Mô hình dữ liệu chi tiết 52](#_Toc213325975)

[2.7 So sánh hiệu năng truy vấn 55](#_Toc213325976)

[2.7.1 Kịch bản 1: Hiển thị chi tiết tin tuyển dụng với danh sách ứng viên 55](#_Toc213325977)

[2.7.2 Kịch bản 2: Tìm kiếm công việc theo nhiều tiêu chí 56](#_Toc213325978)

[2.7.3 Kịch bản 3: Cập nhật trạng thái ứng viên 57](#_Toc213325979)

[2.7.4 Tổng kết so sánh hiệu năng 60](#_Toc213325980)

[2.8 Phân tích Thiết kế Cơ chế Lưu trữ Đệm với Redis 60](#_Toc213325981)

[2.8.1 Cơ chế Quản lý Phiên 61](#_Toc213325982)

[2.8.2 Chiến lược Caching Kết quả Truy vấn 61](#_Toc213325983)

[2.9 Kết luận 63](#_Toc213325984)

[CHƯƠNG 3: TRIỂN KHAI ỨNG DỤNG 64](#_Toc213325985)

[3.1 Giới thiệu chương 64](#_Toc213325986)

[3.2 Tổng quan kiến trúc hệ thống 64](#_Toc213325987)

[3.2.1 Kiến trúc tổng thể 64](#_Toc213325988)

[3.2.2 Các tầng chính 66](#_Toc213325989)

[3.2.3 Vai trò của các tác nhân 67](#_Toc213325990)

[3.3 Hệ thống cơ sở dữ liệu và lưu trữ 68](#_Toc213325991)

[3.3.1 Tích hợp MongoDB 68](#_Toc213325992)

[3.3.2 Cấu hình, kết nối và Repository Pattern 68](#_Toc213325993)

[3.3.3 Cấu trúc dữ liệu, phân trang và lọc vai trò 71](#_Toc213325994)

[3.4 Cơ chế lưu trữ tạm và tối ưu hiệu năng (Redis) 73](#_Toc213325995)

[3.4.1 Redis Cache và cơ chế hoạt động 73](#_Toc213325996)

[3.4.2 Tích hợp Redis trong ASP.NET Core 73](#_Toc213325997)

[3.4.3 Cache-aside và TTL 75](#_Toc213325998)

[3.4.4 Invalidation theo scope 77](#_Toc213325999)

[3.4.5 So sánh hiệu năng 78](#_Toc213326000)

[3.5 Cấu hình môi trường và triển khai API 79](#_Toc213326001)

[3.5.1 Thiết lập dự án 79](#_Toc213326002)

[3.5.2 appsettings và Dependency Injection 80](#_Toc213326003)

[3.5.3 Các nhóm API chính 82](#_Toc213326004)

[3.6 Giao diện triển khai 82](#_Toc213326005)

[3.6.1 Login/Register 82](#_Toc213326006)

[3.6.2 Dashboard Admin/Recruiter 85](#_Toc213326007)

[3.6.3 Job Listing/Detail 86](#_Toc213326008)

[3.6.4 Company Management/Candidate Profile 89](#_Toc213326009)

[3.6.5 Admin 91](#_Toc213326010)

[3.7 Kết luận chương 93](#_Toc213326011)

[CHƯƠNG 4: KẾT LUẬN 95](#_Toc213326012)

[TÀI LIỆU THAM KHẢO 96](#_Toc213326013)

MỤC LỤC ẢNH

[Hình 1.1: Mô hình Client - Server 4](#_Toc213326318)

[Hình 1.2: AOF 5](#_Toc213326319)

[Hình 1.3: RDB 5](#_Toc213326320)

[Hình 1.4: Replication 7](#_Toc213326321)

[Hình 1.5: Cluster 9](#_Toc213326322)

[Hình 1.6: Sentinel 10](#_Toc213326323)

[Hình 1.7: Tài nguyên tải về 13](#_Toc213326324)

[Hình 1.8: Các tệp trong thư mục nén tải về 13](#_Toc213326325)

[Hình 1.9: Chạy lệnh ping kiểm tra kết nối 15](#_Toc213326326)

[Hình 1.10: Bật tính năng WSL trên Windows 16](#_Toc213326327)

[Hình 1.11: Cài đặt Ubuntu từ Microsoft Store 16](#_Toc213326328)

[Hình 1.12: Cài đặt 17](#_Toc213326329)

[Hình 1.13: Tải redis 17](#_Toc213326330)

[Hình 1.14: Chạy lệnh ping kiểm tra 18](#_Toc213326331)

[Hình 1.15: Cấu trúc dữ liệu 19](#_Toc213326332)

[Hình 3.1: Sơ đồ kiến trúc hệ thống 65](#_Toc213326333)

[Hình 3.2: Khởi tạo MongoDbContext 69](#_Toc213326334)

[Hình 3.3: Triển khai lớp JobRepository 70](#_Toc213326335)

[Hình 3.4: JSON Example đầy đủ của Job Document 72](#_Toc213326336)

[Hình 3.5: Code đầy đủ RedisCacheService 74](#_Toc213326337)

[Hình 3.6: Code snippet GetJobs action với cache-aside và key strategy 76](#_Toc213326338)

[Hình 3.7: Code snippet PublishJob action với invalidation logic 77](#_Toc213326339)

[Hình 3.8: Kết quả test trên route “Get profile” 79](#_Toc213326340)

[Hình 3.9: Kết quả test trên route “Get All Jobs” 79](#_Toc213326341)

[Hình 3.10: appsettings.json 80](#_Toc213326342)

[Hình 3.11: Program.cs 81](#_Toc213326343)

[Hình 3.12: Giao diện đăng nhập - Admin 91](#_Toc213326344)

[Hình 3.13: Giao diện dashboard - Admin 91](#_Toc213326345)

[Hình 3.14: Giao diện Quản lý công việc - Admin 92](#_Toc213326346)

[Hình 3.15: Giao diện chi tiết công việc - Admin 92](#_Toc213326347)

[Hình 3.16: Giao diện quản lý công ty - Admin 93](#_Toc213326348)

[Hình 3.17: Giao diện thông tin công ty - Admin 93](#_Toc213326349)

MỤC LỤC BẢNG

[Bảng 1.1: Bảng so sánh Redis với RDBMS 3](#_Toc213326350)

[Bảng 1.2: Lệnh cơ bản với Strings 19](#_Toc213326351)

[Bảng 1.3: Lệnh cơ bản với List 22](#_Toc213326352)

[Bảng 1.4: Lệnh cơ bản với Set 23](#_Toc213326353)

[Bảng 1.5: Lệnh cơ bản với Hash 25](#_Toc213326354)

[Bảng 1.6: Lệnh cơ bản với Sorted Set 27](#_Toc213326355)

[Bảng 1.7: Lệnh cơ bản với Bitmap 28](#_Toc213326356)

[Bảng 1.8: Lệnh cơ bản với HyperLogLog 30](#_Toc213326357)

[Bảng 1.9: Lệnh cơ bản với Stream 32](#_Toc213326358)

[Bảng 1.10: Lệnh cơ bản với Geo 33](#_Toc213326359)

[Bảng 1.11: Lệnh cơ bản với JSON 35](#_Toc213326360)

[Bảng 1.12: Lệnh cơ bản với TimeSeries 36](#_Toc213326361)

[Bảng 1.13: Lệnh cơ bản khi truy vấn Full-Text Search (FT) 38](#_Toc213326362)

[Bảng 1.14: Lệnh cơ bản để kiểm tra thông tin server 43](#_Toc213326363)

[Bảng 1.15: Lệnh cơ bản quản trị cơ sở dữ liệu 44](#_Toc213326364)

[Bảng 1.16: Lệnh cơ bản dùng cho “Xóa dữ liệu” 44](#_Toc213326365)

[Bảng 1.17: Lệnh cơ bản dùng cho kiểm tra key và bộ nhớ 45](#_Toc213326366)

[Bảng 2.1: Bảng tổng kết so sánh hiệu năng 60](#_Toc213326367)

[Bảng 3.1 So sánh hiệu năng Cache Hit và Cache Miss bằng Postman 78](#_Toc213326368)

DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT

|  |  |  |
| --- | --- | --- |
| **Viết tắt** | **Tiếng Anh** | **Tiếng Việt** |
| RDBMS | Relational Database Management System | Hệ quản trị cơ sở dữ liệu quan hệ |
| CRUD | Create – Read – Update – Delete | Bốn thao tác cơ bản: Tạo – Đọc – Cập nhật – Xóa |
| RAM | Random Access Memory | Bộ nhớ truy cập ngẫu nhiên |
| CSDL | – | Cơ sở dữ liệu |
| WSL | Windows Subsystem for Linux | Hệ thống con Windows cho Linux |
| AOF | Append Only File | Tệp chỉ ghi thêm (Nhật ký cam kết) |
| RDB | Redis Database | Cơ sở dữ liệu Redis (Tệp Snapshot) |
| Redis | REmote DIctionary Server | Hệ thống lưu trữ cấu trúc dữ liệu trong bộ nhớ |
| NoSQL | Not Only SQL / Non-Relational SQL | Cơ sở dữ liệu phi quan hệ |
| SQL | Structured Query Language | Ngôn ngữ truy vấn có cấu trúc |
| TTL | Time To Live | Thời gian sống (thời hạn tồn tại của dữ liệu) |
| JWT | JSON Web Token | Token Web JSON |
| FT | Full-Text Search | Truy vấn tìm kiếm toàn văn |
| RTO | Recovery Time Objective | Mục tiêu thời gian khôi phục |
| GUI | Graphical User Interface | Giao diện người dùng đồ họa |
| I/O | Input/Output | Vào/Ra (trong bối cảnh vòng lặp I/O) |
| SDS | Simple Dynamic Strings | Chuỗi động đơn giản |
| LRU | Least Recently Used | Ít được sử dụng gần đây nhất (Chính sách trục xuất bộ nhớ) |
| DI | Dependency Injection | Cơ chế tiêm phụ thuộc |
| DTOs | Data Transfer Objects | Đối tượng truyền tải dữ liệu |
| API | Application Programming Interface | Giao diện lập trình ứng dụng |

# TỔNG QUAN VỀ REDIS

## Giới thiệu Redis

### Redis là gì?

Redis (REmote DIctionary Server) là một hệ thống lưu trữ cấu trúc dữ liệu trong bộ nhớ (in-memory data structure store), mã nguồn mở. Redis có thể được sử dụng như một cơ sở dữ liệu, bộ nhớ đệm (cache), message broker, hoặc streaming engine.

Khác với cơ sở dữ liệu truyền thống lưu trữ trên đĩa, Redis lưu dữ liệu trong RAM, từ đó đạt được tốc độ xử lý cực nhanh và hiệu quả trong nhiều trường hợp sử dụng.

Hãy hình dung Redis giống như một kho key-value hiệu năng cao - tương tự một từ điển hoặc hash map khổng lồ - nhưng toàn bộ dữ liệu được lưu trong RAM. Chính nhờ thiết kế này mà Redis có thể xử lý hàng trăm nghìn lệnh mỗi giây với độ trễ rất thấp.

### Các tính năng nổi bật của Redis

Redis mang lại nhiều ưu điểm khiến nó trở nên phổ biến trong giới phát triển:

* Tốc độ cao: Hoạt động hoàn toàn trong bộ nhớ giúp Redis có độ trễ dưới 1ms.
* Đa dạng kiểu dữ liệu: Hỗ trợ nhiều cấu trúc như *string, hash, list, set, sorted set,...*
* Khả năng lưu bền (Persistence): Có thể lưu dữ liệu xuống đĩa định kỳ hoặc ghi log các thao tác để đảm bảo an toàn dữ liệu.
* Replication: Hỗ trợ mô hình master-slave (leader-follower replication) để tăng tính sẵn sàng.
* Clustering: Có thể triển khai kiến trúc phân tán.
* Lua scripting: Cho phép chạy script tuỳ chỉnh ngay trong Redis.
* Transactions: Hỗ trợ giao dịch cơ bản.
* Pub/Sub: Cung cấp mô hình publish/subscribe cho việc nhắn tin thời gian thực.

### Khi nào nên sử dụng Redis?

Redis đặc biệt phù hợp trong các tình huống yêu cầu tốc độ cao và xử lý tức thì:

* Caching: Lưu trữ dữ liệu truy cập thường xuyên để giảm tải cho database chính.
* Quản lý session: Lưu trữ session người dùng trong các ứng dụng web.
* Phân tích thời gian thực: Đếm và theo dõi các sự kiện ngay khi chúng xảy ra.
* Leaderboard/Counting: Sử dụng sorted set để xây dựng bảng xếp hạng hoặc bộ đếm.
* Hàng đợi (Queues): Dùng list để xử lý các tác vụ theo thứ tự.
* Giao tiếp thời gian thực: Triển khai chat, messaging với Pub/Sub.

### Redis so với Cơ sở dữ liệu truyền thống

**Sơ đồ so sánh**

A diagram of a diagram

AI-generated content may be incorrect.

*Giải thích sơ đồ:*

Một yêu cầu từ Client sẽ được gửi đến Database:

* Nếu cơ sở dữ liệu được chọn là Redis, dữ liệu sẽ được xử lý trực tiếp trong bộ nhớ (In-Memory Processing) và phản hồi lại rất nhanh.
* Nếu cơ sở dữ liệu được chọn là RDBMS truyền thống, dữ liệu sẽ được xử lý thông qua truy xuất ổ đĩa (Disk I/O) trước khi phản hồi, nên tốc độ chậm hơn.

**Redis có thay thế hoàn toàn RDBMS không?**

Redis không được thiết kế để thay thế toàn bộ cơ sở dữ liệu truyền thống trong mọi tình huống. Thay vào đó, mỗi hệ thống có ưu điểm riêng. Dưới đây là bảng so sánh:

Bảng .: Bảng so sánh Redis với RDBMS

|  |  |  |
| --- | --- | --- |
| Tiêu chí | Redis | Cơ sở dữ liệu truyền thống (RDBMS) |
| Lưu trữ dữ liệu | Trong bộ nhớ (RAM), có thể cấu hình thêm lưu bền | Dựa trên ổ đĩa |
| Ngôn ngữ truy vấn | Lệnh đơn giản | SQL hoặc tương tự |
| Hỗ trợ cấu trúc dữ liệu | Nhiều cấu trúc dựng sẵn: string, hash, list, set, sorted set, … | Bảng với khả năng quan hệ |
| Hỗ trợ giao dịch | Cơ bản (Basic transaction) | Tuân thủ ACID đầy đủ |
| Yêu cầu bộ nhớ | Cao hơn (dữ liệu nằm trong RAM) | Thấp hơn (dùng ổ đĩa) |
| Tốc độ | Rất nhanh (microseconds) | Trung bình đến nhanh (milliseconds) |

## Kiến trúc

### Thành phần Client-Server

**Redis Server**: Là lõi của Redis, xử lý các yêu cầu, quản lý dữ liệu và đảm bảo tính bền vững. Redis là một kho khóa-giá trị trong bộ nhớ.

* Mô hình đơn luồng: Redis xử lý tất cả các lệnh tuần tự bằng một luồng duy nhất, giúp tránh sự phức tạp và chi phí của các cơ chế đồng bộ hóa đa luồng.
* Vòng lặp sự kiện: Server sử dụng kiến trúc hướng sự kiện với vòng lặp I/O không chặn (epoll, kqueue, select) để xử lý hàng ngàn kết nối đồng thời mà không bị trễ.

**Client Libraries**: Hỗ trợ nhiều ngôn ngữ, gồm có hai loại là client trực tiếp và client hỗ trợ Object Mapping.

**Hoạt động dữ liệu**: Redis hỗ trợ phong phú các kiểu dữ liệu giá trị (Strings, Lists, Sets, Hashes...) và các thao tác CRUD cơ bản.

A diagram of a server

AI-generated content may be incorrect.

Hình .: Mô hình Client - Server

### Persistence (Tính bền vững)

Redis lưu trữ dữ liệu trong bộ nhớ để truy cập siêu nhanh, nhưng cũng có các tùy chọn lưu trữ trên đĩa để phục hồi dữ liệu. Có 4 tùy chọn chính:

**AOF (Append Only File):**

* Hoạt động như một nhật ký cam kết, ghi lại mọi thao tác ghi vào Redis.
* Có ba chế độ: **Always** (đồng bộ mọi lệnh, bền nhất nhưng chậm nhất), **Everysec** (đồng bộ mỗi giây, cân bằng), và **No** (để HĐH xử lý, nhanh nhất nhưng kém bền hơn).

A diagram of a server

AI-generated content may be incorrect.

Hình .: AOF

* Cấu hình:

|  |
| --- |
| # Redis configuration for AOF  appendonly yes  appendfsync everysec # Sync AOF once per second |

**RDB (Redis Database):**

* Thực hiện các snapshot tại các khoảng thời gian được định trước.
* Snapshot được tạo bằng cách sử dụng một tiến trình con (forked process). Tiến trình cha tiếp tục phục vụ các client, trong khi tiến trình con ghi dữ liệu vào đĩa.
* **Ưu điểm** là các file nhỏ gọn và ít tác động đến hiệu suất.
* **Nhược điểm** là có nguy cơ mất dữ liệu giữa các snapshot.

A close-up of a logo

AI-generated content may be incorrect.

Hình .: RDB

* Cấu hình:

|  |
| --- |
| # Redis configuration for RDB  save 900 1 # Create a snapshot if at least 1 key changes in 900 seconds  save 300 10 # Create a snapshot if at least 10 keys change in 300 seconds  save 60 10000 # Create a snapshot if at least 10000 keys change in 60 seconds |

**AOF và RDB (Hybrid Persistence):**

* Phương pháp này kết hợp ưu điểm của cả AOF và RDB.
* Trong quá trình phục hồi, Redis tải snapshot RDB trước và sau đó phát lại các log AOF gần đây, giúp tăng tốc độ phục hồi.
* Cấu hình:

|  |
| --- |
| # Redis using both persistence methods  appendonly yes  save 900 1 |

**No Persistence:**

* Thường được sử dụng khi Redis đóng vai trò là cache cho các tập dữ liệu nhỏ hoặc dữ liệu ít quan trọng, nơi việc mất dữ liệu không phải là vấn đề quá lớn. Khi Redis sập, toàn bộ dữ liệu sẽ mất.

### Replication (Nhân bản)

Redis áp dụng cơ chế replication theo mô hình master–slave (leader–follower) để đảm bảo tính sẵn sàng cao:

* Master (Leader): Xử lý tất cả các thao tác ghi và truyền các thay đổi dữ liệu đến các bản sao.
* Slaves (Followers): Hoạt động như bản sao chỉ đọc của master, hỗ trợ failover và có thể xử lý nhiều yêu cầu đọc song song. Theo mặc định, các bản sao từ chối lệnh ghi để giữ tính nhất quán dữ liệu.

Quá trình replication diễn ra không đồng bộ (asynchronous), nhờ đó master vẫn duy trì hiệu suất cao và độ trễ thấp. Các bản sao sẽ định kỳ xác nhận dữ liệu đã nhận được.

Mỗi master được gắn với Replication ID và offset (tự động tăng theo từng thao tác) để theo dõi trạng thái đồng bộ.

Redis hỗ trợ hai hình thức đồng bộ:

* Đồng bộ toàn bộ (Full Sync): Khi bản sao không có offset phù hợp hoặc không biết offset, master sẽ tạo một RDB snapshot và gửi sang. Trong quá trình này, master lưu tạm (buffer) các lệnh ghi để đồng bộ sau.
* Đồng bộ một phần (Partial Sync): Khi master và replica chia sẻ cùng một ancestor ID, hệ thống chỉ truyền các phần dữ liệu chênh lệch dựa trên offset, giúp tiết kiệm thời gian và băng thông.

A diagram of a diagram

AI-generated content may be incorrect.

Hình .: Replication

Cấu hình:

|  |
| --- |
| # On the replica server  replicaof 192.168.1.100 6379 |

### Cluster

Redis Cluster cho phép mở rộng theo chiều ngang bằng cách phân vùng dữ liệu trên nhiều node, dựa trên cơ chế master–slave replication.

Một cluster bao gồm nhiều shard, trong đó mỗi shard có một node primary và có thể kèm theo nhiều node replica.

**Các đặc điểm chính:**

* Sharding: Dữ liệu được tự động phân vùng và phân phối trên các node primary khi số lượng shard tăng.
* Hash Slots: Redis Cluster chia dữ liệu thành 16384 hash slots. Khóa được ánh xạ vào slot bằng hàm CRC16(key) mod 16384, và mỗi node chịu trách nhiệm cho một nhóm slot.
* Hash Tags: Hỗ trợ thao tác đa khóa bằng cách sử dụng dấu ngoặc nhọn {} để đảm bảo các khóa liên quan nằm trong cùng một slot.
* Resharding: Khi thêm hoặc bớt shard, hệ thống di chuyển các hash slot thay vì dữ liệu trực tiếp, giúp giảm chi phí và hạn chế ảnh hưởng đến tính sẵn sàng.
* Failover: Nếu một master node gặp sự cố, một replica sẽ được thăng cấp thành master mới.
* Gossiping: Sử dụng giao thức gossip để giám sát tình trạng cluster và thay thế node khi cần.
* Sharded Pub/Sub (Redis 7.0): Cho phép Pub/Sub theo shard, chỉ gửi tin nhắn trong shard cụ thể, hiệu quả hơn so với Pub/Sub toàn cluster.

**Ưu điểm:**

* Phân vùng dữ liệu tự động.
* Mở rộng cả đọc và ghi.
* Đảm bảo hoạt động liên tục ngay cả khi một phần node gặp lỗi.

**Nhược điểm:**

* Cần client tương thích với Redis Cluster.
* Thiết kế ứng dụng phải tính đến ràng buộc thao tác đa khóa.
* Kiến trúc phức tạp hơn khi vận hành ở quy mô lớn.

A diagram of a computer program

AI-generated content may be incorrect.

Hình .: Cluster

**Cấu hình:**

|  |
| --- |
| # Creating a simple 3-node cluster  redis-cli --cluster create 127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 |

### Sentinel

Redis Sentinel là một hệ thống phân tán được thiết kế để quản lý và giám sát Redis, với nhiều tiến trình Sentinel hoạt động phối hợp cùng nhau.

**Chức năng chính:**

* Tính sẵn sàng cao: Liên tục kiểm tra tình trạng của các instance primary và replica.
* Tự động Failover: Khi primary gặp sự cố, Sentinel sẽ thăng cấp một replica thành primary mới và cấu hình lại các replica khác để theo dõi node mới này.
* Quorum-based: Các quyết định failover được thông qua bởi đa số Sentinel, ngăn chặn tình huống split-brain.
* Service Discovery: Client có thể hỏi Sentinel để biết địa chỉ primary hiện tại. Khi failover xảy ra, Sentinel sẽ cung cấp thông tin cập nhật.
* Thông báo: Có khả năng gửi cảnh báo cho quản trị viên hoặc hệ thống liên quan về các sự kiện quan trọng.

**Ưu điểm:**

* Tự động failover khi có sự cố.
* Hỗ trợ service discovery cho client.

**Nhược điểm:**

* Kiến trúc phức tạp hơn vì cần duy trì một nhóm Sentinel riêng.
* Client phải hỗ trợ Sentinel.

A diagram of a company

AI-generated content may be incorrect.

Hình .: Sentinel

**Cấu hình:**

|  |
| --- |
| # sentinel.conf  sentinel monitor mymaster 127.0.0.1 6379 2  sentinel down-after-milliseconds mymaster 5000  sentinel failover-timeout mymaster 60000 |

### Memory Management (Quản lý bộ nhớ)

Redis lưu trữ toàn bộ dữ liệu trong RAM để đạt tốc độ truy cập cực nhanh, vì vậy việc quản lý bộ nhớ hiệu quả là yếu tố then chốt.

**Cấu trúc dữ liệu trong bộ nhớ**

* Hash Table: Lưu trữ tất cả cặp key–value, tra cứu O(1). Khi xảy ra xung đột hash, Redis dùng phương pháp chaining và thực hiện rehash bằng hai bảng hash toàn cục khi cần.
* SDS (Simple Dynamic Strings): Chuỗi động được cải tiến từ C-string, hỗ trợ cả dữ liệu nhị phân. Có thuộc tính len (độ dài) và free (vùng trống cấp phát sẵn), giúp thao tác chuỗi O(1) và an toàn bộ nhớ.
* Ziplist: Cấu trúc nhỏ gọn cho list/hash nhỏ, dữ liệu lưu liên tục trong bộ nhớ. Redis 7.0 thay Ziplist bằng Listpack và dùng Quicklist cho list lớn.
* Intset: Cấu trúc tối ưu cho các tập hợp số nguyên nhỏ.
* Quicklist: Kết hợp linked list và ziplist, được dùng cho các danh sách lớn.

**Chính sách trục xuất (Eviction Policies)**

Khi đạt giới hạn bộ nhớ, Redis áp dụng chiến lược trục xuất để giải phóng không gian (cấu hình ví dụ):

|  |
| --- |
| maxmemory 1gb  maxmemory-policy allkeys-lru |

Các chính sách bao gồm:

* Noeviction: Từ chối ghi thêm dữ liệu mới.
* Volatile-lru: Xóa key ít dùng gần đây, chỉ trong tập có TTL.
* Allkeys-lru: Xóa key ít dùng gần đây, áp dụng cho toàn bộ key.
* Volatile-ttl: Xóa key sắp hết hạn trước.
* Allkeys-random: Xóa key ngẫu nhiên.

**Phân mảnh bộ nhớ**

Redis mặc định dùng Jemalloc để quản lý cấp phát và giảm phân mảnh bộ nhớ.

**Hết hạn dữ liệu & cơ chế xóa**

* TTL (Time To Live): Key có thể cài đặt thời gian sống.
* Lazy Deletion: Key chỉ bị xóa khi được truy cập lại sau khi hết hạn → tiết kiệm CPU.
* Active Expiration: Tiến trình nền định kỳ quét và xóa key hết hạn để giải phóng bộ nhớ kịp thời.

## Hướng dẫn cài đặt

### Cài đặt Redis trên Windows

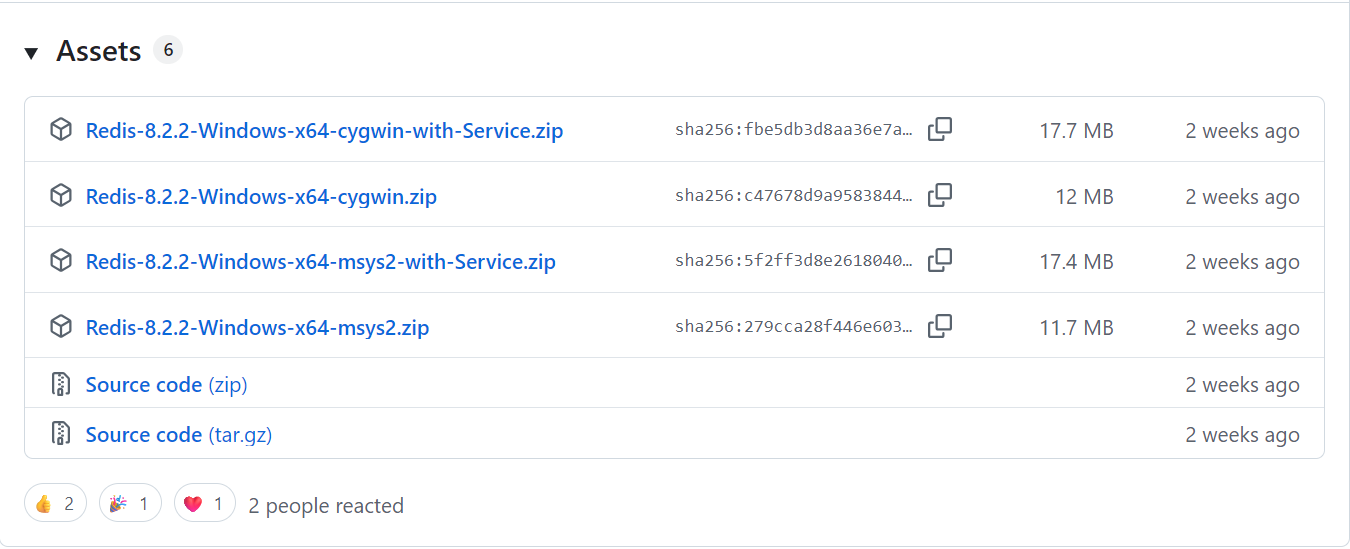
**Bước 1**: Truy cập vào địa chỉ: <https://github.com/redis-windows/redis-windows/releases> và tải xuống tệp cài đặt phù hợp. Tại đây, Redis phiên bản **8.2.2** được cung cấp dưới dạng các bản dựng (build) dành cho môi trường Windows, với hai tùy chọn chính:

* **Cygwin build:** thích hợp cho người dùng cần môi trường tương thích với Unix, tuy nhiên dung lượng lớn hơn và yêu cầu nhiều tài nguyên hơn.
* **MSYS2 build:** gọn nhẹ, hoạt động tối ưu trong môi trường Windows thông thường.

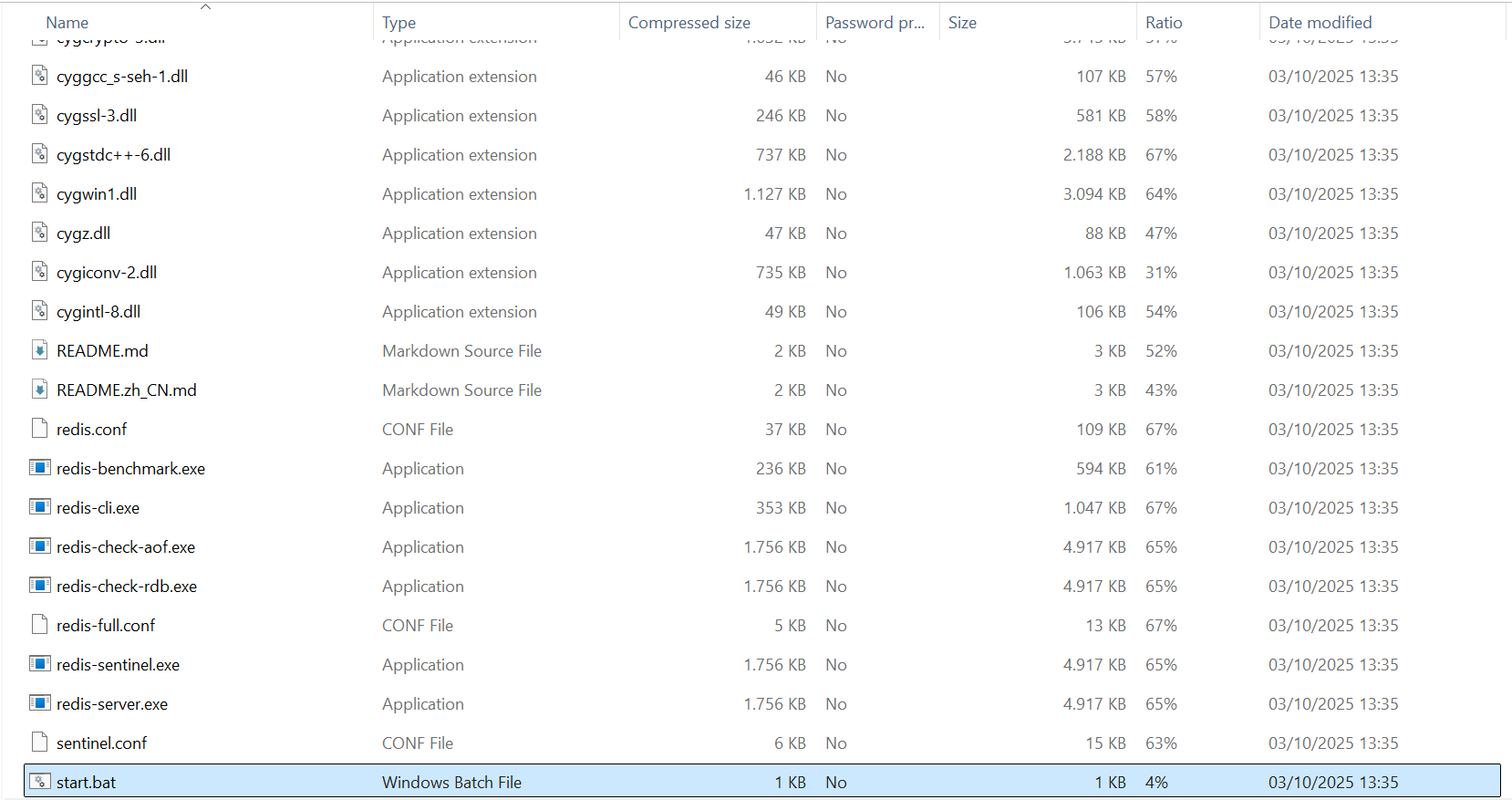
Mỗi tùy chọn đều có hai biến thể:

* **Bản “with-Service”** (ví dụ: Redis-8.2.2-Windows-x64-cygwin-with-Service.zip, dung lượng khoảng 17.7 MB), cho phép Redis tự động khởi động và hoạt động như một **dịch vụ hệ thống**.
* **Bản không “with-Service”** (ví dụ: Redis-8.2.2-Windows-x64-cygwin.zip, khoảng 12 MB), thích hợp khi người dùng muốn **chạy Redis thủ công**.

Đối với hầu hết người dùng Windows, **bản MSYS2** là lựa chọn khuyến nghị. Trong khi đó, **bản Cygwin** phù hợp hơn với các nhà phát triển cần môi trường gần với hệ thống Unix. Ngoài ra, **mã nguồn Redis** (định dạng .zip hoặc .tar.gz) cũng được cung cấp để người dùng tùy chỉnh hoặc biên dịch theo nhu cầu riêng.



Hình .: Tài nguyên tải về



Hình .: Các tệp trong thư mục nén tải về

Dự án Redis dành cho Windows cung cấp ba phương thức chính để khởi động và vận hành hệ thống Redis, phù hợp với các mục đích và mức độ sử dụng khác nhau. Cụ thể như sau:

1. Khởi chạy trực tiếp bằng tệp thực thi (*start.bat*)

Đây là phương thức khởi động nhanh và thuận tiện nhất. Người dùng chỉ cần nhấp đúp vào tệp start.bat trong thư mục cài đặt để Redis được khởi động ngay lập tức mà không cần thao tác qua dòng lệnh.

Phương pháp này đặc biệt phù hợp cho môi trường thử nghiệm hoặc mục đích học tập.

1. Khởi chạy Redis thông qua dòng lệnh

Redis có thể được khởi động trực tiếp bằng *Command Prompt* hoặc *PowerShell* với tệp cấu hình redis.conf như sau:

* **Command Prompt (CMD):** redis-server.exe redis.conf
* **PowerShell:** ./redis-server.exe redis.conf

Phương pháp này cho phép người dùng dễ dàng tùy chỉnh các tham số cấu hình trong tệp redis.conf, phù hợp cho các môi trường phát triển cần kiểm soát chi tiết.

1. Cài đặt và vận hành Redis như một dịch vụ hệ thống Windows

Phương pháp này giúp Redis có thể tự động khởi động cùng hệ thống và hoạt động ổn định ở chế độ nền (background service).

Quá trình cài đặt và quản lý dịch vụ Redis được thực hiện qua các lệnh sau:

* **Cài đặt dịch vụ:**

|  |
| --- |
| sc.exe create Redis binpath=C:\Software\Redis\RedisService.exe start= auto |

* **Khởi động dịch vụ:**

|  |
| --- |
| net start Redis |

* **Dừng dịch vụ:**

|  |
| --- |
| net stop Redis |

* **Gỡ bỏ dịch vụ khỏi hệ thống:**

|  |
| --- |
| sc.exe delete Redis |

Việc chạy Redis như một dịch vụ hệ thống đặc biệt hữu ích trong các môi trường sản xuất hoặc hệ thống cần đảm bảo tính sẵn sàng cao, giúp Redis tự động khởi động khi máy chủ khởi động lại.

**Bước 2:** Kết nối với Redis

Mở **Terminal/Command Prompt**.

Gõ lệnh:

* **redis-cli:** mở Redis Command Line Interface.
* **ping***:* kiểm tra kết nối giữa client và server Redis.

Nếu kết quả trả về là **PONG**, nghĩa là Redis đã được cài đặt thành công và đang hoạt động bình thường trên máy.

A screenshot of a computer

AI-generated content may be incorrect.

Hình .: Chạy lệnh ping kiểm tra kết nối

### Cài đặt Redis thông qua WSL (Windows Subsystem for Linux)

Ngoài việc cài trực tiếp trên Windows, bạn cũng có thể chạy Redis thông qua môi trường Linux trên Windows với WSL. Cách thực hiện như sau:

Mở PowerShell (với quyền Administrator) và nhập “*Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux*”.

A screenshot of a computer

AI-generated content may be incorrect.

Hình .: Bật tính năng WSL trên Windows

Sau khi WSL được kích hoạt, truy cập **Microsoft Store**, tìm kiếm và cài đặt bản phân phối **Ubuntu**.

A screenshot of a computer

AI-generated content may be incorrect.

Hình .: Cài đặt Ubuntu từ Microsoft Store

Mở ứng dụng **Ubuntu** và tiến hành cài đặt Redis theo sự hướng dẫn của trang [Install Redis on Windows | Docs](https://redis.io/docs/latest/operate/oss_and_stack/install/archive/install-redis/install-redis-on-windows/).

A screenshot of a computer

AI-generated content may be incorrect.

Hình .: Cài đặt

A screenshot of a computer screen

AI-generated content may be incorrect.

Hình .: Tải redis

Kết nối với Redis.

A screenshot of a computer

AI-generated content may be incorrect.

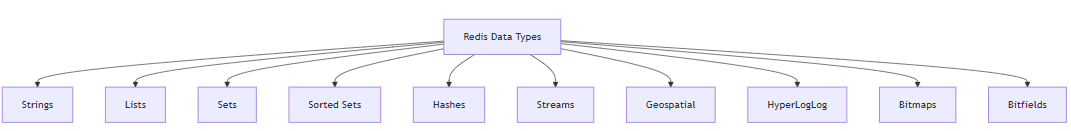
Hình .: Chạy lệnh ping kiểm tra

### Các công cụ GUI tiêu biểu cho Redis

Redis cung cấp nhiều công cụ giao diện trực quan, giúp người dùng dễ dàng quản lý và tương tác với dữ liệu. Dưới đây là bốn công cụ phổ biến nhất: RedisInsight, Redis Desktop Manager (RDM), P3X Redis UI và Medis.

* **RedisInsight**: Được phát triển bởi Redis Labs, RedisInsight nổi bật với giao diện thân thiện, hỗ trợ trực quan hóa dữ liệu, phân tích Slowlog và giám sát hiệu suất. Đây là lựa chọn toàn diện cho các nhà phát triển, mặc dù một số tính năng nâng cao yêu cầu đăng ký trả phí.
* **Redis Desktop Manager (RDM)**: Là một trong những công cụ được ưa chuộng nhất, RDM hỗ trợ hiển thị cấu trúc dữ liệu rõ ràng, bảo mật kết nối qua SSL/TLS và SSH tunneling, đồng thời tích hợp giao diện dòng lệnh (CLI) để thực thi lệnh. Với sự ổn định và dễ sử dụng, RDM được cộng đồng lập trình viên đánh giá cao.
* **P3X Redis UI**: Là công cụ mã nguồn mở dựa trên Electron, P3X Redis UI hỗ trợ quản lý Redis Cluster và Sentinel. Công cụ này phù hợp cho người dùng cần sự linh hoạt, có thể hoạt động cả trực tuyến và ngoại tuyến, nhưng giao diện có thể hơi phức tạp với người mới bắt đầu.
* **Medis**: Tập trung vào sự đơn giản và gọn nhẹ, Medis cho phép người dùng dễ dàng xem và chỉnh sửa dữ liệu. Với thiết kế tối giản, công cụ này lý tưởng cho những ai cần thao tác nhanh chóng, mặc dù không có nhiều tính năng nâng cao.

## Cấu trúc dữ liệu trong Redis



Hình .: Cấu trúc dữ liệu

### Strings

#### Giới thiệu

String là loại dữ liệu cơ bản nhất trong Redis, dùng để lưu trữ chuỗi văn bản, số, hoặc dữ liệu nhị phân (tối đa 512MB). String phù hợp cho các giá trị key-value đơn giản như cấu hình, bộ đếm, hoặc dữ liệu cache.

#### Lệnh cơ bản

Bảng .: Lệnh cơ bản với Strings

|  |  |  |
| --- | --- | --- |
| Lệnh | Cú pháp | Mô tả |
| SET | SET key value [options] | Thiết lập giá trị cho key. |
| GET | GET key | Lấy giá trị của key. |
| DEL | DEL key [key ...] | Xóa một hoặc nhiều key. |
| INCR | INCR key | Tăng giá trị số nguyên lên 1 (khởi tạo 0 nếu chưa tồn tại). |
| DECR | DECR key | Giảm giá trị số nguyên xuống 1. |
| APPEND | APPEND key value | Thêm chuỗi vào giá trị hiện tại. |
| STRLEN | STRLEN key | Trả về độ dài chuỗi. |

#### Ví dụ minh họa

**Thiết lập và thao tác với String**

|  |
| --- |
| SET username “VuDang” *# Tạo key 'username' với giá trị “VuDang*”  GET username *# Lấy giá trị của key*  STRLEN username *# Trả về độ dài chuỗi -> 6*  GETRANGE username 0 1 *# Lấy ký tự từ vị trí [0,1]*  SETRANGE username 7 “Redis!” *# Thay thế từ vị trí thứ 7*  APPEND username “- HUIT!” *# Nối thêm vào cuối chuỗi* |

**Set/Get nhiều key cùng lúc**

|  |
| --- |
| MSET first\_name “Vu” last\_name “Dang” age 21 *# Đặt nhiều key cùng lúc*  MGET first\_name last\_name age *# Lấy nhiều key cùng lúc* |

**Key có thời gian sống (TTL)**

|  |
| --- |
| SET session:user123 “active” EX 3600 *# Tạo key tồn tại 3600 giây (1h)*  SETEX temp\_token 30 “a1b2c3d4” *# Viết gọn, key sống 30 giây*  TTL session:user123 *# Dùng TTL kiểm tra* |

**Set với điều kiện (NX, XX)**

|  |
| --- |
| SET counter 10 NX *# Chỉ set nếu key chưa tồn tại*  SET counter 20 XX *# Chỉ set nếu key đã tồn tại* |

**Tăng giảm giá trị số**

|  |
| --- |
| SET counter 10  INCR counter *# +1 => 11*  GET counter *# -> “11”*  INCRBY counter 5 *# +5 => 16*  GET counter *# -> “16”* |

#### Ứng dụng thực tế

* Caching dữ liệu web: Lưu trữ kết quả truy vấn cơ sở dữ liệu (như danh sách sản phẩm) để giảm tải server chính. Ví dụ, một trang thương mại điện tử có thể lưu trữ nội dung trang chủ trong Redis để trả về nhanh chóng cho người dùng.
* Bộ đếm thời gian thực: Theo dõi số lượt xem bài viết, lượt thích, hoặc số lượng người dùng trực tuyến trên một website. Ví dụ, sử dụng INCR để đếm số lượt truy cập vào một bài blog.
* Cấu hình ứng dụng: Lưu trữ các giá trị cấu hình động, như URL API hoặc thông số hệ thống, để dễ dàng cập nhật mà không cần khởi động lại ứng dụng.
* Lưu trữ token phiên: Lưu trữ các token xác thực tạm thời (như JWT) với thời gian sống (TTL) để quản lý phiên đăng nhập người dùng.

### Lists

#### Giới thiệu

List là danh sách có thứ tự, cho phép thêm/xóa phần tử ở đầu hoặc cuối, hoạt động như hàng đợi (queue) hoặc ngăn xếp (stack).

#### Lệnh cơ bản

Bảng .: Lệnh cơ bản với List

|  |  |  |
| --- | --- | --- |
| Lệnh | Cú pháp | Mô tả |
| LPUSH | LPUSH key element [...] | Thêm phần tử vào đầu danh sách. |
| RPUSH | RPUSH key element [...] | Thêm phần tử vào cuối danh sách. |
| LPOP | LPOP key | Xóa và trả phần tử đầu. |
| RPOP | RPOP key | Xóa và trả phần tử cuối. |
| DEL | DEL key | Xóa danh sách nếu rỗng sau thao tác. |
| LRANGE | LRANGE key start stop | Trả về phạm vi phần tử (0 - 1 cho tất cả). |
| LLEN | LLEN key | Trả về độ dài danh sách. |

#### Ví dụ minh họa

|  |
| --- |
| # Thêm phần tử vào danh sách  LPUSH mylist “world” *# -> (integer) 1*  LPUSH mylist “hello” *# -> (integer) 2*  RPUSH mylist “Redis” *# -> (integer) 3*  LRANGE mylist 0 -1 *# Xem toàn bộ danh sách*  LINDEX mylist 0 *# Lấy phần tử đầu tiên (index = 0)*  LINDEX mylist -1 *# Lấy phần tử cuối cùng (index = -1)*  LPOP mylist *# Xóa và lấy phần tử đầu tiên*  RPOP mylist *# Xóa và lấy phần tử cuối cùng*  LRANGE mylist 0 -1 *# Xem lại danh sách còn gì* |

#### Ứng dụng thực tế

* Hàng đợi tác vụ bất đồng bộ: Quản lý các tác vụ như gửi email hàng loạt, xử lý thanh toán, hoặc tạo báo cáo. Ví dụ, một hệ thống gửi thông báo đẩy có thể sử dụng LPUSH để thêm thông báo và RPOP để xử lý.
* Lịch sử hành động người dùng: Lưu trữ danh sách các hành động gần đây, như lịch sử tìm kiếm hoặc sản phẩm đã xem trên website thương mại điện tử.
* Hàng đợi tin nhắn thời gian thực: Xử lý tin nhắn trong các ứng dụng chat, nơi tin nhắn được thêm vào danh sách bằng RPUSH và đọc bằng LRANGE.
* Hàng đợi xử lý sự kiện: Quản lý luồng sự kiện trong hệ thống IoT, như lưu trữ các tín hiệu từ thiết bị thông minh để xử lý theo thứ tự.

### Sets

#### Giới thiệu

Set là tập hợp không thứ tự, không cho phép trùng lặp, phù hợp để lưu trữ danh sách các mục duy nhất.

#### Lệnh cơ bản

Bảng .: Lệnh cơ bản với Set

|  |  |  |
| --- | --- | --- |
| Lệnh | Cú pháp | Mô tả |
| SADD | SADD key member [...] | Thêm một hoặc nhiều thành viên. |
| SMEMBERS | SMEMBERS key | Trả về tất cả thành viên. |
| SREM | SREM key member [...] | Xóa một hoặc nhiều thành viên. |
| DEL | DEL key | Xóa set nếu rỗng sau thao tác. |
| SCARD | SCARD key | Trả về số lượng thành viên. |
| SISMEMBER | SISMEMBER key member | Kiểm tra thành viên có tồn tại. |

#### Ví dụ minh họa

|  |
| --- |
| *# Thêm phần tử vào Set (chỉ giữ unique, không trùng)*  SADD myset “apple” “banana” “orange” “apple”  SMEMBERS myset *# Xem tất cả phần tử trong Set*  SISMEMBER myset “apple” *# Kiểm tra xem phần tử có trong Set không*  SISMEMBER myset “grape”  SREM myset “orange” *# Xóa phần tử*  SMEMBERS myset  SCARD myset *# Đếm số phần tử*  *# Thao tác Set nâng cao*  SADD set1 “a” “b” “c”  SADD set2 “b” “c” “d”  *# Giao nhau*  SINTER set1 set2  *# Hợp*  SUNION set1 set2  *# Khác biệt*  SDIFF set1 set2  SDIFF set2 set1 |

#### Ứng dụng thực tế

* Danh sách duy nhất: Lưu trữ danh sách người dùng trực tuyến, thẻ tag bài viết, hoặc danh mục sản phẩm để đảm bảo không trùng lặp. Ví dụ, một blog có thể dùng Set để lưu trữ các tag như “công nghệ”, “khoa học”.
* Xử lý giao dịch tập hợp: Sử dụng SINTER, SUNION, hoặc SDIFF để tìm bạn chung giữa hai người dùng trên mạng xã hội hoặc sản phẩm chung trong giỏ hàng của nhiều khách hàng.
* Kiểm tra thành viên nhanh: Xác minh xem một người dùng có trong danh sách đen (blacklist) hay không, như trong hệ thống kiểm duyệt nội dung.
* Quản lý quyền truy cập: Lưu trữ danh sách các vai trò hoặc quyền của người dùng, ví dụ, danh sách người dùng có quyền admin trong một ứng dụng.

### Hashes

#### Giới thiệu

Hash là cấu trúc dữ liệu bản đồ, lưu trữ các cặp trường-giá trị, phù hợp để biểu diễn các đối tượng như thông tin người dùng.

#### Lệnh cơ bản

Bảng .: Lệnh cơ bản với Hash

|  |  |  |
| --- | --- | --- |
| Lệnh | Cú pháp | Mô tả |
| HSET | HSET key field value [...] | Thiết lập giá trị cho trường. |
| HGET | HGET key field | Lấy giá trị của trường. |
| HDEL | HDEL key field [...] | Xóa một hoặc nhiều trường. |
| DEL | DEL key | Xóa hash nếu không còn trường. |
| HGETALL | HGETALL key | Trả về tất cả trường và giá trị. |
| HLEN | HLEN key | Trả về số lượng trường. |

#### Ví dụ minh họa

|  |
| --- |
| *# Tạo một user với nhiều thuộc tính*  HSET user:1 username “vu” email “abc@gmail.com” age “21”  HGET user:1 username *# Lấy một trường cụ thể*  HMGET user:1 username email *# Lấy nhiều trường*  HGETALL user:1 *# Lấy toàn bộ fields và values*  HSET user:1 age “20” *# Cập nhật một trường*  HEXISTS user:1 email *# Kiểm tra field có tồn tại không*  HDEL user:1 age *# Xóa một trường*  HLEN user:1 *# Đếm số field còn lại* |

#### Ứng dụng thực tế

* Lưu trữ thông tin đối tượng: Biểu diễn thông tin người dùng (tên, tuổi, email) hoặc sản phẩm (tên, giá, mô tả) trong các ứng dụng như mạng xã hội hoặc thương mại điện tử.
* Cấu hình phức tạp: Lưu trữ các cấu hình có nhiều thuộc tính, như cài đặt giao diện người dùng hoặc thông số hệ thống, cho phép cập nhật từng trường riêng lẻ.
* Quản lý phiên người dùng: Lưu trữ dữ liệu phiên (session) như thông tin giỏ hàng hoặc trạng thái đăng nhập, với khả năng truy xuất nhanh các trường cụ thể.
* Lưu trữ dữ liệu cấu trúc: Dùng trong các hệ thống phân tích để lưu trữ các thuộc tính của sự kiện, như thông tin giao dịch (mã giao dịch, thời gian, giá trị).

### Sorted Sets

#### Giới thiệu

Sorted Set là tập hợp có thứ tự, mỗi thành viên có điểm số để sắp xếp, phù hợp cho bảng xếp hạng hoặc dữ liệu ưu tiên.

#### Lệnh cơ bản

Bảng .: Lệnh cơ bản với Sorted Set

|  |  |  |
| --- | --- | --- |
| Lệnh | Cú pháp | Mô tả |
| ZADD | ZADD key score member [...] | Thêm thành viên với điểm số. |
| ZRANGE | ZRANGE key start stop | Trả về phạm vi thành viên theo thứ tự. |
| ZREM | ZREM key member [...] | Xóa thành viên. |
| DEL | DEL key | Xóa sorted set. |
| ZCARD | ZCARD key | Trả về số lượng thành viên. |

#### Ví dụ minh họa

|  |
| --- |
| *# Thêm 3 người chơi với điểm*  ZADD leaderboard 100 “Alice”  ZADD leaderboard 200 “Bob”  ZADD leaderboard 150 “Charlie”  *# Xem toàn bộ bảng xếp hạng theo thứ tự tăng*  ZRANGE leaderboard 0 -1 WITHSCORES  *# Xem top 2 theo thứ tự giảm (người điểm cao nhất đứng đầu)*  ZREVRANGE leaderboard 0 1 WITHSCORES  *# Lấy hạng của Alice (bắt đầu từ 0)*  ZRANK leaderboard “Alice”  *# Lấy điểm của Bob*  ZSCORE leaderboard “Bob” |

#### Ứng dụng thực tế

* Bảng xếp hạng thời gian thực: Quản lý bảng xếp hạng người chơi trong trò chơi trực tuyến hoặc bài đăng phổ biến trên mạng xã hội dựa trên số lượt thích. Ví dụ, một ứng dụng game có thể dùng Sorted Set để xếp hạng người chơi theo điểm số.
* Hàng đợi ưu tiên: Quản lý các tác vụ theo mức độ ưu tiên, như xử lý đơn hàng VIP trước trong hệ thống thương mại điện tử.
* Tìm kiếm theo phạm vi: Lấy các mục trong khoảng điểm số, như tìm kiếm các bài viết có số lượt xem từ 100 đến 1000.
* Lập lịch tác vụ: Sắp xếp các tác vụ theo thời gian thực thi, ví dụ, trong hệ thống gửi thông báo định kỳ, với điểm số là timestamp.

### Bitmap

#### Giới thiệu

Bitmap là một chuỗi bit (0 hoặc 1) được sử dụng để lưu trữ và thao tác dữ liệu nhị phân một cách hiệu quả. Mỗi bit trong Bitmap đại diện cho một trạng thái, giúp tiết kiệm bộ nhớ khi xử lý tập hợp lớn các giá trị boolean

#### Lệnh cơ bản

Bảng .: Lệnh cơ bản với Bitmap

|  |  |  |
| --- | --- | --- |
| Chỉ thị | Cú pháp / Giá trị mặc định | Mô tả |
| SETBIT | SETBIT key offset value | Đặt giá trị bit (0 hoặc 1) tại vị trí offset. |
| GETBIT | GETBIT key offset | Lấy giá trị bit tại vị trí offset. |
| BITCOUNT | BITCOUNT key [start end] | Đếm số bit 1 trong phạm vi (hoặc toàn bộ). |
| BITOP | BITOP operation destkey key [key ...] | Thực hiện phép toán bit (AND, OR, XOR, NOT) trên các key. |

#### Ví dụ minh họa

|  |
| --- |
| # Đặt bit tại vị trí 100 là 1 (đánh dấu người dùng đã xem)  SETBIT user:123:views 100 1  # Kiểm tra giá trị bit tại vị trí 100  GETBIT user:123:views 100  # Đặt thêm bit tại vị trí 101 là 1  SETBIT user:123:views 101 1  # Đếm tổng số bit 1 trong phạm vi 0-200  BITCOUNT user:123:views 0 200  # Thực hiện phép AND với một Bitmap khác  BITOP AND result user:123:views user:124:views  # Đặt bit tại vị trí 102 là 0 (đánh dấu không hoạt động)  SETBIT user:123:views 102 0  # Đếm lại số bit 1 sau thay đổi  BITCOUNT user:123:views  # Thực hiện phép OR với Bitmap khác  BITOP OR combined user:123:views user:125:views |

#### Ứng dụng thực tế

* Theo dõi trạng thái người dùng: Đánh dấu người dùng đã đăng nhập hoặc xem bài viết.
* Phân tích hành vi: Đếm ngày hoạt động liên tục của người dùng.
* Lọc dữ liệu: Sử dụng để lọc danh sách dựa trên trạng thái nhị phân.
* Quản lý quyền truy cập: Lưu trữ trạng thái quyền hạn của người dùng.

### HyperLogLog

#### Giới thiệu

HyperLogLog là một cấu trúc dữ liệu xác suất, được thiết kế để ước lượng số lượng phần tử duy nhất (cardinality) trong một tập hợp lớn với độ chính xác cao và sử dụng bộ nhớ rất nhỏ (thường chỉ khoảng 12KB).

#### Lệnh cơ bản

Bảng .: Lệnh cơ bản với HyperLogLog

|  |  |  |
| --- | --- | --- |
| Chỉ thị | Cú pháp / Giá trị mặc định | Mô tả |
| PFADD | PFADD key element [element ...] | Thêm phần tử vào HyperLogLog. |
| PFCOUNT | PFCOUNT key [key ...] | Ước lượng số lượng phần tử duy nhất. |
| PFMERGE | PFMERGE destkey sourcekey [sourcekey ...] | Gộp nhiều HyperLogLog thành một. |

#### Ví dụ minh họa

|  |
| --- |
| # Thêm 3 lượt truy cập duy nhất  PFADD page:visits user1 user2 user3  # Ước lượng số lượng truy cập duy nhất  PFCOUNT page:visits  # Thêm thêm 2 lượt truy cập mới  PFADD page:visits user4 user5  # Gộp dữ liệu từ hai trang thăm  PFADD page:visits2 user6 user7 user8  PFMERGE total page:visits page:visits2  # Kiểm tra tổng số duy nhất sau khi gộp  PFCOUNT total  # Thêm thêm một lượt truy cập vào trang thứ nhất  PFADD page:visits user9  # Kiểm tra lại số duy nhất trên page:visits  PFCOUNT page:visits |

#### Ứng dụng thực tế

* Đếm người dùng truy cập duy nhất: Theo dõi số lượng người truy cập website.
* Phân tích dữ liệu lớn: Ước lượng số IP hoặc sự kiện không trùng lặp.
* Thống kê thời gian thực: Đếm số lượt xem video duy nhất.
* Kết hợp tập hợp: Gộp dữ liệu từ nhiều nguồn để phân tích tổng hợp.

### Streams

#### Giới thiệu

Streams là một kiểu dữ liệu dành cho việc lưu trữ và xử lý dữ liệu dạng luồng theo thứ tự thời gian, hoạt động như một nhật ký (log) có khả năng xử lý dữ liệu thời gian thực. Streams hỗ trợ thêm dữ liệu, đọc theo phạm vi, và quản lý nhóm người tiêu dùng (consumer groups) để xử lý phân tán.

#### Lệnh cơ bản

Bảng .: Lệnh cơ bản với Stream

|  |  |  |
| --- | --- | --- |
| Chỉ thị | Cú pháp / Giá trị mặc định | Mô tả |
| XADD | XADD key \* field value [field value ...] | Thêm entry vào stream. |
| XRANGE | XRANGE key start end | Trả về phạm vi entry theo ID. |
| XREAD | XREAD STREAMS key [key ...] ID [ID ...] | Đọc entry từ stream (có thể block). |
| XDEL | XDEL key ID [ID ...] | Xóa entry theo ID. |

#### Ví dụ minh họa

|  |
| --- |
| *# Thêm một sự kiện nhiệt độ từ cảm biến*  *XADD sensor:stream \* sensor\_id 001 temp 25.5*  *# Thêm thêm một sự kiện khác*  *XADD sensor:stream \* sensor\_id 002 temp 26.0*  *# Đọc tất cả các sự kiện từ đầu stream*  *XRANGE sensor:stream - +*  *# Đọc sự kiện mới nhất (chờ nếu không có)*  *XREAD STREAMS sensor:stream $*  *# Xóa sự kiện đầu tiên*  *XDEL sensor:stream 123456789*  *# Thêm sự kiện với nhiều trường*  *XADD sensor:stream \* sensor\_id 003 temp 27.0 humidity 60*  *# Đọc các sự kiện từ ID cụ thể đến cuối*  *XRANGE sensor:stream 123456789 +* |

#### Ứng dụng thực tế

* Quản lý log hệ thống: Lưu trữ log hoạt động server.
* Hàng đợi tin nhắn thời gian thực: Xử lý tin nhắn trong ứng dụng chat.
* Ghi nhận sự kiện IoT: Lưu trữ dữ liệu cảm biến từ thiết bị thông minh.
* Phân tích luồng dữ liệu: Theo dõi sự kiện liên tục như giao dịch tài chính.

### Geo

#### Giới thiệu

Geo cho phép lưu trữ và truy vấn dữ liệu địa lý, như tọa độ (kinh độ, vĩ độ), và hỗ trợ các phép tính như tìm khoảng cách giữa hai điểm hoặc tìm các điểm trong một bán kính cụ thể.

#### Lệnh cơ bản

Bảng .: Lệnh cơ bản với Geo

|  |  |  |
| --- | --- | --- |
| Chỉ thị | Cú pháp / Giá trị mặc định | Mô tả |
| GEOADD | GEOADD key longitude latitude member | Thêm tọa độ địa lý. |
| GEODIST | GEODIST key member1 member2 [unit] | Tính khoảng cách giữa hai điểm. |
| GEORADIUS | GEORADIUS key longitude latitude radius unit | Tìm điểm trong bán kính. |

#### Ví dụ minh họa

|  |
| --- |
| *# Thêm tọa độ của hai cửa hàng*  *GEOADD stores 106.66 10.76 “StoreA” 106.70 10.80 “StoreB”*  *# Thêm tọa độ của cửa hàng thứ ba*  *GEOADD stores 106.65 10.75 “StoreC”*  *# Tính khoảng cách giữa StoreA và StoreB (đơn vị km)*  *GEODIST stores “StoreA” “StoreB” km*  *# Tìm cửa hàng trong bán kính 5km quanh tọa độ 106.66 10.76*  *GEORADIUS stores 106.66 10.76 5 km*  *# Thêm tọa độ của một nhà hàng*  *GEOADD stores 106.68 10.78 “RestaurantX”*  *# Tính khoảng cách từ StoreA đến RestaurantX*  *GEODIST stores “StoreA” “RestaurantX” km* |

#### Ứng dụng thực tế

* Tìm cửa hàng gần nhất: Gợi ý cửa hàng dựa trên vị trí người dùng.
* Theo dõi phương tiện: Theo dõi vị trí xe buýt hoặc xe giao hàng.
* Kết nối xã hội: Tìm bạn bè trong khu vực dựa trên tọa độ.
* Ứng dụng du lịch: Tìm điểm tham quan trong bán kính nhất định.

### JSON

#### Giới thiệu

RedisJSON (thông qua mô-đun RedisJSON) cho phép lưu trữ, truy vấn và thao tác trực tiếp trên dữ liệu dạng JSON, giúp quản lý các tài liệu phức tạp mà không cần chuyển đổi định dạng.

#### Lệnh cơ bản

Bảng .: Lệnh cơ bản với JSON

|  |  |  |
| --- | --- | --- |
| Chỉ thị | Cú pháp / Giá trị mặc định | Mô tả |
| JSON.SET | JSON.SET key path value | Lưu trữ giá trị JSON tại đường dẫn. |
| JSON.GET | JSON.GET key [path] | Lấy giá trị JSON tại đường dẫn. |
| JSON.DEL | JSON.DEL key path | Xóa phần tử JSON tại đường dẫn. |

#### Ví dụ minh họa

|  |
| --- |
| *# Lưu thông tin người dùng dưới dạng JSON*  *JSON.SET user:123 . '{“name”: “John”, “age”: 30, “city”: “New York”}'*  *# Lấy tên của người dùng*  *JSON.GET user:123 .name*  *# Cập nhật tuổi thành 31*  *JSON.SET user:123 .age 31*  *# Thêm trường email*  *JSON.SET user:123 .email* [*john@example.com*](mailto:john@example.com)  *# Xóa trường city*  *JSON.DEL user:123 .city*  *# Thêm danh sách sở thích*  *JSON.SET user:123 .hobbies “[\”reading\”, \”gaming\”]”*  *# Lấy toàn bộ dữ liệu người dùng*  *JSON.GET user:123 .* |

#### Ứng dụng thực tế

* Lưu trữ thông tin người dùng: Quản lý dữ liệu người dùng phức tạp.
* Cấu hình ứng dụng: Lưu trữ cấu hình động dạng JSON.
* Dữ liệu sản phẩm: Lưu thông tin sản phẩm trong thương mại điện tử.
* API thời gian thực: Truy xuất nhanh dữ liệu JSON trong ứng dụng web.

### TimeSeries

#### Giới thiệu

RedisTimeSeries (thông qua mô-đun RedisTimeSeries) được tối ưu hóa để lưu trữ và phân tích dữ liệu chuỗi thời gian, như số liệu cảm biến, giá cổ phiếu, hoặc thống kê hiệu suất hệ thống. Nó hỗ trợ nén dữ liệu và truy vấn theo thời gian.

#### Lệnh cơ bản

Bảng .: Lệnh cơ bản với TimeSeries

|  |  |  |
| --- | --- | --- |
| Chỉ thị | Cú pháp / Giá trị mặc định | Mô tả |
| TS.ADD | TS.ADD key timestamp value | Thêm giá trị tại timestamp. |
| TS.RANGE | TS.RANGE key from to | Truy vấn dữ liệu trong khoảng thời gian. |
| TS.DEL | TS.DEL key from to | Xóa dữ liệu trong khoảng thời gian. |

#### Ví dụ minh họa

|  |
| --- |
| *# Thêm giá trị nhiệt độ tại thời điểm 1697059200 (Unix timestamp)*  *TS.ADD sensor:temp 1697059200 25.5*  *# Thêm thêm giá trị tại thời điểm khác*  *TS.ADD sensor:temp 1697062800 26.0*  *# Lấy dữ liệu từ đầu đến thời điểm 1697062800*  *TS.RANGE sensor:temp 0 1697062800*  *# Thêm giá trị mới tại thời điểm hiện tại (09:33 AM +07, 14/09/2025, Unix: 1726287180)*  *TS.ADD sensor:temp 1726287180 26.5*  *# Xóa dữ liệu từ 1697059200 đến 1697062800*  *TS.DEL sensor:temp 1697059200 1697062800*  *# Thêm giá trị nhiệt độ khác tại thời điểm hiện tại*  *TS.ADD sensor:temp 1726287200 26.7*  *# Lấy dữ liệu trong khoảng 24 giờ gần nhất*  *TS.RANGE sensor:temp -1726200780 1726287180* |

#### Ứng dụng thực tế

* Theo dõi IoT: Lưu trữ dữ liệu nhiệt độ hoặc độ ẩm từ cảm biến.
* Phân tích tài chính: Ghi nhận giá cổ phiếu theo thời gian.
* Giám sát server: Theo dõi CPU hoặc bộ nhớ theo thời gian thực.
* Dự báo thời tiết: Lưu trữ và phân tích dữ liệu khí hậu.

## Truy vấn Full-Text Search (FT) trong Redis

### Giới thiệu

Truy vấn Full-Text Search (FT) trong Redis, được cung cấp qua module RediSearch, cho phép thực hiện tìm kiếm văn bản toàn diện trên dữ liệu được lưu trữ dưới dạng hash, JSON hoặc text.

Tương tự như aggregation trong MongoDB (sử dụng $match, $project, $sort để lọc, biến đổi và sắp xếp dữ liệu), FT kết hợp tìm kiếm văn bản với các bộ lọc, sắp xếp và phân trang để xử lý dữ liệu lớn một cách hiệu quả. Nó hỗ trợ tìm kiếm fuzzy, prefix, wildcard, và các toán tử logic, giúp Redis trở thành công cụ mạnh mẽ cho tìm kiếm thời gian thực mà không cần engine bên ngoài như Elasticsearch. Để sử dụng FT, cần cài đặt module RediSearch trên Redis server.

### Lệnh cơ bản

Để sử dụng FT, trước tiên cần tạo index bằng lệnh FT.CREATE để định nghĩa schema cho dữ liệu. Sau đó, sử dụng FT.SEARCH để thực hiện truy vấn, tương tự như aggregation pipeline trong MongoDB với các stage lọc và sắp xếp.

Bảng .: Lệnh cơ bản khi truy vấn Full-Text Search (FT)

|  |  |  |
| --- | --- | --- |
| Chỉ thị | Cú pháp / Giá trị mặc định | Mô tả |
| FT.CREATE | FT.CREATE index\_name ON {HASH|JSON|TAG} PREFIX {n} prefix SCHEMA {field type [options]} | Tạo index cho tìm kiếm, định nghĩa schema cho các trường (TEXT cho văn bản, NUMERIC cho số, TAG cho phân loại). |
| FT.SEARCH | FT.SEARCH index\_name query [options] | Thực hiện tìm kiếm văn bản với bộ lọc, sắp xếp và phân trang (tương tự $match + $sort + $limit trong MongoDB). |
| FT.EXPLAIN | FT.EXPLAIN index\_name query | Giải thích kế hoạch thực thi truy vấn, hiển thị cách Redis xử lý (tương tự explain trong MongoDB). |
| FT.DROPINDEX | FT.DROPINDEX index\_name [DD] | Xóa index (DD để xóa dữ liệu gốc). |

### Ứng dụng thực tế

* Tìm kiếm văn bản trong catalog sản phẩm: Tạo index cho trường title và description, sử dụng FT.SEARCH với query “@title:smartphone @price:[100 500]” để lọc sản phẩm theo từ khóa và giá, tương tự aggregation pipeline trong MongoDB với $match trên text và range.
* Phân tích log hoặc bài viết: Sử dụng FT.SEARCH với fuzzy search (“hello~”) để tìm kiếm gần đúng, kết hợp SORTBY để sắp xếp theo ngày, giống như $sort và $regex trong MongoDB aggregation.
* Tìm kiếm theo vị trí địa lý: Kết hợp với trường GEO trong index để query “@loc:[lon lat radius km]”, tương tự geospatial query trong MongoDB với $geoWithin.
* Tìm kiếm thời gian thực với phân trang: Sử dụng LIMIT và OFFSET trong FT.SEARCH để phân trang kết quả, kết hợp RETURN để chọn trường trả về, giống như $skip, $limit và $project trong aggregation.

### Cách sử dụng FT

Để sử dụng FT trong Redis một cách hiệu quả, làm theo các bước sau:

1. Cài đặt RediSearch:
   * Cài module RediSearch từ nguồn chính thức hoặc thông qua package manager như *redis-stack-server*.
   * Khởi động Redis với module: *redis-server --loadmodule* */path/to/redisearch.so*.
2. Tạo Index:
   * Xác định dữ liệu cần index (hash, JSON, hoặc tag) và tạo index với *FT.CREATE*. Ví dụ, index cho sách với các trường title (TEXT) và published\_at (NUMERIC).
   * Đảm bảo thêm prefix phù hợp để tránh xung đột key.
3. Thêm dữ liệu:
   * Sử dụng lệnh *HSET* (cho hash) hoặc *JSON.SET* (cho JSON) để thêm dữ liệu vào key có prefix khớp với index.
   * Dữ liệu sẽ tự động được index hóa khi thêm.
4. Thực hiện truy vấn:
   * Sử dụng FT.SEARCH với cú pháp query:
     + Tìm kiếm cơ bản: “*term*” (tìm từ “*term*” trong tất cả trường TEXT).
     + Bộ lọc:
       1. *@field:[min max]* (lọc số hoặc khoảng thời gian)
       2. *@field:{value}* (lọc tag).
     + Toán tử: *+term1 -term2 (AND, NOT), term1 term2 (OR)*.
     + Sắp xếp: *SORTBY field [ASC\|DESC]*.
     + Phân trang: *LIMIT offset count*.
   * Ví dụ: *FT.SEARCH idx:books “@title:lord @published\_at:[1900 2000]” SORTBY published\_at DESC LIMIT 0 5*.
5. Tối ưu hóa hiệu suất:
   * Chỉ index các trường cần thiết để giảm tải bộ nhớ.
   * Sử dụng *SORTABLE* cho các trường cần sắp xếp để tăng tốc độ query.
   * Kiểm tra kế hoạch thực thi với *FT.EXPLAIN* để tối ưu query.
6. Quản lý Index:
   * Cập nhật schema hoặc xóa index với *FT.DROPINDEX* khi cần (thêm [*DD*] để xóa dữ liệu gốc).
   * Theo dõi kích thước index qua *FT.INFO index\_name*.

### Ví dụ minh họa

|  |
| --- |
| # Tạo index cho dữ liệu sách (tương tự định nghĩa schema trong MongoDB)  FT.CREATE idx:books ON HASH PREFIX 1 book: SCHEMA title TEXT SORTABLE body TEXT published\_at NUMERIC SORTABLE  # Thêm dữ liệu sách (tương tự insert documents)  HSET book:1 title “Harry Potter and the Sorcerer's Stone” body “A wizard boy discovers his powers” published\_at 1997  HSET book:2 title “The Fellowship of the Ring” body “A fellowship embarks on a quest to destroy a ring” published\_at 1954  HSET book:3 title “1984” body “A dystopian novel about surveillance and totalitarianism” published\_at 1949  # Tìm kiếm sách chứa “wizard” (tương tự $match với text search)  FT.SEARCH idx:books “wizard” RETURN 3 title body published\_at  # Tìm kiếm với bộ lọc năm xuất bản > 1950 và sắp xếp theo năm giảm dần (tương tự $match + $sort + $limit)  FT.SEARCH idx:books “@published\_at:[1950 +inf] wizard” SORTBY published\_at DESC RETURN 2 title published\_at LIMIT 0 2  # Giải thích truy vấn (tương tự explain plan)  FT.EXPLAIN idx:books “@published\_at:[1950 +inf] wizard”  # Xóa index nếu cần (tương tự drop collection)  FT.DROPINDEX idx:books |

## Một số thao tác quản trị cơ bản

### Kiểm tra thông tin server

#### Giới thiệu

Redis cung cấp các lệnh để kiểm tra trạng thái và hiệu suất server, giúp quản trị viên giám sát hệ thống.

#### Lệnh cơ bản

Bảng .: Lệnh cơ bản để kiểm tra thông tin server

|  |  |  |  |
| --- | --- | --- | --- |
| Lệnh | Cú pháp | Mô tả | Ví dụ |
| INFO | INFO [section] | Hiển thị thông tin chi tiết về server (CPU, RAM, kết nối, …) | INFO SERVER |
| MONITOR | MONITOR | Theo dõi toàn bộ lệnh client gửi đến Redis theo thời gian thực | MONITOR |

#### Ứng dụng thực tế

* Giám sát hiệu suất hệ thống: Kiểm tra mức sử dụng bộ nhớ, CPU, số lượng kết nối, và thời gian hoạt động để tối ưu hóa tài nguyên. Ví dụ, một quản trị viên có thể dùng INFO để phát hiện tình trạng quá tải kết nối.
* Gỡ lỗi ứng dụng: Sử dụng MONITOR để theo dõi các lệnh được gửi từ ứng dụng, giúp phát hiện các truy vấn chậm hoặc sai logic.
* Phân tích hiệu suất: Dùng INFO để thu thập số liệu về số lượng lệnh được xử lý mỗi giây, từ đó đánh giá khả năng mở rộng của hệ thống.
* Kiểm tra trạng thái server: Đảm bảo server Redis đang hoạt động bình thường trước khi triển khai bản cập nhật hoặc bảo trì.

### Quản lý cơ sở dữ liệu

#### Giới thiệu

Redis hỗ trợ nhiều cơ sở dữ liệu logic (mặc định là 16, từ 0 đến 15), cho phép quản lý dữ liệu riêng biệt.

#### Lệnh cơ bản

Bảng .: Lệnh cơ bản quản trị cơ sở dữ liệu

|  |  |  |  |
| --- | --- | --- | --- |
| Lệnh | Cú pháp | Mô tả | Ví dụ |
| SELECT | SELECT index | Chuyển sang cơ sở dữ liệu logic theo index (0–15 mặc định) | SELECT 1 |
| DBSIZE | DBSIZE | Tra cứu số lượng key trong cơ sở dữ liệu. | DBSIZE |

#### Ứng dụng thực tế

* Phân tách dữ liệu theo ứng dụng: Sử dụng các cơ sở dữ liệu khác nhau cho các ứng dụng, như cơ sở dữ liệu 0 cho web, cơ sở dữ liệu 1 cho ứng dụng di động.
* Quản lý tài nguyên server: Dùng DBSIZE để kiểm tra số lượng key, từ đó đánh giá mức độ sử dụng và lên kế hoạch mở rộng bộ nhớ.
* Tối ưu hóa hiệu suất: Tách biệt dữ liệu của các khách hàng hoặc dự án để giảm xung đột truy cập và tăng tốc độ truy xuất.
* Kiểm tra trước khi sao lưu: Dùng DBSIZE để xác định khối lượng dữ liệu cần sao lưu hoặc di chuyển sang server khác.

### Xóa dữ liệu

#### Giới thiệu

Redis cung cấp các lệnh để xóa dữ liệu một cách linh hoạt, đảm bảo quản lý hiệu quả tài nguyên.

#### Lệnh cơ bản

Bảng .: Lệnh cơ bản dùng cho “Xóa dữ liệu”

|  |  |  |  |
| --- | --- | --- | --- |
| Lệnh | Cú pháp | Mô tả | Ví dụ |
| DEL | DEL key [key ...] | Xóa một hoặc nhiều key ngay lập tức (synchronous) | DEL mykey mylist |
| UNLINK | UNLINK key [key ...] | Xóa key theo cách không đồng bộ (asynchronous), tránh block server khi key lớn | UNLINK myhash |

#### Ứng dụng thực tế

* Dọn dẹp dữ liệu hết hạn: Xóa các key phiên người dùng đã hết hạn hoặc dữ liệu cache không còn cần thiết, như giỏ hàng tạm thời.
* Tối ưu hiệu suất server: Sử dụng UNLINK để xóa các key lớn (như danh sách hoặc hash lớn) mà không làm gián đoạn hoạt động server.
* Quản lý bộ nhớ: Xóa định kỳ các key không còn sử dụng để giải phóng bộ nhớ, đặc biệt trong các hệ thống có lưu lượng truy cập cao.
* Bảo trì hệ thống: Dùng DEL hoặc UNLINK để xóa dữ liệu thử nghiệm hoặc dữ liệu lỗi trong quá trình phát triển ứng dụng.

### Kiểm tra key và bộ nhớ

Một số câu lệnh dùng để kiểm tra key và bộ nhớ:

Bảng .: Lệnh cơ bản dùng cho kiểm tra key và bộ nhớ

|  |  |  |  |
| --- | --- | --- | --- |
| Lệnh | Cú pháp | Mô tả | Ví dụ |
| TYPE | TYPE key | Xác định kiểu dữ liệu của key | TYPE mystring → string |
| OBJECT encoding | OBJECT encoding key | Xem cách Redis lưu trữ dữ liệu bên trong (embstr, int, ziplist, …) | OBJECT encoding mystring |
| OBJECT idletime | OBJECT idletime key | Thời gian (giây) key chưa được truy cập | OBJECT idletime mystring |
| MEMORY USAGE | MEMORY USAGE key | Dung lượng bộ nhớ (byte) mà key đang chiếm | MEMORY USAGE mystring → 50 |

## Kết luận

Chương 1 đã trình bày toàn cảnh về Redis, từ những khái niệm nền tảng đến kiến trúc, đặc điểm và cách triển khai thực tế. Redis nổi bật với cơ chế lưu trữ dữ liệu trong bộ nhớ, mang lại tốc độ xử lý vượt trội và hỗ trợ nhiều kiểu dữ liệu linh hoạt. Các tính năng quan trọng như Persistence, Replication, Cluster và Sentinel không chỉ giúp hệ thống đảm bảo độ tin cậy và khả năng mở rộng mà còn mở ra nhiều ứng dụng thiết thực trong quản lý phiên, bộ nhớ đệm hay xử lý hàng đợi.

Song song đó, phần nội dung về cài đặt và sử dụng các công cụ hỗ trợ đã giúp việc quản trị Redis trở nên trực quan, dễ dàng và hiệu quả hơn. Những thao tác cơ bản như kiểm tra thông tin hệ thống, quản lý cơ sở dữ liệu hay giám sát tài nguyên cũng được làm rõ, góp phần xây dựng nền tảng vững chắc cho người đọc trước khi đi sâu vào các chương sau. Như vậy, chương 1 không chỉ cung cấp cái nhìn toàn diện về Redis mà còn đặt nền móng quan trọng để nghiên cứu các giải pháp ứng dụng và tối ưu hiệu suất trong thực tiễn.

# PHÂN TÍCH VÀ THIẾT KẾ CƠ SỞ DỮ LIỆU NoSQL

## Bối cảnh

Trong bối cảnh chuyển đổi số diễn ra mạnh mẽ, việc quản lý và khai thác cơ hội việc làm đã trở thành yêu cầu không thể thiếu đối với cả người lao động và doanh nghiệp. Hệ thống quản lý tuyển dụng hiện đại đóng vai trò then chốt như một cầu nối hiệu quả, giúp tối ưu hóa toàn bộ quy trình từ đăng tải tin tuyển dụng, tiếp nhận hồ sơ ứng tuyển cho đến các giai đoạn sàng lọc và phỏng vấn. Theo số liệu thống kê từ Bộ Lao động - Thương binh và Xã hội Việt Nam, năm 2024 đã ghi nhận hơn 1,5 triệu lao động trẻ gia nhập thị trường việc làm. Con số ấn tượng này đặt ra thách thức lớn cho các nền tảng tuyển dụng trong việc xử lý khối lượng dữ liệu khổng lồ, đồng thời đảm bảo tốc độ phản hồi và độ tin cậy ở mức cao nhất.

## Phân tích thị trường hiện tại

Khảo sát các nền tảng tuyển dụng phổ biến tại Việt Nam cho thấy mỗi hệ thống đều có những điểm mạnh và hạn chế riêng. TopCV nổi bật với khả năng matching tự động thông minh, tuy nhiên vẫn chưa tối ưu về xử lý dữ liệu real-time. CareerLink được đánh giá cao về giao diện thân thiện với doanh nghiệp vừa và nhỏ, nhưng hiệu suất xử lý giảm đáng kể khi quy mô dữ liệu tăng cao. VietnamWorks sở hữu lượng dữ liệu lớn nhưng đi kèm với chi phí cao và khả năng tùy chỉnh hạn chế.

## Giải pháp đề xuất

Để khắc phục những hạn chế trên, hệ thống đề xuất tận dụng sức mạnh của MongoDB - cơ sở dữ liệu NoSQL hướng tài liệu (document-oriented), cho phép lưu trữ dữ liệu phi cấu trúc một cách linh hoạt với khả năng mở rộng theo chiều ngang. Kết hợp với Redis - hệ thống caching in-memory hiệu suất cao, hỗ trợ xử lý real-time và quản lý session. Kiến trúc này tập trung vào hai collection chính với thiết kế nhúng (embedded) tối ưu: User (nhúng thông tin Company) và Job (nhúng danh sách Applicant).

## Phân tích yêu cầu hệ thống

### Yêu cầu chức năng

Hệ thống quản lý tuyển dụng được thiết kế để phục vụ ba nhóm đối tượng chính với các chức năng đặc thù:

**Ứng viên (Candidate):** Đây là vai trò mặc định khi người dùng đăng ký tài khoản mới. Ứng viên có khả năng tìm kiếm việc làm theo nhiều tiêu chí đa dạng như vị trí, địa điểm, mức lương và kinh nghiệm yêu cầu. Họ có thể nộp hồ sơ ứng tuyển trực tuyến kèm thư xin việc và tài liệu đính kèm, theo dõi trạng thái xét duyệt theo thời gian thực qua các giai đoạn sàng lọc và phỏng vấn, cũng như cập nhật hồ sơ cá nhân bao gồm kỹ năng, kinh nghiệm làm việc và trình độ học vấn.

**Nhà tuyển dụng (Recruiter):** Người dùng có thể chuyển đổi từ vai trò Candidate sang Recruiter thông qua quy trình xác minh của quản trị viên. Khi gửi yêu cầu chuyển đổi, người dùng phải cung cấp đầy đủ thông tin doanh nghiệp bao gồm tên công ty, website chính thức, email doanh nghiệp, số điện thoại liên hệ, quy mô nhân sự, lĩnh vực kinh doanh, địa chỉ trụ sở và giới thiệu về công ty. Nếu công ty đã tồn tại trong hệ thống, người dùng chỉ cần liên kết với công ty hiện có. Sau khi được phê duyệt, Recruiter có quyền đăng tải tin tuyển dụng, quản lý danh sách ứng viên, thực hiện sàng lọc hồ sơ thông qua hệ thống chấm điểm, lập lịch phỏng vấn và xuất các báo cáo thống kê về hiệu quả tuyển dụng.

**Quản trị viên (Admin):** Có toàn quyền quản lý người dùng trên hệ thống, phê duyệt các yêu cầu chuyển đổi vai trò từ Candidate sang Recruiter, xác minh thông tin doanh nghiệp, giám sát hoạt động tổng thể của nền tảng và tạo các báo cáo tổng hợp phục vụ công tác quản lý.

### Yêu cầu phi chức năng

**Hiệu năng:** Hệ thống cần đảm bảo thời gian phản hồi tối đa 200ms cho các truy vấn thông thường và hỗ trợ đồng thời tối thiểu 1.000 người dùng mà không giảm chất lượng dịch vụ. Đây là yêu cầu quan trọng trong bối cảnh thị trường tuyển dụng với lưu lượng truy cập cao vào giờ cao điểm.

**Bảo mật:** Triển khai mã hóa bcrypt cho mật khẩu người dùng với cost factor tối thiểu là 10, áp dụng xác thực JWT (JSON Web Token) với thời gian sống hợp lý cho mỗi phiên làm việc, mã hóa HTTPS cho mọi giao tiếp giữa client và server, và tuân thủ Nghị định 13/2023/NĐ-CP về bảo vệ dữ liệu cá nhân tại Việt Nam.

**Khả năng mở rộng:** Hệ thống phải có khả năng mở rộng theo chiều ngang (horizontal scaling) khi số lượng người dùng và dữ liệu tăng trưởng. MongoDB hỗ trợ sharding tự động để phân tán dữ liệu và replica set để đảm bảo tính sẵn sàng cao.

**Khả dụng:** Đảm bảo uptime 99.9% trong năm, tương đương tối đa 8.76 giờ downtime, với phương án backup tự động hàng ngày và kế hoạch disaster recovery cho phép khôi phục dữ liệu trong vòng 4 giờ (RTO - Recovery Time Objective).

## Lựa chọn công nghệ: So sánh SQL và NoSQL

### Phân tích đặc thù của hệ thống tuyển dụng

Trước khi đưa ra quyết định về công nghệ cơ sở dữ liệu, cần phân tích đặc điểm của hệ thống tuyển dụng:

**Tính đa dạng của dữ liệu:** Thông tin ứng viên và yêu cầu công việc rất đa dạng và thay đổi liên tục. Một vị trí trong ngành công nghệ có thể yêu cầu danh sách kỹ năng kỹ thuật chi tiết, trong khi vị trí marketing lại tập trung vào portfolio sáng tạo. Mỗi ngành nghề có những yêu cầu riêng biệt, khó có thể chuẩn hóa thành schema cố định.

**Mẫu truy vấn chủ đạo:** Hơn 80% các thao tác trong hệ thống là đọc dữ liệu - tìm kiếm công việc, xem hồ sơ ứng viên, hiển thị thông tin công ty. Các thao tác ghi (tạo hồ sơ, cập nhật trạng thái) chiếm tỷ lệ nhỏ hơn. Điều này ảnh hưởng đến việc lựa chọn mô hình tối ưu cho read-heavy workload.

**Nhu cầu mở rộng:** Khi hệ thống phát triển, số lượng người dùng và tin tuyển dụng có thể tăng trưởng nhanh chóng. Một nền tảng tuyển dụng thành công có thể có hàng triệu người dùng và hàng chục triệu tin tuyển dụng, đòi hỏi khả năng scale hiệu quả.

### So sánh SQL và NoSQL trong bối cảnh tuyển dụng

**Tính linh hoạt của schema:**

SQL với schema cứng nhắc yêu cầu định nghĩa trước cấu trúc bảng. Khi cần thêm trường mới (ví dụ: thêm trường “remote\_salary” cho các vị trí làm việc từ xa), cần thực hiện migration ALTER TABLE trên toàn bộ dữ liệu, gây downtime hoặc lock table. Đối với dữ liệu đa dạng như hồ sơ ứng viên, việc sử dụng nhiều bảng phụ (candidate\_skills, candidate\_certifications, candidate\_education) hoặc các trường nullable dẫn đến sự phức tạp không cần thiết.

MongoDB cho phép mỗi document có cấu trúc riêng biệt. Một ứng viên ngành IT có thể có mảng certifications chứa các chứng chỉ kỹ thuật, trong khi ứng viên ngành nghệ thuật có thể có trường portfolio chứa link các tác phẩm. Không cần migration khi thêm trường mới, chỉ cần cập nhật document cụ thể.

**Hiệu suất truy vấn:**

Trong SQL, để hiển thị một tin tuyển dụng đầy đủ cần JOIN nhiều bảng:

* JOIN với bảng Company để lấy thông tin công ty
* JOIN với bảng Applications để lấy danh sách ứng viên
* JOIN với bảng Users để lấy thông tin chi tiết mỗi ứng viên
* JOIN với bảng Job\_Skills để lấy yêu cầu kỹ năng

Mỗi JOIN tốn chi phí xử lý, đặc biệt khi dữ liệu lớn. Index có thể cải thiện nhưng không loại bỏ hoàn toàn overhead.

MongoDB với embedded documents có thể trả về toàn bộ thông tin trong một lần truy vấn duy nhất. Job document đã chứa sẵn mảng applicants với thông tin snapshot của mỗi ứng viên, không cần JOIN. Điều này đặc biệt quan trọng khi recruiter xem danh sách ứng viên - thao tác được thực hiện liên tục và cần tốc độ cao.

**Khả năng mở rộng:**

SQL truyền thống mở rộng theo chiều dọc (vertical scaling) - nâng cấp CPU, RAM, disk của server. Khi đạt giới hạn phần cứng, việc scale tiếp trở nên rất tốn kém. Sharding trong SQL phức tạp, thường cần công cụ bên thứ ba và yêu cầu thay đổi code ứng dụng đáng kể.

MongoDB hỗ trợ sharding tự động (horizontal scaling) - phân tán dữ liệu trên nhiều server theo shard key. Khi cần tăng capacity, chỉ cần thêm shard mới vào cluster mà không cần thay đổi code. MongoDB tự động cân bằng dữ liệu giữa các shard.

### Hạn chế của NoSQL và cách khắc phục

**Tính nhất quán dữ liệu:**

MongoDB ban đầu không đảm bảo ACID ở cấp độ multi-document như SQL. Tuy nhiên, từ phiên bản 4.0, MongoDB đã hỗ trợ multi-document transactions, cho phép thực hiện các thao tác phức tạp cần tính nhất quán. Trong hệ thống tuyển dụng, transactions được áp dụng khi cần cập nhật đồng thời nhiều documents quan trọng, ví dụ khi thay đổi trạng thái ứng tuyển và gửi thông báo.

**Dữ liệu trùng lặp:**

Với embedded documents, thông tin công ty được nhúng trong User document của mỗi recruiter có thể dẫn đến dư thừa. Giải pháp là chỉ lưu tham chiếu (companyId) trong User, và cache thông tin công ty quan trọng trong Redis để tăng tốc độ truy xuất. Khi thông tin công ty thay đổi, chỉ cần cập nhật một bản ghi chính và invalidate cache.

**Truy vấn phân tích phức tạp:**

Các báo cáo thống kê đa chiều (ví dụ: phân tích xu hướng tuyển dụng theo thời gian, ngành nghề, địa lý) khó thực hiện hơn so với SQL. Giải pháp là sử dụng MongoDB Aggregation Pipeline cho các báo cáo thời gian thực, và đồng bộ dữ liệu định kỳ sang data warehouse (PostgreSQL hoặc BigQuery) cho các phân tích sâu offline.

## Thiết kế cơ sở dữ liệu NoSQL

### Triết lý thiết kế: Embedded vs Reference

Trong MongoDB, có hai cách chính để mô hình hóa mối quan hệ giữa các thực thể:

**Embedding (Nhúng):** Dữ liệu con được lưu trực tiếp bên trong document cha dưới dạng nested document hoặc array. Phù hợp khi:

* Dữ liệu con có mối quan hệ chặt chẽ với dữ liệu cha (relationship “1-to-few”)
* Dữ liệu thường được truy vấn cùng nhau
* Dữ liệu con có kích thước giới hạn (MongoDB giới hạn document size là 16MB)

Trong hệ thống, danh sách applicants được nhúng trực tiếp vào Job document vì khi xem một tin tuyển dụng, recruiter thường muốn xem ngay danh sách ứng viên và trạng thái của họ.

**Referencing (Tham chiếu):** Lưu ObjectId của document khác để tham chiếu, giống foreign key trong SQL. Phù hợp khi:

* Dữ liệu có mối quan hệ lỏng lẻo
* Dữ liệu được truy cập độc lập
* Dữ liệu con có thể phát triển không giới hạn (relationship “1-to-many” hoặc “many-to-many”)

Chúng tôi lưu companyId như một reference trong User document thay vì nhúng toàn bộ company object để tránh dư thừa khi nhiều recruiter thuộc cùng một công ty.

### Mô hình dữ liệu chi tiết

Hệ thống sử dụng hai collections chính trong MongoDB với thiết kế denormalized nhằm tối ưu hiệu năng truy vấn.

* **Collection USERS**

Lưu trữ toàn bộ người dùng của hệ thống với cấu trúc linh hoạt hỗ trợ ba vai trò: candidate, recruiter và admin. Schema được thiết kế để cho phép chuyển đổi vai trò mà không cần migration dữ liệu.

**Cấu trúc chính:**

* Thông tin xác thực: email (unique index), passwordHash (bcrypt), role, isVerified
* Thông tin cá nhân chung: profile object chứa fullName, avatar, address, bio, socialLinks
* Thông tin theo vai trò:
  + candidateProfile: resumeUrl, skills, languages, education[], workExperience[], certifications[], expectedSalary, jobPreferences, profileCompleteness
  + recruiterProfile: position, department, employeeId, joinedDate
  + company: embedded snapshot thông tin công ty (denormalized)
* Metadata: statistics, lastLoginAt, isActive, timestamps

Thiết kế này áp dụng nested documents cho dữ liệu thường được truy vấn đồng thời (education, experience, certifications) và denormalization có chọn lọc (company snapshot) để giảm số lượng query.

* **Collection JOBS**

Lưu trữ tin tuyển dụng và toàn bộ dữ liệu ứng tuyển theo mô hình embedded documents. Thiết kế này tối ưu cho use case recruiter xem chi tiết công việc cùng danh sách ứng viên.

**Cấu trúc chính:**

* Thông tin công việc: title, jobDetails, requirements, benefits, salary{min, max, currency}
* Yêu cầu: experience, education, employmentType, gender, ageRange{min, max}
* Phân loại: skills[], categories[], keywords[] (indexed để tối ưu tìm kiếm)
* Thời gian và địa điểm: startDate, endDate, vacancies, workMode, workplace{address, city, district}
* Trạng thái: status, priority, isActive
* Thống kê: views, applicationCount, rejectedCount, interviewedCount, hiredCount
* companySnapshot: embedded object chứa {name, logoUrl, tier}

**Mảng applicants (Embedded Subdocuments):**

Mỗi phần tử là một application đầy đủ bao gồm:

* applicantSnapshot: bản chụp thông tin ứng viên tại thời điểm apply {fullName, email, phone, resumeUrl, currentPosition, yearsOfExperience}
* coverLetter, attachments[]
* status và statusHistory[]: timeline thay đổi trạng thái {status, changedAt, changedBy, note}
* screening: kết quả sàng lọc tự động {score, matchPercentage, strengths[], weaknesses[]}
* interviews[]: mảng vòng phỏng vấn {round, type, scheduledAt, interviewer, status, feedback, score}
* Timestamps: appliedAt, viewedByRecruiterAt, updatedAt

**Ưu điểm thiết kế:**

* Atomic operations: mọi thao tác với application là atomic trên một document
* Hiệu năng cao: một query duy nhất lấy đủ thông tin công việc và danh sách ứng viên
* Snapshot pattern: đảm bảo tính nhất quán dữ liệu theo thời gian
* Rich history tracking: statusHistory và interviews lưu trữ đầy đủ lịch sử

**Hạn chế:** Giới hạn 16MB/document của MongoDB có thể ảnh hưởng khi số lượng ứng viên lớn (>1000), phù hợp cho hệ thống quy mô vừa và nhỏ.

## So sánh hiệu năng truy vấn

### Kịch bản 1: Hiển thị chi tiết tin tuyển dụng với danh sách ứng viên

**Với SQL (thiết kế normalized):**

|  |
| --- |
| *-- Query 1: Lấy thông tin job*  SELECT \* FROM jobs WHERE id = ?;  *-- Query 2: Lấy thông tin công ty*  SELECT \* FROM companies WHERE id = ?;  *-- Query 3: Lấy danh sách applications*  SELECT \* FROM applications WHERE job\_id = ?;  *-- Query 4: Với mỗi application, lấy thông tin candidate (giả sử 15 ứng viên)*  SELECT \* FROM users WHERE id IN (?, ?, ..., ?);  *-- Query 5: Lấy skills của mỗi candidate*  SELECT \* FROM user\_skills WHERE user\_id IN (?, ?, ..., ?); |

Tổng: 5 queries với nhiều JOINs, mỗi query phải scan indexes và thực hiện JOIN. Với 15 ứng viên, query 5 phải xử lý khoảng 75-150 records (trung bình mỗi ứng viên 5-10 skills).

Thời gian ước tính: 150-300ms (tùy thuộc vào network latency và database load)

**Với MongoDB (thiết kế embedded):**

|  |
| --- |
| db.jobs.findOne({ \_id: jobId }) |

Chỉ 1 query duy nhất trả về toàn bộ Job document bao gồm mảng applicants với đầy đủ thông tin. MongoDB đọc một document liên tục từ disk, không cần JOIN.

Thời gian ước tính: 10-30ms

Kết luận: MongoDB nhanh hơn 5-10 lần cho trường hợp này.

### Kịch bản 2: Tìm kiếm công việc theo nhiều tiêu chí

Yêu cầu: Tìm job trong ngành IT, ở TP.HCM, yêu cầu skill “React”, mức lương 20-40 triệu, đang active.

**Với SQL:**

|  |
| --- |
| SELECT j.\*, c.name as company\_name, c.logo  FROM jobs j  JOIN companies c ON j.company\_id = c.id  JOIN job\_skills js ON j.id = js.job\_id  WHERE j.status = 'published'  AND j.salary\_min >= 20000000  AND j.salary\_max <= 40000000  AND j.location\_city = 'Hồ Chí Minh'  AND js.skill\_name = 'React'  AND j.category = 'IT'  ORDER BY j.created\_at DESC  LIMIT 20; |

JOIN giữa jobs, companies và job\_skills. Với index hợp lý, query vẫn phải scan nhiều records.

Thời gian ước tính: 50-100ms

**Với MongoDB:**

|  |
| --- |
| db.jobs.find({  status: 'published',  'salary.min': { $gte: 20000000 },  'salary.max': { $lte: 40000000 },  'locations.city': 'Hồ Chí Minh',  'skills.name': 'React',  categories: 'IT'  })  .sort({ createdAt: -1 })  .limit(20) |

Không cần JOIN vì thông tin công ty có thể được cache. Với compound index phù hợp, MongoDB scan rất hiệu quả.

Thời gian ước tính: 20-40ms

Kết luận: MongoDB nhanh hơn 2-3 lần nhờ tránh JOIN và tận dụng document model.

### Kịch bản 3: Cập nhật trạng thái ứng viên

**Với SQL:**

|  |
| --- |
| BEGIN TRANSACTION;  *-- Update application status*  UPDATE applications  SET status = 'interviewing',  updated\_at = NOW()  WHERE id = ?;  *-- Insert timeline record*  INSERT INTO application\_timeline (application\_id, status, changed\_at, changed\_by)  VALUES (?, 'interviewing', NOW(), ?);  *-- Update job stats*  UPDATE jobs  SET interviewing\_count = interviewing\_count + 1,  pending\_count = pending\_count - 1  WHERE id = ?;  COMMIT; |

Cần 3 queries trong transaction để đảm bảo tính nhất quán.

Thời gian ước tính: 30-60ms

**Với MongoDB:**

|  |
| --- |
| db.jobs.updateOne(  {  \_id: jobId,  'applicants.\_id': applicationId  },  {  $set: {  'applicants.$.status': 'interviewing',  'applicants.$.lastStatusUpdate': new Date()  },  $push: {  'applicants.$.timeline': {  status: 'interviewing',  changedAt: new Date(),  changedBy: userId  }  },  $inc: {  'stats.interviewingCount': 1,  'stats.pendingCount': -1  }  }  ) |

Chỉ 1 atomic operation cập nhật nhiều trường trong cùng một document. MongoDB đảm bảo atomicity ở document level.

Thời gian ước tính: 10-20ms

Kết luận: MongoDB nhanh hơn 2-3 lần nhờ atomic operations và tránh transaction overhead.

### Tổng kết so sánh hiệu năng

Bảng .: Bảng tổng kết so sánh hiệu năng

|  |  |  |
| --- | --- | --- |
| Kịch bản | SQL | MongoDB |
| Hiển thị job + applicants | 150-300ms | 10-30ms |
| Tìm kiếm job | 50-100ms | 20-40ms |
| Cập nhật trạng thái | 30-60ms | 10-20ms |

MongoDB có lợi thế rõ rệt trong các trường hợp:

* Truy vấn dữ liệu có mối quan hệ 1-to-many cần hiển thị cùng nhau
* Read-heavy workload với ít write
* Dữ liệu có cấu trúc linh hoạt, thay đổi thường xuyên

SQL vẫn phù hợp hơn cho:

* Các truy vấn phân tích phức tạp với nhiều aggregations
* Dữ liệu có mối quan hệ many-to-many phức tạp
* Yêu cầu ACID transactions nghiêm ngặt across nhiều entities

Dưới đây là đoạn báo cáo chi tiết được viết lại theo văn phong học thuật, chỉ tập trung mô tả các chiến lược đã được thiết kế và áp dụng trong hệ thống, loại bỏ các phần chưa triển khai và các đoạn mã nguồn.

## Phân tích Thiết kế Cơ chế Lưu trữ Đệm với Redis

Để tối ưu hóa hiệu năng hệ thống, giảm độ trễ truy xuất dữ liệu và giảm tải cho cơ sở dữ liệu chính (MongoDB), kiến trúc của dự án tích hợp Redis làm một lớp đệm (cache layer) chiến lược. Trong phạm vi triển khai hiện tại, hệ thống tập trung vào hai chiến lược chính: Quản lý phiên (Session Management) và Caching kết quả truy vấn (Query Result Caching) theo mô hình Cache-Aside.

### Cơ chế Quản lý Phiên

Hệ thống sử dụng Redis để tăng cường bảo mật và tính linh hoạt trong việc quản lý phiên người dùng, cụ thể là quản lý vòng đời của Refresh Tokens. Thay vì chỉ dựa vào cơ chế stateless của JWT (JSON Web Tokens), hệ thống lưu trữ trạng thái của các Refresh Token đang hoạt động trên Redis.

**Quy trình hoạt động được thiết kế như sau:**

1. **Khi đăng nhập:** Sau khi người dùng xác thực thành công, hệ thống tạo ra một access\_token (có thời hạn ngắn) và một refresh\_token (có thời hạn dài). Refresh\_token này sau đó được lưu vào Redis dưới dạng một cặp key-value, trong đó key là chính token và value là định danh của người dùng (UserID). Một Thời gian Tồn tại (Time To Live - TTL) được thiết lập, khớp với thời hạn của refresh token (ví dụ: 7 ngày).
2. **Khi làm mới token (Refresh):** Khi access\_token hết hạn, máy khách (client) gửi refresh\_token lên máy chủ. Hệ thống sẽ truy vấn Redis để kiểm tra sự tồn tại của token này. Nếu token tồn tại và hợp lệ trong Redis, hệ thống sẽ cấp một access\_token mới và tiếp tục phiên làm việc.
3. **Khi đăng xuất:** Khi người dùng chủ động chọn “Đăng xuất”, hệ thống sẽ gửi yêu cầu xóa key tương ứng với refresh\_token đó khỏi Redis. Điều này đảm bảo rằng token đó ngay lập tức bị vô hiệu hóa và không thể sử dụng để yêu cầu cấp access\_token mới, bất kể thời hạn (TTL) ban đầu của nó là bao lâu.

Chiến lược này cho phép máy chủ chủ động thu hồi các phiên đăng nhập, giải quyết điểm yếu cố hữu của JWT stateless là không thể thu hồi token trước khi nó tự hết hạn.

### Chiến lược Caching Kết quả Truy vấn

Để giải quyết bài toán thắt cổ chai hiệu năng ở các tác vụ đọc dữ liệu (read-heavy), hệ thống triển khai triệt để mô hình Cache-Aside. Đây là chiến lược caching phổ biến nhất, trong đó logic của ứng dụng chịu trách nhiệm chính trong việc quản lý dữ liệu cache.

**Luồng xử lý dữ liệu theo mô hình Cache-Aside:**

1. **Kiểm tra Cache (Cache Check):** Khi ứng dụng nhận được một yêu cầu truy vấn dữ liệu (ví dụ: GET /api/jobs), tầng dịch vụ (service layer) đầu tiên sẽ tạo ra một “cache key” (khóa cache) duy nhất. Khóa này thường là sự kết hợp của tên thực thể và các tham số truy vấn (ví dụ: jobs:public:page:1:size:20).
2. **Cache Hit:** Ứng dụng dùng khóa này để truy vấn Redis. Nếu tìm thấy dữ liệu (Cache Hit), dữ liệu sẽ được trả về ngay lập tức cho người dùng, bỏ qua hoàn toàn việc truy vấn cơ sở dữ liệu chính. Điều này làm giảm đáng kể thời gian phản hồi, thường xuống mức dưới 30ms.
3. **Cache Miss:** Nếu không tìm thấy dữ liệu trong Redis (Cache Miss), ứng dụng sẽ thực hiện bước tiếp theo là truy vấn cơ sở dữ liệu MongoDB để lấy dữ liệu.
4. **Cập nhật Cache (Cache Population):** Sau khi nhận được kết quả từ MongoDB, ứng dụng sẽ ghi (lưu) kết quả này vào Redis với “cache key” đã tạo ở bước 1, đồng thời thiết lập một TTL (ví dụ: 5 phút). Cuối cùng, dữ liệu được trả về cho người dùng. Các yêu cầu tương tự tiếp theo trong vòng 5 phút sẽ được hưởng lợi từ “Cache Hit”.

Chiến lược này được áp dụng toàn diện trên các điểm cuối (endpoints) có tần suất đọc cao và dữ liệu tương đối ít thay đổi, bao gồm:

* Danh sách việc làm (phân trang và lọc theo vai trò).
* Hồ sơ chi tiết của người dùng và công ty.
* Các dữ liệu thống kê và biểu đồ trên Dashboard.

**Đảm bảo Tính nhất quán Dữ liệu**

Một thách thức của caching là dữ liệu trong cache có thể trở nên cũ (stale) khi dữ liệu gốc trong MongoDB thay đổi. Để giải quyết vấn đề này, hệ thống áp dụng chiến lược vô hiệu hóa cache (Cache Invalidation) một cách chủ động.

Khi một hành động ghi (Write) xảy ra (ví dụ: POST, PUT, DELETE), ứng dụng không chỉ cập nhật dữ liệu trong MongoDB mà còn thực hiện một lệnh xóa (ví dụ: DEL) các cache key có liên quan trong Redis. Ví dụ, khi một nhà tuyển dụng cập nhật thông tin một công việc, hệ thống sẽ xóa tất cả các key liên quan đến danh sách công việc (jobs:public:\*, jobs:admin:\*) và key chi tiết của công việc đó.

Điều này đảm bảo rằng lần truy cập tiếp theo sẽ là một “Cache Miss”, buộc hệ thống phải tải lại dữ liệu mới nhất từ MongoDB và cập nhật vào cache, qua đó duy trì tính nhất quán giữa cache và cơ sở dữ liệu.

## Kết luận

Chương 2 đã trình bày chi tiết quá trình phân tích và thiết kế cơ sở dữ liệu NoSQL cho hệ thống quản lý tuyển dụng. Việc lựa chọn MongoDB thay vì SQL truyền thống dựa trên những phân tích cụ thể về đặc thù của hệ thống: tính đa dạng của dữ liệu, mẫu truy vấn read-heavy và nhu cầu mở rộng linh hoạt.

Thiết kế hai collections chính (User và Job) với chiến lược embedding cho các mối quan hệ 1-to-many đã được chứng minh mang lại hiệu năng vượt trội, với tốc độ truy vấn nhanh hơn 2-10 lần so với mô hình SQL normalized. Kết quả so sánh cho thấy MongoDB đặc biệt phù hợp cho các trường hợp cần hiển thị dữ liệu có mối quan hệ chặt chẽ, như việc hiển thị tin tuyển dụng cùng danh sách ứng viên.

Chiến lược kết hợp MongoDB cho persistent storage và Redis cho caching tạo nên một kiến trúc cân bằng giữa hiệu năng, tính linh hoạt và khả năng mở rộng, đáp ứng tốt các yêu cầu của một nền tảng tuyển dụng hiện đại với hàng nghìn người dùng đồng thời.

# TRIỂN KHAI ỨNG DỤNG

## Giới thiệu chương

Chương 3 trình bày chi tiết quá trình triển khai hệ thống quản lý tuyển dụng, tập trung vào việc xây dựng ứng dụng web API với các chức năng cốt lõi bao gồm quản lý người dùng, công ty, tin tuyển dụng và hồ sơ ứng tuyển. Mục tiêu của chương là làm rõ phương thức hệ thống được thiết kế và triển khai dựa trên kiến trúc phân tầng, đồng thời phân tích vai trò cụ thể của MongoDB và Redis trong việc bảo đảm tính linh hoạt, khả năng mở rộng và hiệu năng.

Hệ thống sử dụng MongoDB làm cơ sở dữ liệu chính, lưu trữ dữ liệu dưới dạng tài liệu BSON, hỗ trợ các cấu trúc lồng nhau phức tạp như CompanySnapshot được nhúng trực tiếp vào tài liệu Job. Hệ thống indexing đa dạng (single field, compound, multikey, text) được áp dụng để tối ưu hóa các truy vấn phức tạp. Redis được triển khai theo chiến lược cache-aside với giá trị TTL (Time To Live) 5 phút, nhằm giảm tải cho cơ sở dữ liệu và cải thiện thời gian phản hồi của các endpoint có tần suất truy cập cao.

Cấu trúc nội dung chương được tổ chức theo trình tự từ tổng quan đến chi tiết, bắt đầu với kiến trúc hệ thống và vai trò của các tác nhân. Tiếp theo, chương đi sâu vào phương thức tích hợp MongoDB thông qua Repository Pattern, chiến lược truy vấn phân tầng theo vai trò, cơ chế cache Redis, và cấu hình môi trường. Cuối cùng, chương mô tả các nhóm API chính, phương pháp kiểm thử qua Postman Collection và tổng quan giao diện người dùng.

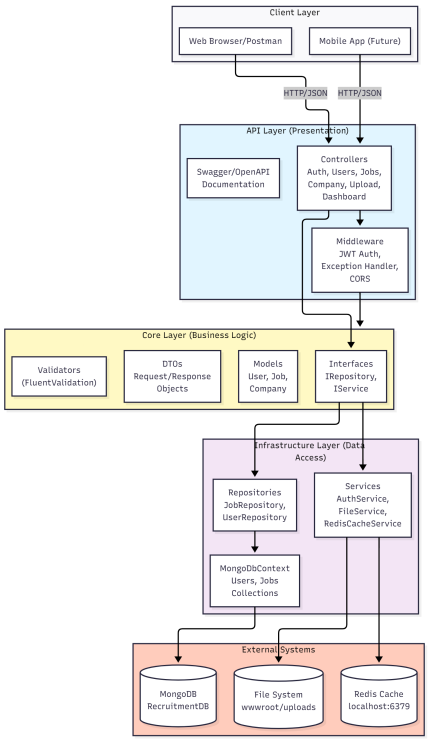
## Tổng quan kiến trúc hệ thống

### Kiến trúc tổng thể

Hệ thống được thiết kế dựa trên mô hình kiến trúc ba tầng, kết hợp các nguyên tắc của Clean Architecture, nhằm bảo đảm tính tách biệt rõ ràng (Separation of Concerns) giữa các thành phần, tạo điều kiện thuận lợi cho bảo trì và mở rộng. Kiến trúc này định nghĩa các giao thức (interface) rõ ràng cho việc giao tiếp giữa các tầng, giúp giảm thiểu sự phụ thuộc chặt chẽ và tăng khả năng tái sử dụng mã nguồn.

Hệ thống được phân chia thành ba tầng chính: Tầng Trình diễn (Presentation Layer), Tầng Logic Nghiệp vụ (Business Logic Layer), và Tầng Truy cập Dữ liệu (Data Access Layer). Tầng Trình diễn chịu trách nhiệm xử lý các yêu cầu HTTP, thực hiện xác thực và ủy quyền, sau đó chuyển tiếp yêu cầu đến Tầng Logic Nghiệp vụ. Tầng Logic Nghiệp vụ chứa các quy tắc nghiệp vụ cốt lõi, xử lý dữ liệu và điều phối luồng công việc. Tầng Truy cập Dữ liệu chịu trách nhiệm tương tác trực tiếp với cơ sở dữ liệu MongoDB và dịch vụ cache Redis.

Kiến trúc đồng thời tích hợp các dịch vụ bổ trợ như JWT (JSON Web Token) để quản lý phiên, dịch vụ xử lý tệp tin (CV, hình ảnh) và dịch vụ cache Redis. Các dịch vụ này được quản lý thông qua cơ chế Dependency Injection (DI) của ASP.NET Core. Luồng dữ liệu được tổ chức tuần tự: Client → API Layer (Controllers) → Services & Business Logic → Repositories (Data Access) → MongoDB/Redis Cache.



Hình .: Sơ đồ kiến trúc hệ thống

### Các tầng chính

Tầng Presentation (API Layer) Tầng Trình diễn được triển khai bằng ASP.NET Core 8 Web API, với vai trò là giao diện (endpoint) chính tiếp nhận các tương tác từ client. Tầng này bao gồm các Controller (ví dụ: JobsController, UsersController) xử lý các nhóm chức năng nghiệp vụ cụ thể. Các Controller tiếp nhận yêu cầu HTTP dưới dạng JSON, thực hiện kiểm tra tính hợp lệ của dữ liệu đầu vào (Input Validation) thông qua Data Annotations và FluentValidation, sau đó ủy thác việc xử lý logic cho các Service tương ứng. Tầng API cũng tích hợp các Middleware để xử lý xác thực (Authentication), phân quyền (Authorization), xử lý ngoại lệ toàn cục (Global Exception Handling) và ghi log. Middleware xác thực JWT kiểm tra tính hợp lệ của token. Tầng này đồng thời cấu hình Swagger/OpenAPI để tự động sinh tài liệu API, hỗ trợ quá trình tích hợp.

Tầng Core (Business Logic Layer) Tầng Core chứa các quy tắc nghiệp vụ cốt lõi và được thiết kế độc lập với các chi tiết triển khai (framework, cơ sở dữ liệu). Tầng này bao gồm các thành phần chính: Models (định nghĩa cấu trúc dữ liệu), DTOs (Data Transfer Objects - đối tượng truyền tải dữ liệu), Interfaces (định nghĩa các hợp đồng cho repository và service), và các lớp Validation. Các Model trong tầng Core phản ánh cấu trúc của các tài liệu MongoDB. Các DTO được sử dụng cho mục đích truyền tải dữ liệu giữa các tầng, đồng thời che giấu các chi tiết hạ tầng không cần thiết. Thư viện AutoMapper được sử dụng để thực hiện ánh xạ tự động giữa Models và DTOs. Tầng Core định nghĩa các Interface (ví dụ: IJobRepository, IUserRepository, ICacheService), đóng vai trò là cầu nối tuân thủ nguyên tắc Dependency Inversion.

Tầng Infrastructure (Data Access Layer) Tầng Infrastructure chịu trách nhiệm triển khai các Interface được định nghĩa trong tầng Core và tương tác trực tiếp với các dịch vụ bên ngoài (MongoDB, Redis, hệ thống tệp tin). Tầng này bao gồm các thành phần: MongoDbContext quản lý kết nối MongoDB, các lớp Repository (ví dụ: JobRepository) thực thi các thao tác CRUD, và RedisCacheService quản lý cache. MongoDbContext được đăng ký dưới dạng Singleton Service, đảm bảo một thực thể (instance) duy nhất được duy trì. Context này cung cấp các IMongoCollection tương ứng với từng tài liệu. Các Repository sử dụng Context này để thực hiện truy vấn. T tầng Infrastructure cũng chứa các dịch vụ như AuthService (xử lý JWT) và FileService (xử lý tải tệp).

### Vai trò của các tác nhân

Hệ thống định nghĩa bốn loại tác nhân (actor) chính, mỗi tác nhân có vai trò và quyền hạn (permission) riêng biệt, đảm bảo an ninh và tối ưu hóa luồng nghiệp vụ.

* **Public User (Người dùng công khai):** Được cấp quyền truy cập không cần xác thực vào các tài nguyên công khai, chủ yếu là danh sách các tin tuyển dụng đã được duyệt (published jobs). Tác nhân này có thể thực hiện tìm kiếm việc làm. Để thực hiện các thao tác khác (ví dụ: nộp hồ sơ), người dùng bắt buộc phải đăng ký và đăng nhập.
* **Candidate (Ứng viên):** Là người dùng đã xác thực, có quyền xem, tìm kiếm và nộp hồ sơ ứng tuyển. Ứng viên có thể quản lý thông tin cá nhân, tải lên CV (PDF, DOCX), và theo dõi trạng thái các đơn ứng tuyển. Hệ thống cho phép ứng viên gửi yêu cầu đăng ký công ty mới, chuyển vai trò thành Recruiter sau khi được Admin phê duyệt.
* **Recruiter (Nhà tuyển dụng):** Là đại diện của các công ty đã được xác minh. Recruiter có quyền đăng tin tuyển dụng (ở trạng thái nháp - draft), chỉnh sửa và công bố (publish). Một chức năng quan trọng là quản lý các đơn ứng tuyển (Applications) cho các tin tuyển dụng thuộc công ty của mình, bao gồm xem danh sách, tải CV và cập nhật trạng thái.
* **Admin (Quản trị viên):** Có quyền hạn cao nhất, thực hiện quản lý và giám sát toàn bộ tài nguyên (người dùng, công ty, tin tuyển dụng). Admin có khả năng truy cập tất cả tin tuyển dụng bất kể trạng thái (trừ các tin đã xóa mềm), thực hiện chỉnh sửa hoặc xóa. Vai trò Admin cũng bao gồm việc phê duyệt các yêu cầu đăng ký công ty mới và truy cập dashboard thống kê.

## Hệ thống cơ sở dữ liệu và lưu trữ

### Tích hợp MongoDB

Hệ thống sử dụng MongoDB làm cơ sở dữ liệu chính. Cấu trúc lưu trữ dạng tài liệu BSON cho phép lưu trữ các đối tượng lồng nhau (nested objects) và mảng (arrays) một cách tự nhiên.

Một ví dụ điển hình là tài liệu Job nhúng (embed) một bản sao thông tin công ty (CompanySnapshot) tại thời điểm đăng tin. Cách tiếp cận này giúp giảm thiểu số lượng truy vấn (query) cần thiết khi hiển thị danh sách tin tuyển dụng và đảm bảo tính nhất quán thời điểm (point-in-time consistency), nghĩa là thông tin công ty hiển thị trong tin tuyển dụng không bị ảnh hưởng bởi các cập nhật sau đó của tài liệu Company gốc.

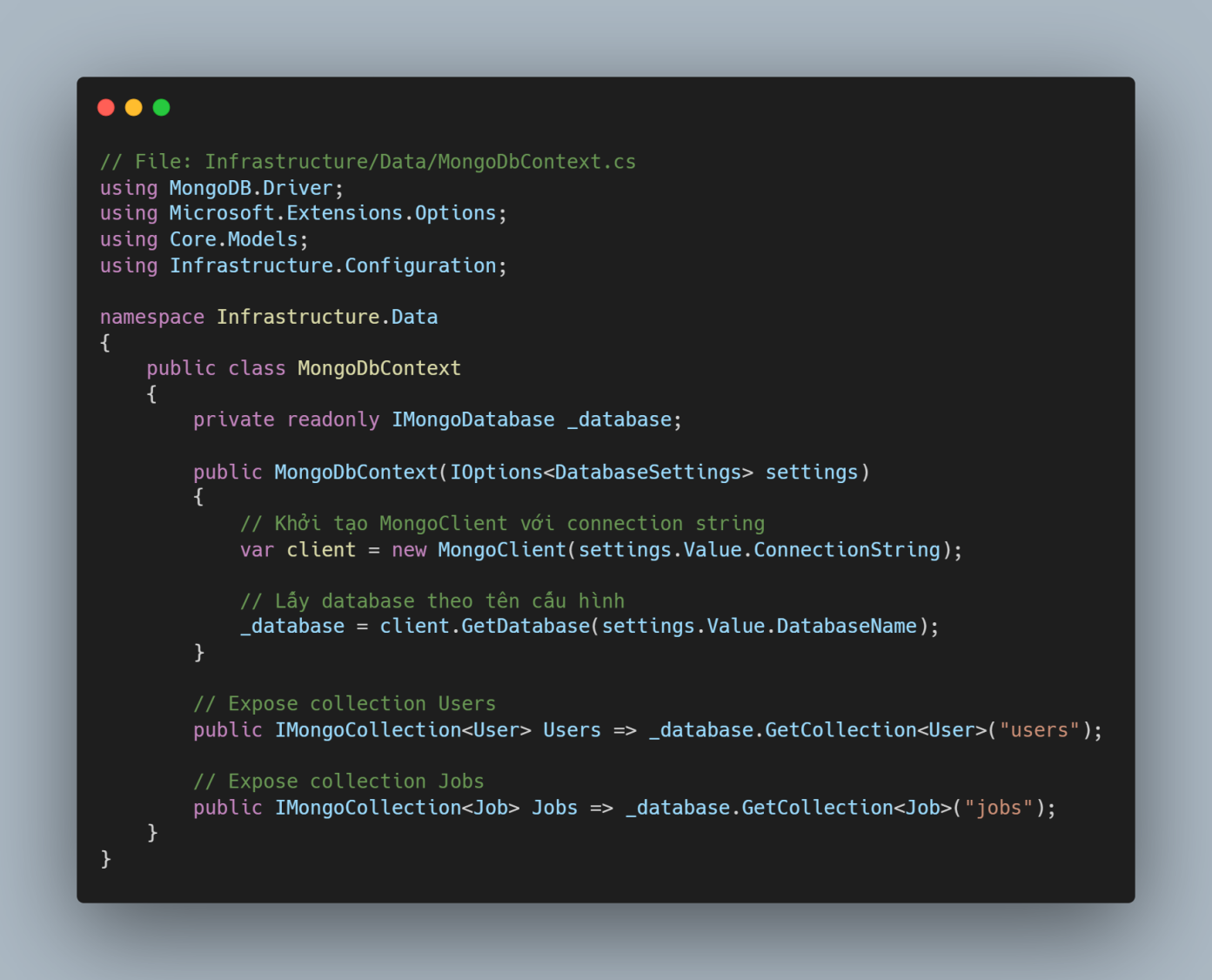
Hệ thống chỉ mục (indexing) đa dạng của MongoDB (single field, compound, multikey, text) được sử dụng để tối ưu hóa các truy vấn phức tạp. Driver .NET chính thức (MongoDB.Driver) được tích hợp, cung cấp API hỗ trợ LINQ, cơ chế bất đồng bộ (async/await) và các collection định kiểu mạnh (strongly-typed).

### Cấu hình, kết nối và Repository Pattern

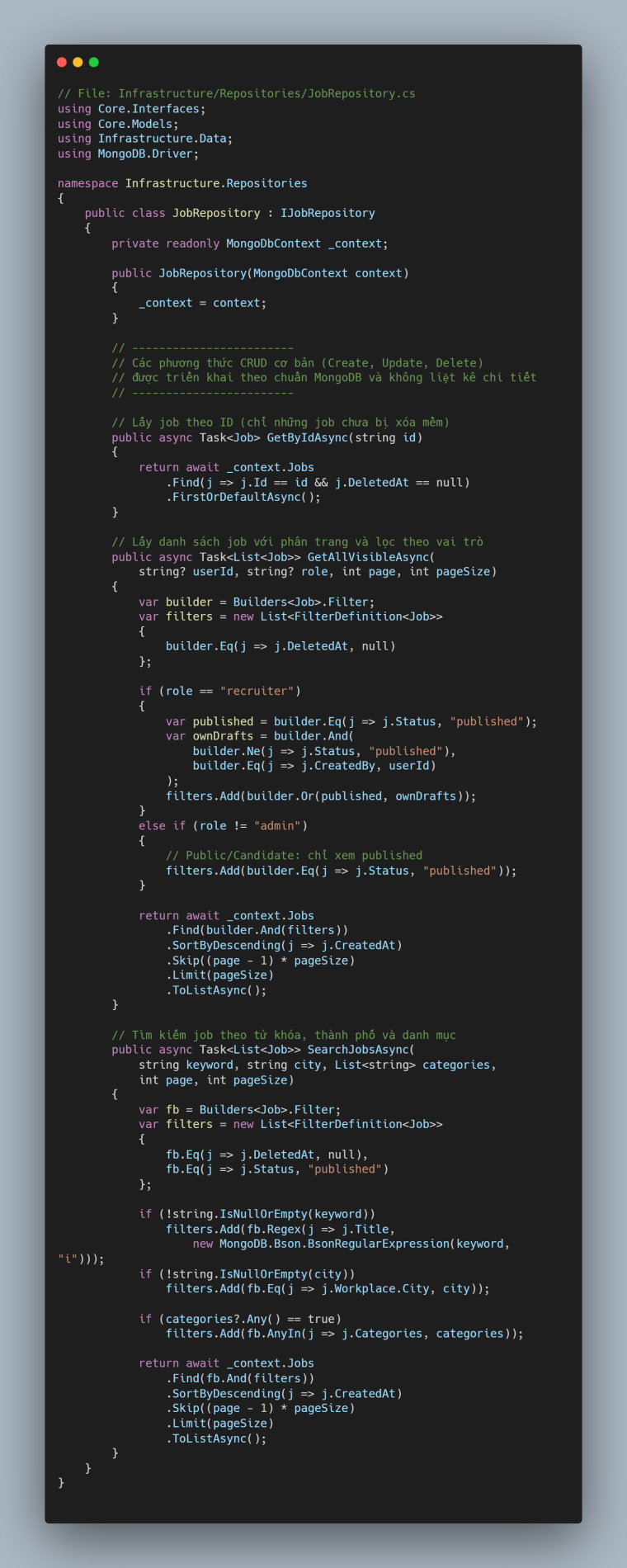
Quá trình cấu hình và kết nối đến MongoDB được quản lý thông qua lớp DatabaseSettings, chứa các thuộc tính ConnectionString và DatabaseName. Các giá trị này được nạp (bind) từ tệp appsettings.json thông qua Options Pattern của ASP.NET Core, cho phép quản lý cấu hình linh hoạt giữa các môi trường (Development, Staging, Production).

Lớp MongoDbContext được khởi tạo, nhận IOptions<DatabaseSettings> qua constructor injection, khởi tạo MongoClient và truy cập IMongoDatabase tương ứng. MongoDbContext được đăng ký dưới dạng Singleton Service để tối ưu hóa quản lý kết nối.

Hệ thống áp dụng Repository Pattern để trừu tượng hóa tầng truy cập dữ liệu. Mỗi thực thể (entity) chính có một Interface (ví dụ: IJobRepository) định nghĩa trong Tầng Core. Các lớp triển khai (implementation) của các interface này (ví dụ: JobRepository) nằm trong Tầng Infrastructure, nhận MongoDbContext qua DI và sử dụng các IMongoCollection để thực thi các thao tác dữ liệu.



Hình .: Khởi tạo MongoDbContext



Hình .: Triển khai lớp JobRepository

### Cấu trúc dữ liệu, phân trang và lọc vai trò

Tài liệu (document) Job là thực thể trung tâm, chứa các trường như tiêu đề, mô tả, yêu cầu, trạng thái (draft/published), tài liệu nhúng CompanySnapshot, địa điểm (Workplace), danh mục (Categories), từ khóa (Keywords), và các metadata (CreatedBy, CreatedAt, UpdatedAt, DeletedAt). Thiết kế này hỗ trợ cơ chế xóa mềm (soft delete) và theo dõi lịch sử.

Cơ chế phân trang được triển khai theo phương pháp offset-based, sử dụng công thức Skip((page-1) × pageSize) và Limit(pageSize). Kết quả được đóng gói trong đối tượng PagedResponse bao gồm danh sách items, CurrentPage, PageSize, TotalCount và TotalPages.

Việc lọc dữ liệu theo vai trò được triển khai bằng FilterDefinitionBuilder của MongoDB. Chiến lược lọc được định nghĩa: Public User và Candidate chỉ xem được các tin published và chưa bị xóa mềm; Recruiter xem được các tin published và các tin draft do chính họ tạo; Admin xem được tất cả các tin (trừ các tin đã xóa mềm). Chức năng tìm kiếm nâng cao (keyword, city, categories) được áp dụng trên tập dữ liệu đã published.



Hình .: JSON Example đầy đủ của Job Document

## Cơ chế lưu trữ tạm và tối ưu hiệu năng (Redis)

### Redis Cache và cơ chế hoạt động

Hệ thống triển khai Redis, một kho lưu trữ dữ liệu trong bộ nhớ (in-memory data store) có độ trễ thấp, làm lớp đệm (cache) giữa API và MongoDB. Mục tiêu chính là tối ưu hóa các endpoint có tần suất đọc cao, ví dụ như danh sách việc làm.

Hệ thống lưu trữ các đối tượng DTO đã được tuần tự hóa (serialized) sang JSON dưới dạng String của Redis. Tham số TTL (Time To Live) được áp dụng để dữ liệu tự động hết hạn, đảm bảo tính tươi mới (freshness) của dữ liệu. Chính sách giải phóng bộ nhớ (eviction policy) như LRU (Least Recently Used) được cấu hình để quản lý bộ nhớ hiệu quả khi đạt đến giới hạn.

### Tích hợp Redis trong ASP.NET Core

Lớp cấu hình RedisSettings (bao gồm ConnectionString và InstanceName) được nạp từ appsettings.json. ConnectionMultiplexer (từ thư viện StackExchange.Redis) được đăng ký dưới dạng Singleton để tái sử dụng kết nối một cách hiệu quả và an toàn trong môi trường đa luồng (thread-safe).

Interface ICacheService được định nghĩa để trừu tượng hóa các thao tác (Get/Set/Remove/Exists). Lớp RedisCacheService triển khai interface này, sử dụng ConnectionMultiplexer và được inject vào các service hoặc controller cần thiết, tuân thủ nguyên tắc Dependency Inversion.



Hình .: Code đầy đủ RedisCacheService

### Cache-aside và TTL

Hệ thống áp dụng chiến lược cache-aside (look-aside caching). Khi một yêu cầu dữ liệu được thực hiện:

1. Hệ thống kiểm tra ICacheService trước.
2. Nếu tìm thấy (cache hit), dữ liệu được trả về trực tiếp từ Redis.
3. Nếu không tìm thấy (cache miss), hệ thống truy vấn MongoDB, sau đó lưu kết quả vào Redis với TTL đã định trước, và cuối cùng trả về dữ liệu cho client.

Ưu điểm của phương pháp này là chỉ dữ liệu được yêu cầu mới được cache, và hệ thống vẫn có thể hoạt động (dù chậm hơn) nếu Redis tạm thời gặp sự cố.

Chiến lược định danh khóa (key strategy) được thiết kế theo mẫu jobs:{viewerKey}:page:{page}:size:{pageSize} để đảm bảo tính tách biệt của dữ liệu cache dựa trên vai trò người xem, số trang và kích thước trang.



Hình .: Code snippet GetJobs action với cache-aside và key strategy

### Invalidation theo scope

Cơ chế vô hiệu hóa cache (cache invalidation) theo phạm vi được thiết kế để cân bằng giữa tính nhất quán (consistency) và hiệu năng.

* Khi tạo tin draft, chỉ cache của Recruiter đó bị vô hiệu hóa.
* Khi publish hoặc unpublish một tin, cache của Public, Candidate và Admin liên quan đến danh sách chung sẽ bị vô hiệu hóa.
* Khi update hoặc delete, các scope cache liên quan (bao gồm cả danh sách và chi tiết) sẽ bị xóa.

Việc lựa chọn phạm vi vô hiệu hóa phù hợp là rất quan trọng: under-invalidation (xóa không đủ) có thể dẫn đến dữ liệu cũ (stale data), trong khi over-invalidation (xóa quá nhiều) làm giảm tỷ lệ cache hit.



Hình .: Code snippet PublishJob action với invalidation logic

### So sánh hiệu năng

Để kiểm chứng hiệu quả thực tế của cơ chế cache-aside đã triển khai, một loạt các kiểm thử đơn lẻ (single-request validation) đã được thực hiện bằng công cụ Postman. Phương pháp kiểm thử bao gồm việc gửi yêu cầu đến cùng một endpoint hai lần liên tiếp:

* Lần gọi 1 (Cache Miss): Đây là lần gọi đầu tiên khi dữ liệu chưa tồn tại trong Redis. Hệ thống phải thực hiện truy vấn đầy đủ đến MongoDB, sau đó tuần tự hóa (serialize) kết quả và lưu vào cache trước khi trả về cho client.
* Lần gọi 2 (Cache Hit): Lần gọi này được thực hiện ngay sau đó. Hệ thống phát hiện khóa (key) đã tồn tại trong cache, lấy dữ liệu trực tiếp từ bộ nhớ Redis và trả về ngay lập tức, bỏ qua hoàn toàn bước truy vấn cơ sở dữ liệu.

Kết quả so sánh thời gian phản hồi (Response Time) trên các endpoint tiêu biểu được ghi nhận trực tiếp từ Postman.

Bảng . So sánh hiệu năng Cache Hit và Cache Miss bằng Postman

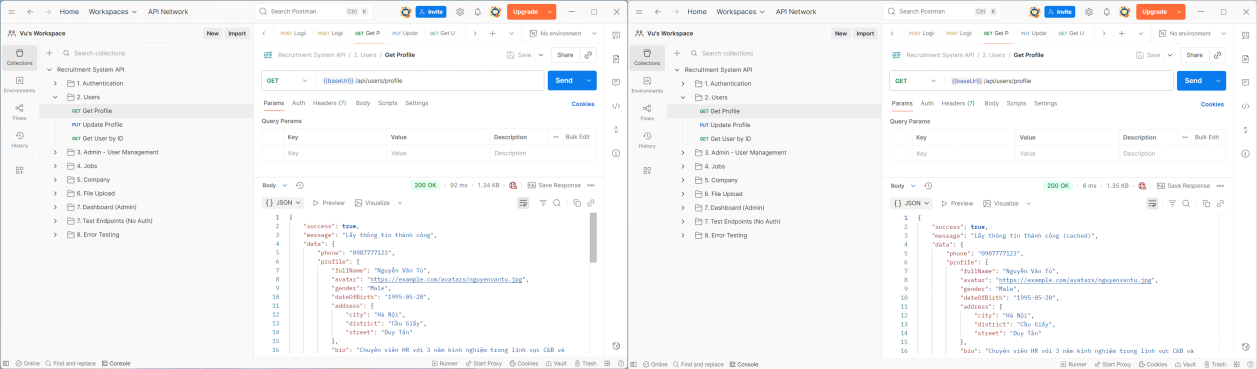
|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Loại kiểm tra** | **Thời gian (ms)** |
| GET .../jobs?page=1&pageSize=20 | Cache Miss (Lần 1) | **54 ms** |
| GET .../jobs?page=1&pageSize=20 | Cache Hit (Lần 2) | **13 ms** |
| GET .../api/users/profile | Cache Miss (Lần 1) | **92 ms** |
| GET .../api/users/profile | Cache Hit (Lần 2) | **6 ms** |

**Phân tích kết quả:**

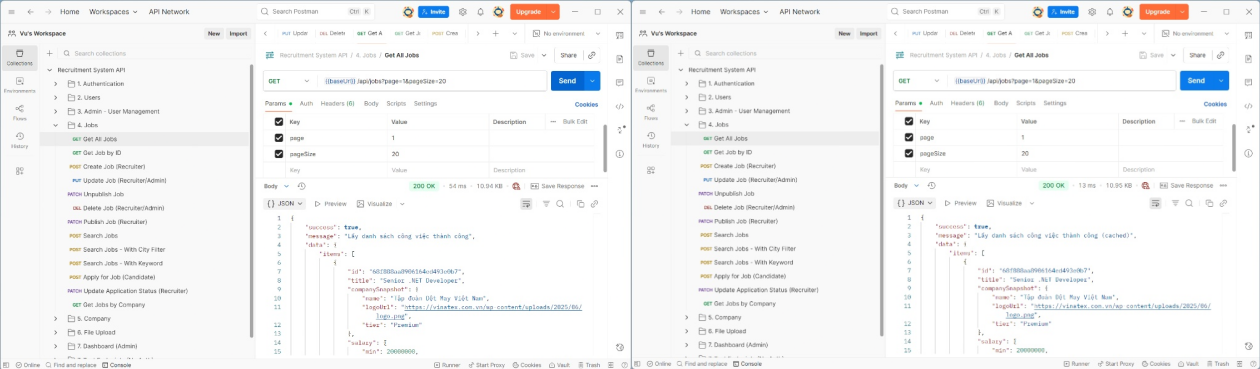
Dữ liệu kiểm thử cục bộ xác nhận rằng việc triển khai Redis cache hoạt động hiệu quả và mang lại lợi ích rõ rệt:

* Đối với endpoint GET /api/jobs, thời gian phản hồi giảm **75.9%** (từ 54ms xuống 13ms) khi dữ liệu được phục vụ từ cache.
* Đối với endpoint GET /api/users/profile, hiệu quả còn lớn hơn, thời gian phản hồi giảm **93.5%** (từ 92ms xuống chỉ còn 6ms).

Những con số này chứng minh rằng lớp đệm (cache) đã thành công trong việc giảm đáng kể độ trễ cho các yêu cầu đọc dữ liệu thường xuyên. Mặc dù đây là các số liệu đo lường đơn lẻ, chưa phản ánh khả năng chịu tải (throughput) tổng thể của hệ thống, chúng đã xác minh được mục tiêu thiết kế: giải phóng tài nguyên cho cơ sở dữ liệu và cải thiện tốc độ phản hồi cho người dùng cuối.



Hình .: Kết quả test trên route “Get profile”



Hình .: Kết quả test trên route “Get All Jobs”

## Cấu hình môi trường và triển khai API

### Thiết lập dự án

Cấu trúc Solution bao gồm ba dự án chính: API (Controllers, Middleware, Program.cs), Core (Models, DTOs, Interfaces, Validators), và Infrastructure (MongoDbContext, Repositories, Services, Settings). Cấu trúc này tuân thủ nguyên tắc Separation of Concerns và Dependency Inversion.

Các gói NuGet (package) chính được sử dụng bao gồm: MongoDB.Driver, StackExchange.Redis, Swashbuckle.AspNetCore (Swagger), Microsoft.AspNetCore.Authentication.JwtBearer, AutoMapper, và System.IdentityModel.Tokens.Jwt.

### appsettings và Dependency Injection

Tệp appsettings.json chuẩn hóa các tham số cấu hình, bao gồm Logging, DatabaseSettings (MongoDB), RedisSettings, JwtSettings, FileUploadSettings và Cors.

Cơ chế Dependency Injection (DI) trong Program.cs thực hiện các nhiệm vụ:

1. Nạp (bind) các lớp Settings từ cấu hình.
2. Đăng ký các dịch vụ (ví dụ: MongoDbContext, RedisCacheService - Singleton; IRepository, IBusinessService - Scoped).
3. Cấu hình AutoMapper, Authentication/Authorization (JWT), CORS, Controllers và Swagger (bao gồm định nghĩa bảo mật).



Hình .: appsettings.json



Hình .: Program.cs

### Các nhóm API chính

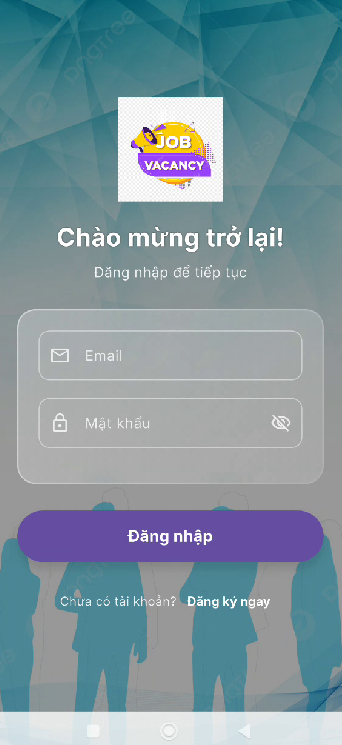
* Nhóm Auth API: Cung cấp các endpoint register, login, refresh, logout, me để quản lý xác thực và token.
* Nhóm Users API: Hỗ trợ quản lý hồ sơ người dùng và các thao tác quản trị (danh sách, vai trò, vòng đời tài khoản).
* Nhóm Jobs API: Trung tâm của hệ thống, bao gồm danh sách (phân trang, lọc theo vai trò, cache), chi tiết, tạo/sửa/xóa mềm, publish/unpublish, tìm kiếm, và quy trình ứng tuyển.
* Nhóm Company API: Xử lý đăng ký, duyệt và cập nhật thông tin công ty.
* Nhóm Upload API: Hỗ trợ tải lên CV/ảnh, kiểm tra kích thước/định dạng tệp.
* Nhóm Dashboard API: Cung cấp các số liệu thống kê tổng hợp.

Hệ thống áp dụng định dạng phản hồi nhất quán (wrapper object chứa data, message, statusCode) và một Global Exception Handler để chuẩn hóa lỗi, tránh rò rỉ thông tin chi tiết của hệ thống.

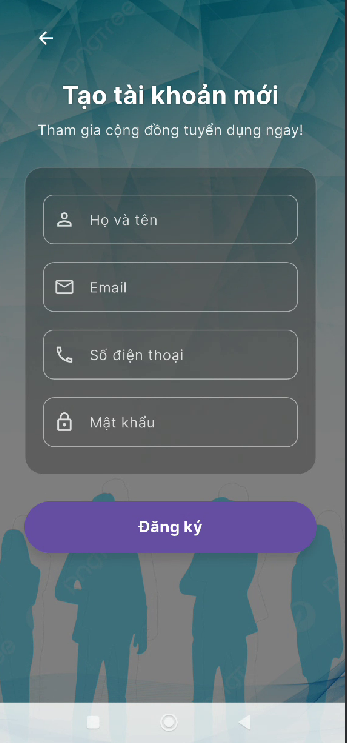
## Giao diện triển khai

Mục này mô tả tổng quan các màn hình chính và cách thức chúng tương tác với API. Hình ảnh minh họa và chú thích chi tiết được trình bày trong Phụ lục. Mô tả tập trung vào chức năng và luồng tương tác để hỗ trợ việc đối chiếu với hành vi của backend, giữ trọng tâm kỹ thuật của chương.

### Login/Register



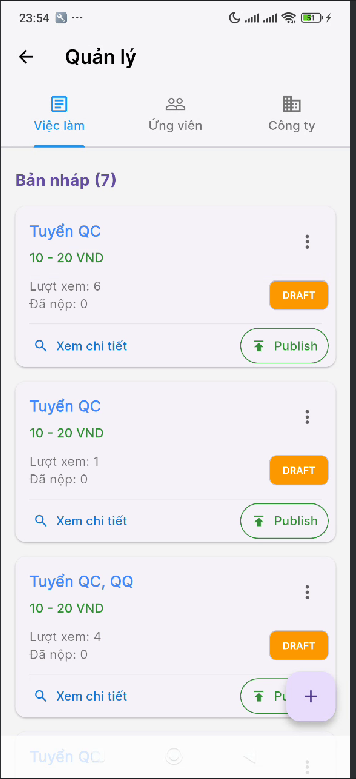
Hình 3.12: Màn hình Đăng Nhập



Hình 3.13: Màn hình Đăng Ký

### Dashboard Admin/Recruiter

Giao diện tổng quan thống kê cho Admin và giao diện quản lý tin tuyển dụng/hồ sơ ứng tuyển cho Recruiter.

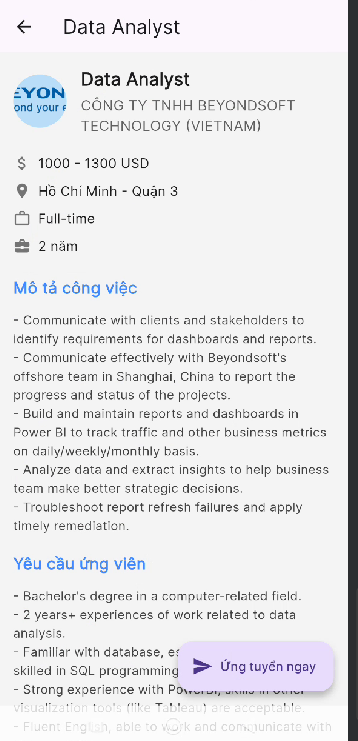


Hình 3.14: Màn hình Dashboard của Recruiter

### Job Listing/Detail



Hình 3.15: Màn hình danh sách việc làm

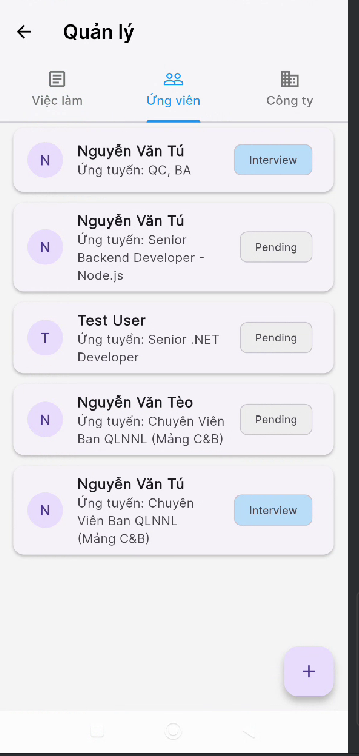
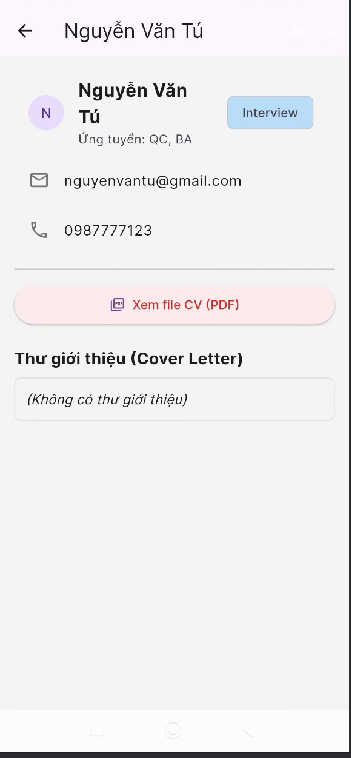


Hình 3.16: Màn hình chi tiết công việc

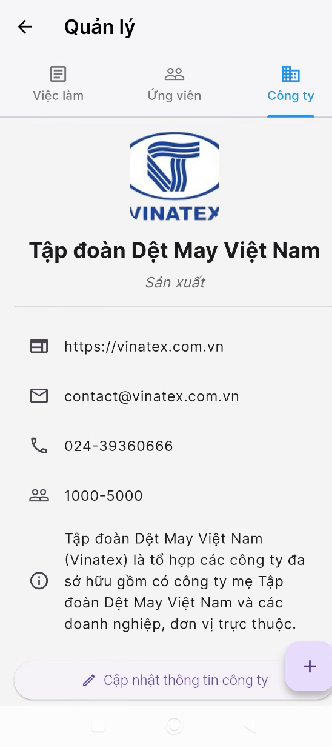


Hình 3.17: Màn hình nộp hồ sơ ứng tuyển

### Company Management/Candidate Profile

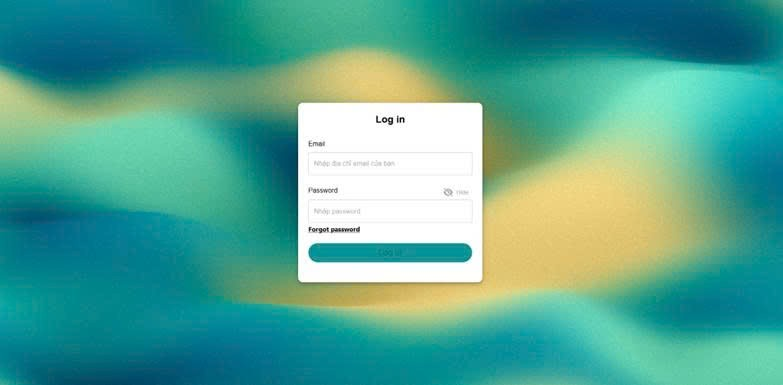
Hình 3.18: Màn hình quản lý hồ sơ cá nhân, cv của candidate



Hình 3.19: Màn hình quản lý thông tin công ty

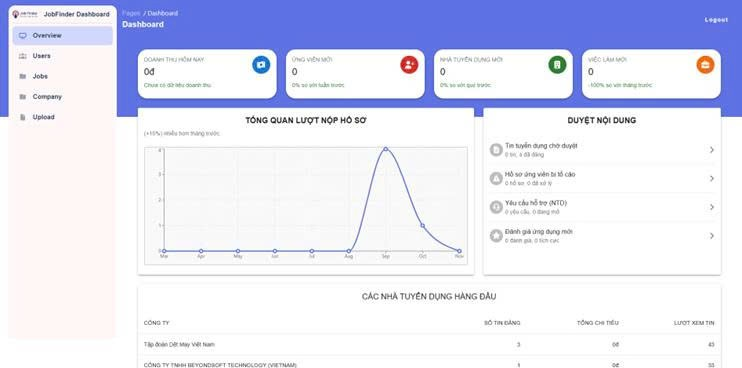
### Admin

#### Giao diện login



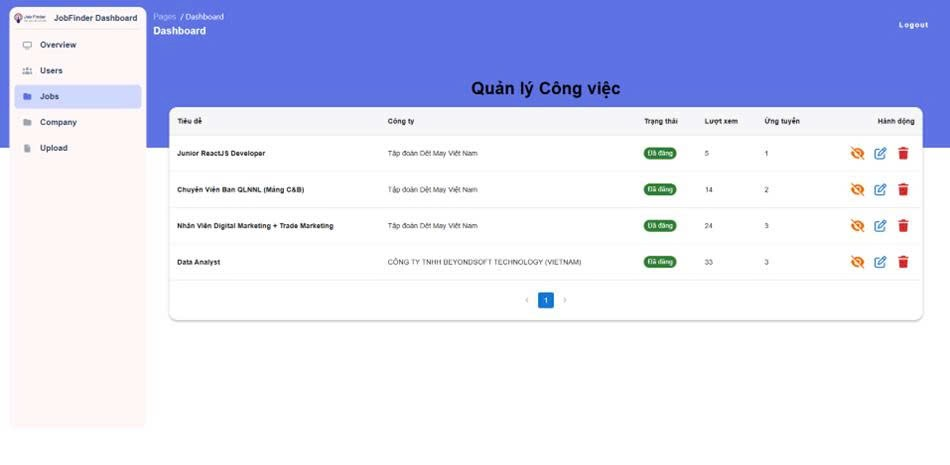
Hình .: Giao diện đăng nhập - Admin

#### Giao diện dashboard



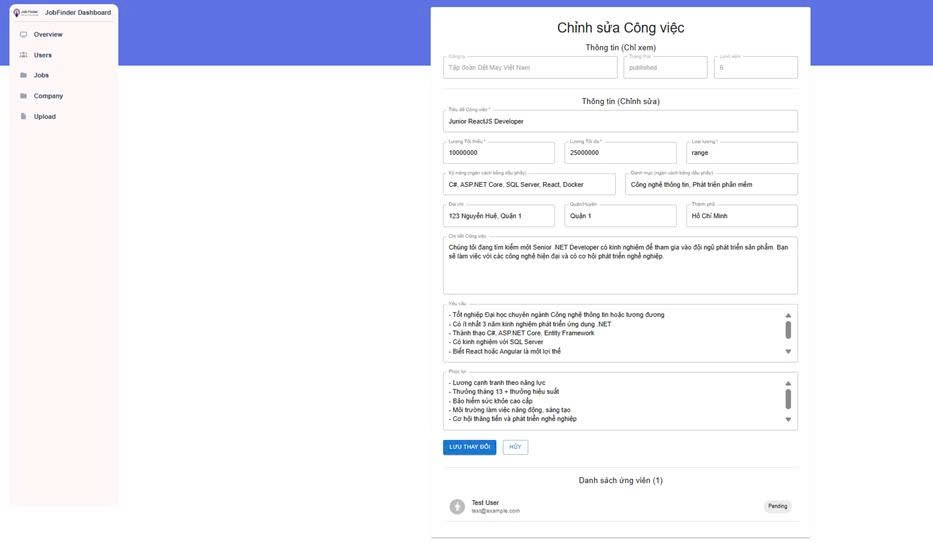
Hình .: Giao diện dashboard - Admin

#### Chức năng chức năng quản lý công việc



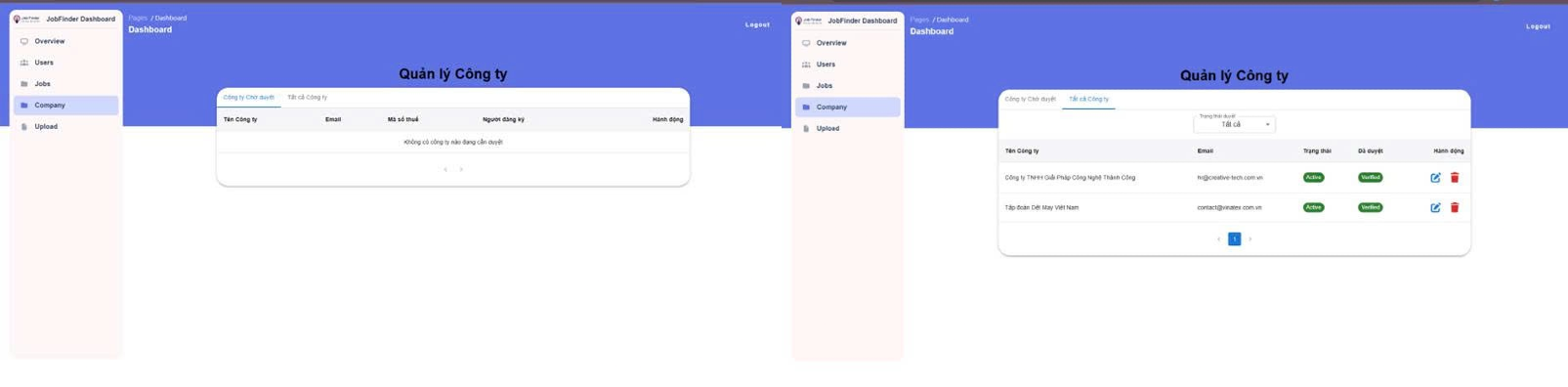
Hình .: Giao diện Quản lý công việc - Admin

#### Giao diện chức năng chi tiết công việc



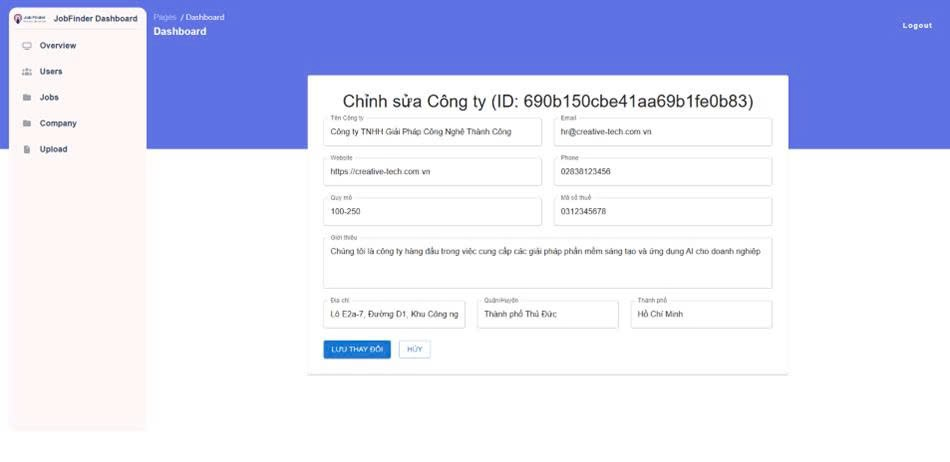
Hình .: Giao diện chi tiết công việc - Admin

#### Giao diện chức năng quản lý công ty



Hình .: Giao diện quản lý công ty - Admin

#### Giao diện chức năng chỉnh sửa thông tin công ty



Hình .: Giao diện thông tin công ty - Admin

## Kết luận chương

Chương 3 đã trình bày toàn diện quá trình triển khai hệ thống quản lý tuyển dụng, từ kiến trúc tổng thể đến các chi tiết kỹ thuật. Hệ thống được xây dựng dựa trên kiến trúc ba tầng và các nguyên tắc Clean Architecture, đảm bảo tính tách biệt và khả năng bảo trì.

MongoDB được triển khai làm cơ sở dữ liệu chính, tận dụng mô hình tài liệu linh hoạt và cơ chế nhúng (embedded documents) để tối ưu hóa truy vấn. Repository Pattern được áp dụng kết hợp với các chiến lược phân trang và lọc dữ liệu theo vai trò.

Cơ chế đệm Redis đóng vai trò then chốt trong việc tối ưu hiệu năng thông qua chiến lược cache-aside và TTL 5 phút, mang lại cải thiện đáng kể về thời gian phản hồi và giảm tải cho MongoDB. Cơ chế vô hiệu hóa cache theo phạm vi (scope-based invalidation) được thiết kế để đảm bảo tính nhất quán dữ liệu.

Kiến trúc Dependency Injection được quản lý chặt chẽ, và các nhóm API được tổ chức logic. Postman Collection được cung cấp như một công cụ kiểm thử tự động. Để hoàn thiện báo cáo, các tài liệu bổ sung (sơ đồ, code snippet, số đo benchmark) cần được đưa vào các mục đã đánh dấu trong chương.

# KẾT LUẬN

TÀI LIỆU THAM KHẢO

1. Redis Commands Documentation: <https://redis.io/docs/latest/commands/>.
2. Redis Official Website: <https://redis.io/>.
3. Contabo Blog: Everything about Redis:<https://contabo.com/blog/everything-about-redis/>.
4. Codersee: Redis Database Explained: <https://codersee.com/redis-database-explained/>.