**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

**A logo with a ball and a sphere

Description automatically generated**-----🙞🙞🙞🙞🙞-----

BÁO CÁO CUỐI KỲ MÔN BIG DATA ESSENTIALS

**ĐỀ TÀI:**

**Tìm hiểu CSDL NoSQL. Tìm hiểu và cài đặt Neo4j. Cho ví dụ minh họa thiết kế CSDL, thao tác thêm, xóa, sửa, truy vấn…**

**GVHD:** Lê Thị Minh Châu

**Lớp:** Thứ 2 (tiết 3-6)

**Mã lớp:** BDES333877\_23\_1\_01

Sinh viên thực hiện: (nhóm 1)

Đỗ Ngọc Chí Công MSSV: 21133013

Nguyễn Trọng Dũng MSSV: 21133021

          Huỳnh Gia Hân         MSSV: 21133031

Trần Thị Ngọc Trang MSSV: 21133109

*Thành phố Hồ Chí Minh, tháng 12, năm 2023*

|  |  |
| --- | --- |
| BỘ GIÁO DỤC VÀ ĐÀO TẠO  **TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT**  **THÀNH PHỐ HỒ CHÍ MINH** | **CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM**  **Độc lập – Tự do – Hạnh phúc** |

**NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN**

**Họ và tên sinh viên thực hiện**

* Đỗ Ngọc Chí Công - 21133013
* Nguyễn Trọng Dũng - 21133021
* Huỳnh Gia Hân - 21133031
* Trần Thị Ngọc Trang - 21133109

**Chuyên ngành:** Kỹ thuật dữ liệu (Data Engineering)

**Đề tài:** Tìm hiểu CSDL NoSQL. Tìm hiểu và cài đặt Neo4j. Cho ví dụ minh hoạ thiết kế CSDL, thao tác thêm, xoá, sửa, truy vấn….

**Môn học:** Nhập môn dữ liệu lớn (Big Data Essentials)

**Nhận xét**

Tp HCM, / / 2023

Giảng viên hướng dẫn

(Tên và chữ ký)

Lê Thị Minh Châu

|  |  |
| --- | --- |
| BỘ GIÁO DỤC VÀ ĐÀO TẠO  **TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT**  **THÀNH PHỐ HỒ CHÍ MINH** | **CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM**  **Độc lập – Tự do – Hạnh phúc** |

**BẢNG PHÂN CÔNG CÔNG VIỆC**

|  |  |  |
| --- | --- | --- |
| **Tên** | **MSSV** | **Phân công** |
| Đỗ Ngọc Chí Công | 21133013 | - Tìm hiểu NoSQl, Neo4j tổng quan, tổng quan Cypher Query Language.  - Cypher Query Language (FORSEARCH, FUNCTION, UNIQUE CONSTRAINT)  - Xây dựng giao diện kết nối với Neo4j sử dụng framework FastAPI minh hoạ:  + Tìm thông tin Movies theo từ khóa và xem được số lượt đánh giá của nó.  + Tìm bộ phim theo Id của phim đó.  - Viết báo cáo. |
| Nguyễn Trọng Dũng | 21133021 | - Tìm hiểu NoSQl, Neo4j Tổng quan.  - Cypher Query Language (MERGE, DELETE, REMOVE)  - Xây dựng giao diện kết nối với Neo4j sử dụng framework FastAPI minh hoạ:  + Xóa Rating của một User từ một bộ phim.  + Xóa một User dựa vào Id của User đó.  - Viết báo cáo. |
| Huỳnh Gia Hân | 21133031 | - Tìm hiểu NoSQl, Neo4j Tổng quan.  - Cypher Query Language (CREATE, WITH, SET)  - Xây dựng giao diện kết nối với Neo4j sử dụng framework FastAPI minh hoạ:  + Tạo thông tin một User mới.  + Chỉnh sửa thông tin name của User.  + Tạo đánh giá cho một bộ phim từ một User.  - Viết báo cáo |
| Trần Thị Ngọc Trang | 21133109 | - Tìm hiểu NoSQl, Neo4j Tổng quan, Cypher Query Language (Match, Return, Where)  - Xây dựng giao diện kết nối với Neo4j sử dụng framework FastAPI minh hoạ:  + Tìm Danh Sách Top10 phim sắp xếp theo Doanh Thu giảm dần.  + Đưa ra đề xuất một bộ phim mà người dùng chưa xem.  - Viết báo cáo |

**LỜI CẢM ƠN**

Bài báo cáo này là sản phẩm của một quá trình học tập và làm việc nhóm. Để có thể hoàn thành bài báo cáo, chúng em đã nhận được rất nhiều sự hỗ trợ từ cô và các bạn. Do đó nhóm chúng em xin được phép gửi lời cảm ơn chân thành và sâu sắc nhất đến:

*1) Cô Lê Thị Minh Châu – giảng viên bộ môn Nhập môn Dữ liệu lớn (Big Data Essentials), khoa Công nghệ Thông tin trường Đại học Sư phạm Kỹ thuật thành phố Hồ Chí Minh. Nhóm xin cảm ơn sự quan tâm và giúp đỡ tận tình của cô trong suốt quá trình giảng dạy. Cảm ơn cô đã luôn giải đáp những thắc mắc cũng như đưa ra những nhận xét, góp ý giúp nhóm thực hiện cải thiện chất lượng công việc của nhóm.*

*2) Thư viện trường Đại học Sư phạm Kỹ thuật thành phố Hồ Chí Minh, nơi cung cấp môi trường học tập, nghiên cứu và làm việc nhóm để chúng em có thể hoàn thành tốt báo cáo của nhóm mình.*

*3) Các bạn học cùng lớp đã có những nhận xét, đóng góp về mặt kiến thức lẫn tinh thần cho nhóm.*

Đề tài và bài báo cáo được chúng em thực hiện trong khoảng thời gian ngắn, với những hạn chế khác về mặt kiến thức, kỹ thuật và kinh nghiệm trong việc thực hiện báo cáo. Do đó, trong quá trình làm đề tài có những thiếu sót là điều không thể tránh khỏi. Chúng em rất mong nhận được những ý kiến đóng góp quý báu của cô để kiến thức của chúng em được hoàn thiện hơn và làm tốt hơn nữa trong những lần sau. Chúng em xin chân thành cảm ơn!

**MỤC LỤC**

[**PHẦN 1. MỞ ĐẦU** 1](#_Toc153317644)

[**PHẦN 2. NỘI DUNG** 2](#_Toc153317645)

[**CHƯƠNG 1. GIỚI THIỆU NOSQL** 2](#_Toc153317646)

[**1.1. Khái niệm NoSQL** 2](#_Toc153317647)

[**1.2. Đặc điểm nổi bật của NoSQL** 2](#_Toc153317648)

[**1.3. Các loại NoSQL phổ biến** 3](#_Toc153317649)

[**1.4 Sự khác nhau giữa RDBMs và NoSQL** 4](#_Toc153317650)

[**CHƯƠNG 2. NEO4J - TỔNG QUAN** 6](#_Toc153317651)

[**2.1. Khái niệm về Neo4j** 6](#_Toc153317652)

[**2.2. Ưu điểm của Neo4j** 6](#_Toc153317653)

[**2.3. Những đặc điểm của Neo4j** 7](#_Toc153317654)

[2.3.1. Lược đồ linh hoạt (Flexible Schema) 7](#_Toc153317655)

[2.3.2. Thuộc tính ACID 8](#_Toc153317656)

[2.3.3. Khả năng mở rộng và tính sẵn sàng 8](#_Toc153317657)

[2.3.4. Ngôn ngữ truy vấn Cypher 8](#_Toc153317658)

[2.3.5. Ứng dụng xây dựng trang web 9](#_Toc153317659)

[2.3.6. Drives. 9](#_Toc153317660)

[**2.4. Graph Data Modeling trong Neo4j** 9](#_Toc153317661)

[2.4.1. Node 10](#_Toc153317662)

[2.4.2. Labels 10](#_Toc153317663)

[2.4.3. Relationships 10](#_Toc153317664)

[2.4.4. Relationship Types 11](#_Toc153317665)

[2.4.5. Properties 11](#_Toc153317666)

[2.4.6. Traversals 12](#_Toc153317667)

[2.4.7. Paths 12](#_Toc153317668)

[**2.5. Neo4j - Thiết lập môi trường** 12](#_Toc153317669)

[2.5.1. Cài đặt Neo4j Desktop trên Windows 12](#_Toc153317670)

[2.5.2. Cài đặt Neo4j Desktop trên Linux 17](#_Toc153317671)

[**2.6. Làm việc với Neo4j** 20](#_Toc153317672)

[2.6.1. Neo4j Desktop 20](#_Toc153317673)

[2.6.2. Neo4j Browser 21](#_Toc153317674)

[**2.7. Tạo Local DBMS trong Neo4j Desktop và Import File** 22](#_Toc153317675)

[2.7.1. Tạo Project và Local DBMS 22](#_Toc153317676)

[2.7.2. Import dump file 23](#_Toc153317677)

[2.7.3. Import csv file 27](#_Toc153317678)

[**CHƯƠNG 3. Neo4j – CYPHER QUERY LANGUAGE** 31](#_Toc153317679)

[**3.1. Neo4j - Cypher Query Language** 31](#_Toc153317680)

[**3.2. MATCH** 31](#_Toc153317681)

[**3.3. RETURN** 33](#_Toc153317682)

[**3.4. WHERE** 40](#_Toc153317683)

[**3.5. CREATE** 42](#_Toc153317684)

[**3.6. WITH** 45](#_Toc153317685)

[**3.7. SET** 47](#_Toc153317686)

[**3.8. MERGE** 51](#_Toc153317687)

[**3.9. DELETE** 54](#_Toc153317688)

[**3.10. REMOVE** 56](#_Toc153317689)

[**3.11. FOREACH** 59](#_Toc153317690)

[**3.12. FUNCTION** 60](#_Toc153317691)

[3.12.1. Aggregating Functions (Các hàm tổng hợp) 60](#_Toc153317692)

[3.12.2. String functions (Các hàm Chuỗi) 63](#_Toc153317693)

[3.12.3. Temporal functions (Các hàm Thời gian) 66](#_Toc153317694)

[**3.13. UNIQUE CONSTRAINT** 67](#_Toc153317695)

[**CHƯƠNG 4. XÂY DỰNG ỨNG DỤNG KẾT NỐI NEO4J** 70](#_Toc153317696)

[**4.1. Mô tả dữ liệu sử dụng trên Neo4j** 70](#_Toc153317697)

[4.1.1. Mô tả dữ liệu 70](#_Toc153317698)

[4.1.2. Kết nối Cơ Sở Dữ Liệu 70](#_Toc153317699)

[**4.2. Xây dựng ứng dụng kết nối với Neo4j** 71](#_Toc153317700)

[4.2.1. Khai phá dữ liệu bằng đồ thị 71](#_Toc153317701)

[4.2.2. Mô tả ứng dụng 74](#_Toc153317702)

[4.2.3. Xây dựng API để tương tác 75](#_Toc153317703)

[**PHẦN 3: KẾT LUẬN** 93](#_Toc153317704)

[**TÀI LIỆU THAM KHẢO** 94](#_Toc153317705)

# **PHẦN 1. MỞ ĐẦU**

**1. Mục đích của đề tài**

“Cơ sở dữ liệu” là thuật ngữ đã trở nên rất quen thuộc với các nhà phát triển ứng dụng, có rất nhiều dạng cơ sở dữ liệu như: Cơ sở dữ liệu Quan hệ, cơ sở dữ liệu Đối tượng, cơ sở dữ liệu Time-Series và nhiều loại khác. Một trong những dạng cơ sở dữ liệu thú vị và phổ biến hiện nay là cơ sở dữ liệu Đồ thị - Neo4j, một hệ quản lý cơ sở dữ liệu đồ thị mã nguồn mở.

Mục đích chính của đề tài này là tìm hiểu, phân tích, thực hiện các thao tác cơ bản và áp dụng cơ sở dữ liệu Đồ thị - Neo4j trong ngữ cảnh các ứng dụng và dự án thực tế. Bằng cách này, chúng ta sẽ khám phá sức mạnh của mô hình đồ thị trong việc mô phỏng và tương tác với các mối quan hệ phức tạp giữa dữ liệu.

**2. Đối tượng và phạm vi nghiên cứu**

Đối tượng nghiên cứu của đồ án là tìm hiểu, phân tích và sử dụng CSDL NoSQL, Neo4j.

Phạm vi nghiên cứu:

* Tìm hiểu về cơ sở dữ liệu đồ thị và Neo4j.
* Thiết kế CSDL minh họa với Neo4j.
* Thực hiện các thao tác thêm, xóa, sửa và truy vấn trên CSDL đồ thị.

**3. Kết quả dự kiến đạt được**

Hiểu rõ về cơ sở dữ liệu đồ thị và cách nó hoạt động.

Thiết kế và triển khai một cơ sở dữ liệu đồ thị sử dụng Neo4j.

Thực hiện thành công các thao tác thêm, xóa, sửa và truy vấn trên CSDL Neo4j.

# **PHẦN 2. NỘI DUNG**

# **CHƯƠNG 1. GIỚI THIỆU NOSQL**

## **1.1. Khái niệm NoSQL**

NoSQL là Non\_Relational: tạm dịch là không có quan hệ. NoSQL là cơ sở dữ liệu không quan hệ, ràng buộc giữa các Collection (hay được gọi là bảng trong cơ sở dữ liệu bình thường). Có nghĩa là giữa các Collection sẽ không có khóa ngoại hay khóa chính như trong cơ sở dữ liệu bình thường. Hạn chế bớt rối khi cơ sở dữ liệu nhiều bảng. NoSQL đặc biệt nhấn mạnh đến mô hình lưu trữ các cặp giá trị - khóa và hệ thống phân tán.

Cơ sở dữ liệu NoSQL sử dụng nhiều mô hình dữ liệu để truy cập và quản lý dữ liệu. Các loại cơ sở dữ liệu này được tối ưu hóa dành riêng cho các ứng dụng yêu cầu mô hình dữ liệu linh hoạt có lượng dữ liệu lớn và độ trễ thấp, có thể đạt được bằng cách giảm bớt một số hạn chế về tính nhất quán của dữ liệu của các cơ sở dữ liệu khác.

## **1.2. Đặc điểm nổi bật của NoSQL**

* Khả năng mở rộng (Scalability): NoSQL lưu trữ dữ liệu của mình theo dạng cặp giá trị “key – value”. Sử dụng số lượng lớn các node để lưu trữ thông tin. Gần như không có giới hạn cho dữ liệu và người dùng hệ thống.
* Tính sẵn sàng (High Availability): Vì chấp nhận sự trùng lặp trong lưu trữ dữ liệu nên nếu một node bị chết sẽ không ảnh hưởng tới toàn bộ hệ thống.
* Tính nguyên tố (Atomicity): Độc lập trạng thái dữ liệu trong các thao tác.
* Tính nhất quán (Consistency): Chấp nhận tính nhất quán yếu, cập nhật mới không đảm bảo các truy xuất sau đó thấy được sự thay đổi. Sau một khoảng thời gian lan truyền thì tính nhất quán cuối cùng của dữ liệu mới được đảm bảo.
* Tính bền vững (Durability): Dữ liệu có thể tồn tại trong bộ nhớ máy tính, nhưng cũng đồng thời được lưu trữ tại đĩa cứng.
* Triển khai linh hoạt (Deployment Flexibility): Hệ thống sẽ tự động nhận biết việc bổ sung thêm hay loại bỏ các node. Hệ thống không đòi hỏi cấu hình phần cứng mạnh, đồng nhất.
* Mô hình hóa linh hoạt (Modeling flexibility): cặp dữ liệu key-value, dữ liệu cấu trúc (Hierarchical data), Graphs.
* Truy vấn linh hoạt (Query Flexibility): Multi-Gets, load một tập giá trị dựa vào tập khóa (Range queries)
* Khả năng mở rộng theo chiều ngang (Horizontal scalable): Bình thường, với các hệ quản trị cơ sở dữ liệu quan hệ, khi mà dữ liệu quá lớn phương pháp tăng khả năng lưu trữ là sẽ phải mở rộng (nâng cấp máy chủ), còn đối với NoSQL thì chỉ cần bổ sung thêm máy chủ khác vì hệ thống hỗ trợ lưu trữ phân tán trên nhiều máy.

## **1.3. Các loại NoSQL phổ biến**

Với NoSQL, dữ liệu có thể được lưu trữ theo kiểu không có lược đồ hoặc dạng tự do. Dữ liệu bất kỳ có thể được lưu trữ trong bản ghi bất kỳ. Trong số các cơ sở dữ liệu NoSQL, có 4 mô hình lưu trữ dữ liệu phổ biến, do đó, có 4 loại hệ thống NoSQL phổ biến là Document database, Key-value stores, Wide column stores và Graph database.

**1. Document database** (ví dụ: CouchDB, MongoDB): Dữ liệu được thêm vào lưu trữ dưới dạng cấu trúc JSON tự do hoặc “tài liệu”, trong đó dữ liệu có thể là bất kỳ kiểu nào, từ số nguyên đến chuỗi hay đến các văn bản tự do.

* *Ưu điểm*: Dùng khi dữ liệu nguồn không được mô tả đầy đủ.
* *Nhược điểm*: Hiệu năng truy vấn, Không có cú pháp chuẩn cho câu truy vấn dữ liệu.

**2. Key-value stores** (ví dụ: Redis, Riak): Các giá trị dạng tự do, từ số nguyên hoặc chuỗi đơn giản đến các tài liệu JSON phức tạp, được truy cập trong cơ sở dữ liệu bằng các khóa.

* *Ưu điểm*: Tìm kiếm rất nhanh.
* *Nhược điểm*: Lưu dữ liệu không theo khuôn dạng (schema) nhất định.

**3. Wide column stores** (ví dụ: HBase, Cassandra): Dữ liệu được lưu trữ trong các cột thay vì các hàng như trong một hệ thống SQL thông thường. Bất kỳ số lượng cột nào (và do đó nhiều loại dữ liệu khác nhau) có thể được nhóm hoặc tổng hợp khi cần cho truy vấn hoặc chế độ xem dữ liệu.

* *Ưu điểm*: Tìm kiếm nhanh, Phân tán dữ liệu tốt.
* *Nhược điểm*: Hỗ trợ được với rất ít phần mềm.

**4. Graph database** (ví dụ: Neo4j): Dữ liệu được biểu diễn dưới dạng mạng hoặc đồ thị của các thực thể và các mối quan hệ của thực thể đó, với mỗi node trong biểu đồ là một khối dữ liệu ở dạng tự do.

* *Ưu điểm:* Ứng dụng các thuật toán trên đồ thị như Đường đi ngắn nhất, liên thông, …
* *Nhược điểm:* Phải duyệt nội bộ đồ thị, để trả lời lại các truy vấn. Không dễ để phân tán.

## **1.4 Sự khác nhau giữa RDBMs và NoSQL**

|  |  |  |
| --- | --- | --- |
|  | **RDBMs** | **NoSQL** |
| **Cấu trúc** | Cấu trúc dựa trên các bảng | Cấu trúc phân làm 4 loại chính:   * Document base * Key-value pair base * Wide-column Store * Graph database |
| **Lược đồ** | Lược đồ được định nghĩa để lưu trữ dữ liệu có cấu trúc | Không có một định nghĩa trước nào cho lược đồ, mà lược đồ linh hoạt dựa theo thành phần dữ liệu. |
| **Khả năng mở rộng** | Được mở rộng theo chiều dọc, nếu chúng ta muốn mở rộng cơ sở dữ liệu thì phải nâng cấp phần cứng, điều này làm hạn chế khả năng mở rộng của CSDL quan hệ. | Mở rộng theo chiều ngang, nghĩa là nếu muốn mở rộng cơ sở dữ liệu, chúng ta chỉ cần thêm các node và tạo ra mạng lưới phân phối dựa trên yêu cầu mà ta đưa ra, đây là cách giảm tải trên cơ sở dữ liệu quan hệ. |
| **Các hệ cơ sở dữ liệu tiêu biểu** | * Oracle * MySql * Postgres * MS-SQL | * MongoDB * BigTable * Redis * RavenDb * Cassendra * Hbase * Neo4j * CouchDb |
| **Phân loại** | Phân làm hai loại chính là CSDL quan hệ mã nguồn mở và CSDL mã nguồn đóng. | Phân chia thành 5 loại dựa trên cách lưu trữ data: ∙ Key-value pair store database ∙ Graph database ∙ Document Store ∙ Column Store ∙ XML Store |
| **Khái niệm cơ bản** | Dựa trên những quy luật và tính ACID (Atomic (Tính nguyên tố), Consistent (Tính nhất quán), Isolated (Tính cô lập), Durable (Tính bền vững)) | Dựa trên 3 tính chất: nhất quán, sẵn sàng và phân tán. |

# **CHƯƠNG 2. NEO4J - TỔNG QUAN**

## **2.1. Khái niệm về Neo4j**

Cơ sở dữ liệu đồ thị Neo4j lưu trữ các nút và mối quan hệ thay vì bảng hoặc tài liệu. Dữ liệu được lưu trữ giống việc phác thảo ý tưởng trên bảng trắng. Dữ liệu được lưu trữ mà không giới hạn trong một mô hình được xác định trước, cho phép người phân tích suy nghĩ và sử dụng mô hình một cách rất linh hoạt.

## **2.2. Ưu điểm của Neo4j**

* Khả năng đọc và viết dữ liệu có hiệu suất cao mà không gây ra vấn đề nghiêm trọng cụ thể Neo4j thực hiện việc đọc và viết nhanh như chớp, trong khi đó vẫn bảo vệ được tính nhất quán của dữ liệu.
* Dễ dàng để học và dễ sử dụng: Neo4j có một giao diện thân thiện, ngôn ngữ truy vấn dễ học và nguồn tài nguyên học dồi dào.
* Dễ dàng để chạy dữ liệu vào bên trong Neo4j cụ thể là chạy dữ liệu có kích thước lớn với bộ nhớ thấp. Và có thể lựa chọn loại và kích cỡ dữ liệu để đưa vào mà không cần lo lắng về khối lượng lưu trữ.
* Kết nối mọi thứ thành đồ thị, hay hỗ trợ cho nhu về việc sử dụng cơ sở dữ liệu đồ thị với mục đích là xử lý các thông tin được kết nối với nhau một cách hiệu quả.
* Neo4j tạo ra mô hình dữ liệu mạnh mẽ và linh hoạt: Neo4j cho phép lưu trữ và quản lý dữ liệu trong một cách rất tự nhiên, ánh xạ đến mối quan hệ và những thực thể ở thế giới thực. Điều này cho phép người nghiên cứu nhìn nhận và trình bày mối quan hệ và phân tích dữ liệu một cách hiệu quả.
* Neo4j có một cộng động rộng rãi, nơi mà những nhà phát triển và người dùng chia sẻ kiến thức, hỗ trợ lẫn nhau.
* Truy xuất dễ dàng − Sử dụng Neo4j, biểu diễn và truy xuất dễ dàng (duyệt/điều hướng) dữ liệu được kết nối nhanh hơn khi so sánh với các cơ sở dữ liệu khác.
* Ngôn ngữ truy vấn Cypher − Neo4j cung cấp ngôn ngữ truy vấn khai báo để biểu thị biểu đồ một cách trực quan, những nhà phát triển và những người không rành về công nghệ đều có khả năng sử dụng công cụ một cách dễ dàng và hiệu quả.
* Tính ứng dụng cao, Neo4j hiện nay được sử dụng rộng rãi trong các lĩnh vực như: phân tích mạng lưới xã hội, quản lý chuỗi cung, ứng dụng trong bảo mật, ...

## **2.3. Những đặc điểm của Neo4j**

### 2.3.1. Lược đồ linh hoạt (Flexible Schema)

Neo4j cung cấp cách tiếp cận không cần lược đồ mạnh mẽ và linh hoạt để quản lý dữ liệu. Không giống như cơ sở dữ liệu quan hệ truyền thống có cấu trúc cứng nhắc, Neo4j cho phép lưu trữ và truy vấn dữ liệu dựa trên mối quan hệ giữa các thực thể, mang lại một số lợi ích sau:

Khả năng thích ứng:

* Không có cấu trúc được xác định trước: không cần xác định trước các bảng, cột hoặc mối quan hệ. Mô hình dữ liệu có thể phát triển một cách nhanh chóng khi nhu cầu thay đổi.
* Dễ dàng bổ sung dữ liệu mới: Các thực thể và mối quan hệ mới có thể được thêm liền mạch mà không làm thay đổi lược đồ hiện có.
* Thuộc tính thay đổi động: Thuộc tính nút và mối quan hệ có thể được thêm, xóa hoặc sửa đổi bất kỳ lúc nào.

Nhanh nhẹn:

* Truy vấn nhanh: Việc duyệt qua các mối quan hệ giữa các thực thể có hiệu quả cao trong Neo4j, dẫn đến truy vấn nhanh hơn so với cơ sở dữ liệu quan hệ.
* Mối quan hệ phức tạp: Có thể dễ dàng lập mô hình các mối quan hệ phức tạp giữa các thực thể, bao gồm các mối quan hệ nhiều-nhiều và đệ quy.
* Dữ liệu phi cấu trúc: Neo4j có thể lưu trữ dữ liệu phi cấu trúc như hình ảnh, tài liệu và JSON trực tiếp trong các nút và mối quan hệ.

Tính linh hoạt trong vận hành:

* Khả năng đa cơ sở dữ liệu: Neo4j có thể tích hợp với các cơ sở dữ liệu khác, cho phép tận dụng lợi ích của cả cơ sở dữ liệu quan hệ và đồ thị.
* Đổi mới với lược đồ khác nhau: Có thể thử nghiệm các thiết kế lược đồ biểu đồ khác nhau mà không ảnh hưởng đến dữ liệu hiện có.
* Phát triển AI và máy học linh hoạt: Lược đồ linh hoạt cho phép lặp lại và thử nghiệm nhanh chóng trong các dự án AI và máy học.

### 2.3.2. Thuộc tính ACID

Ý tưởng chính của ACID là cung cấp một môi trường an toàn để vận hành dữ liệu. Từ viết tắt ACID là viết tắt của:

Atomic (Tính nguyên tố): Tất cả các hoạt động trong một giao dịch đều thành công hoặc mọi hoạt động đều được khôi phục.

Consistent (Tính nhất quán):Khi hoàn thành hoặc khôi phục giao dịch, cơ sở dữ liệu ở trạng thái nhất quán; không có cập nhật một phần hoặc lỗi logic.

Isolated (Tính cô lập): Các giao dịch không tranh chấp với nhau. Việc truy cập dữ liệu gây tranh cãi được cơ sở dữ liệu kiểm duyệt để các giao dịch dường như chạy tuần tự.

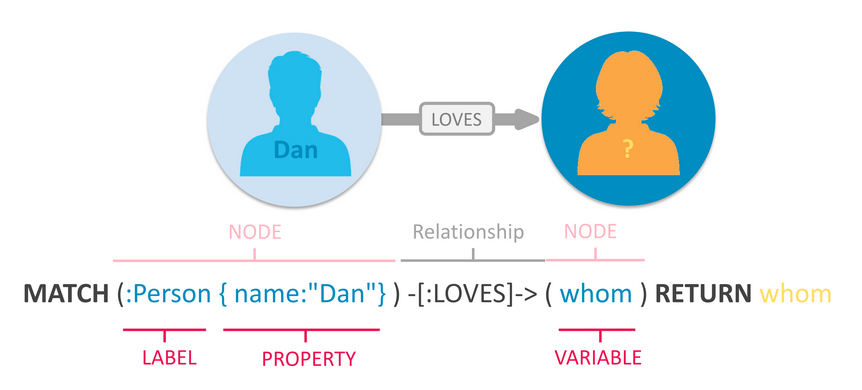
Durable (Tính bền vững):Kết quả của việc áp dụng một giao dịch là vĩnh viễn, ngay cả khi có lỗi.

### 2.3.3. Khả năng mở rộng và tính sẵn sàng

Khả năng mở rộng:Neo4j tạo điều kiện cho việc mở rộng quy mô cơ sở dữ liệu bằng cách tăng số lần đọc/ghi và âm lượng mà không ảnh hưởng đến tính toàn vẹn dữ liệu và tốc độ xử lý truy vấn.

Tính sẵn sàng:Neo4j cung cấp khả năng sao chép để đảm bảo an toàn và độ tin cậy của dữ liệu.

### 2.3.4. Ngôn ngữ truy vấn Cypher



Cypher là ngôn ngữ truy vấn biểu đồ của Neo4j cho phép truy xuất dữ liệu từ biểu đồ. Nó giống như SQL dành cho biểu đồ và được lấy cảm hứng từ SQL nên nó cho phép tập trung vào dữ liệu muốn từ biểu đồ (chứ không phải cách lấy dữ liệu đó). Đây là ngôn ngữ đồ thị dễ học nhất cho đến nay vì tính tương tự và tính trực quan của nó với các ngôn ngữ khác.

### 2.3.5. Ứng dụng xây dựng trang web

* Neo4j cũng cung cấp một ứng dụng web trình duyệt Neo4j tích hợp có thể được sử dụng để tạo và truy xuất dữ liệu biểu đồ.
* Neo4j là một công cụ có giá trị để tương tác với dữ liệu biểu đồ của và khám phá các mối quan hệ của nó. Nó bổ sung cho lược đồ linh hoạt của Neo4j bằng cách cung cấp giao diện thân thiện với người dùng cho nhiều tác vụ khác nhau, bao gồm:
* Kết nối với phiên bản Neo4j: Có thể kết nối với phiên bản Neo4j cục bộ hoặc từ xa bằng trình duyệt.
* Trực quan hóa biểu đồ: Trình duyệt trực quan hóa dữ liệu biểu đồ của bạn, cho phép bạn xem mối quan hệ giữa các nút và khám phá các phần khác nhau của biểu đồ.
* Thực thi các truy vấn Cypher: Trình duyệt cung cấp trình soạn thảo và bảng điều khiển Cypher để viết và thực hiện các truy vấn Cypher nhằm truy xuất và thao tác dữ liệu biểu đồ.
* Quản lý biểu đồ: Có thể tạo, xóa và cập nhật các nút và mối quan hệ bằng giao diện trực quan của trình duyệt.
* Nhập và xuất dữ liệu: Trình duyệt cho phép bạn nhập và xuất dữ liệu biểu đồ của mình ở nhiều định dạng khác nhau, bao gồm CSV, JSON và Cypher.

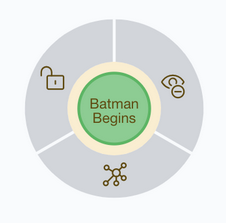
### 2.3.6. Drives.

Neo4j chính thức hỗ trợ trình điều khiển cho .Net, Java, JavaScript, Go và Python và cung cấp trình điều khiển cho tất cả các ngôn ngữ lập trình chính cho tất cả các giao thức và API.

## **2.4. Graph Data Modeling trong Neo4j**

Mô hình hóa dữ liệu đồ thị là quá trình trong đó người dùng mô tả một miền tùy ý dưới dạng biểu đồ được kết nối của các nút và mối quan hệ với các thuộc tính và nhãn. Mô hình dữ liệu đồ thị Neo4j được thiết kế để trả lời các câu hỏi dưới dạng truy vấn Cypher và giải quyết các vấn đề kinh doanh và kỹ thuật bằng cách tổ chức cấu trúc dữ liệu cho cơ sở dữ liệu đồ thị.

### 2.4.1. Node



Một nút đại diện cho một đơn vị dữ liệu cơ bản trong biểu đồ. Nó hoạt động như một thực thể hoặc đối tượng riêng biệt, tương tự như một bản ghi trong bảng cơ sở dữ liệu quan hệ. Tuy nhiên, không giống như các bảng có cột cố định, các nút trong Neo4j linh hoạt và năng động hơn nhiều.

### 2.4.2. Labels

Labels có thể được sử dụng để mô hình hóa miền giá trị của các node lại với nhau, thông thường chúng ta sẽ gộp nhóm các node có cùng kiểu dữ liệu hoặc là thuộc tính thành một tập hợp rồi sau đó gắn label vào cho chúng.

### 2.4.3. Relationships

Các mối quan hệ trong Neo4j Mối quan hệ là khối xây dựng cơ bản của bất kỳ biểu đồ nào trong Neo4j. Chúng đại diện cho các kết nối và tương tác giữa các thực thể khác nhau, được gọi là các nút, trong biểu đồ

Đặc điểm chính:

Có hướng so và không có hướng: Các mối quan hệ có thể có hướng, nghĩa là chúng có một hướng cụ thể (nút nguồn -> nút đích) hoặc không có hướng, biểu thị kết nối qua lại giữa hai nút.

Loại: Tương tự như nhãn nút, các mối quan hệ có thể có các loại phân loại chúng dựa trên bản chất của kết nối mà chúng đại diện.

Ví dụ: Ta có thể có các loại như "FRIENDS WITH", "RATED" hoặc "CONNECTED\_TO". Thuộc tính: Giống như các nút, mối quan hệ có thể chứa các cặp khóa-giá trị được gọi là thuộc tính, lưu trữ thông tin bổ sung về kết nối. Các thuộc tính này có thể nâng cao hơn nữa mô hình dữ liệu của bạn và cung cấp ngữ cảnh cho các mối quan hệ.

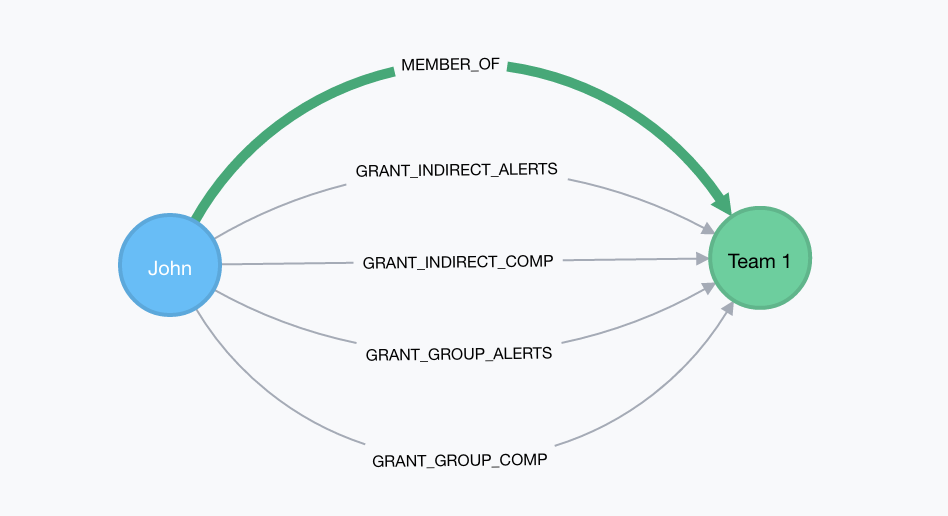
Diagram

Description automatically generated

Biểu diễn mối quan hệ, hay liên kết giữa các node với nhau. Ngoài ra relationship còn có thể cấu trúc phân chia các nodes thành những cấu trúc khác nhau, biến đồ thị thành các dạng cấu trúc giống như là list, tree, map, hoặc có thể là thực thể phức hợp (compound entity). Thực thể phức hợp là thực thể có nhiều liên kết phức tạp liên kết với nhau.

Trong Neo4j, mỗi Relationship là một cạnh có hướng nối Node nguồn và Node đích. Một Node có thể có cạnh nối với chính nó.

### 2.4.4. Relationship Types



Mỗi Relationship chỉ có duy nhất một Relationship Types. Relationship Type giúp ta biết được mối quan hệ giữa các Node là gì. Việc đặt type cho Relationship giúp chúng ta dễ hình dung được Node nào là Node nguồn và Node nào là Node đích.

### 2.4.5. Properties

Thuộc tính trong Neo4j Thuộc tính, còn được gọi là cặp khóa-giá trị, là một khái niệm cơ bản trong Neo4j. Chúng thể hiện các thuộc tính hoặc đặc điểm của cả nút và mối quan hệ trong biểu đồ, cung cấp thông tin và ngữ cảnh cần thiết cho dữ liệu.

### 2.4.6. Traversals

Traversal là cách mà chúng ta truy vấn dữ liệu để trả lời cho một câu hỏi. Travesaling một đồ thị nghĩa là chúng ta bắt đầu từ một Node và lần theo các Relationship (mối quan hệ) theo các quy tắc. Hầu như chúng ta chỉ cần lần theo Đồ thị con của Đồ thị (dựa vào các Label, các Relationship Type).

### 2.4.7. Paths

Trong Neo4j, các đường dẫn biểu thị các chuỗi nút cụ thể và mối quan hệ kết nối hai nút trong biểu đồ. Chúng rất quan trọng để khám phá các mối quan hệ, phân tích tính kết nối và thực hiện các thuật toán đồ thị khác nhau.

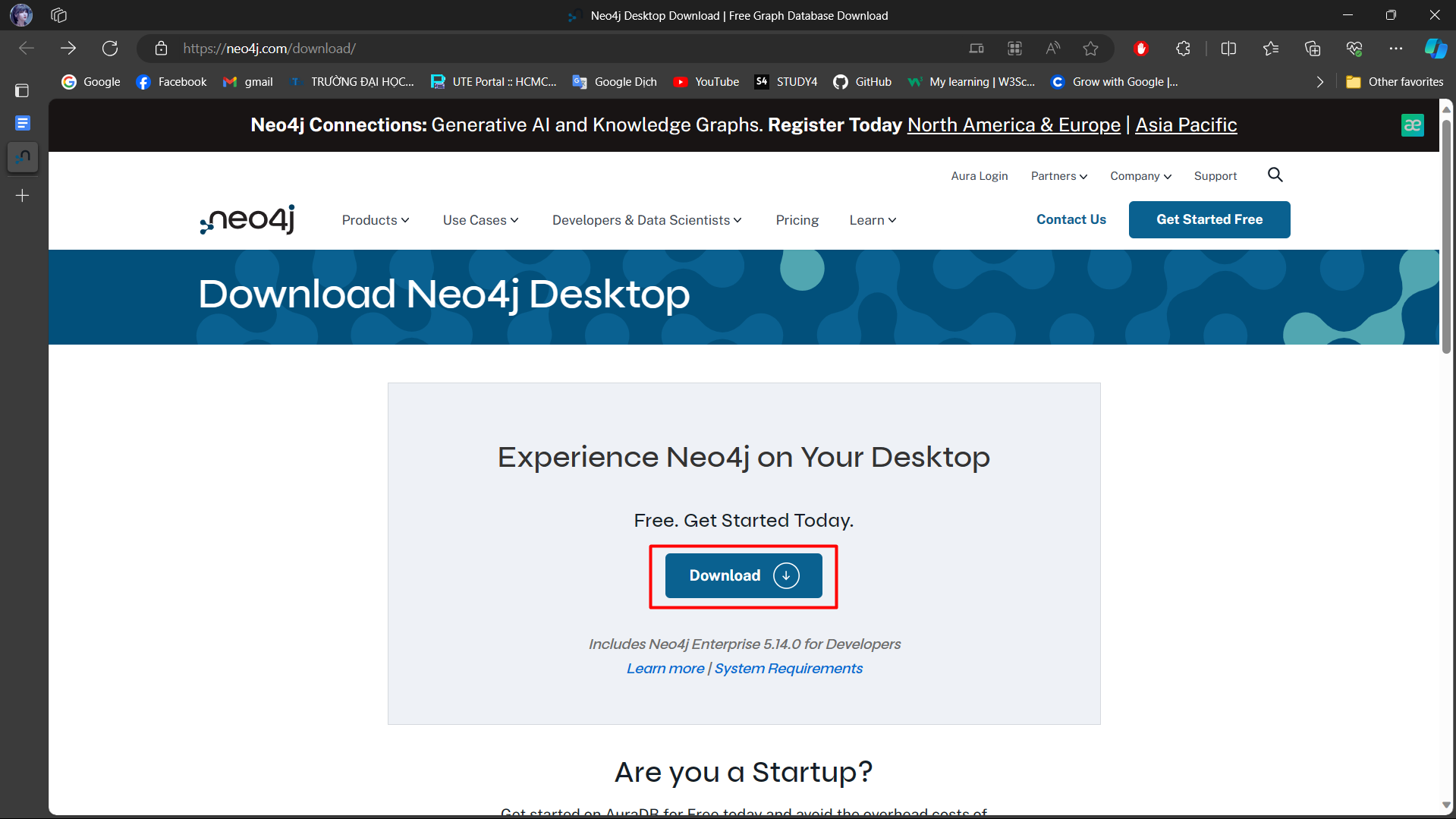
Các loại đường dẫn:

* Đường dẫn đơn giản: Những đường dẫn này bao gồm các nút và mối quan hệ duy nhất, nghĩa là chúng không truy cập cùng một nút hoặc mối quan hệ hai lần.
* Đường dẫn phức tạp: Những đường dẫn này có thể chứa các chu trình hoặc các nút và mối quan hệ lặp lại, tùy thuộc vào thuật toán cụ thể được sử dụng để tìm chúng.
* Đường dẫn ngắn nhất: Những đường dẫn này biểu thị chuỗi nút ngắn nhất và mối quan hệ kết nối hai nút cụ thể, xem xét tính định hướng và trọng số (nếu có). Tất cả các đường dẫn: Những đường dẫn này bao gồm tất cả các chuỗi có thể kết nối hai nút, bất kể độ dài hay độ phức tạp.

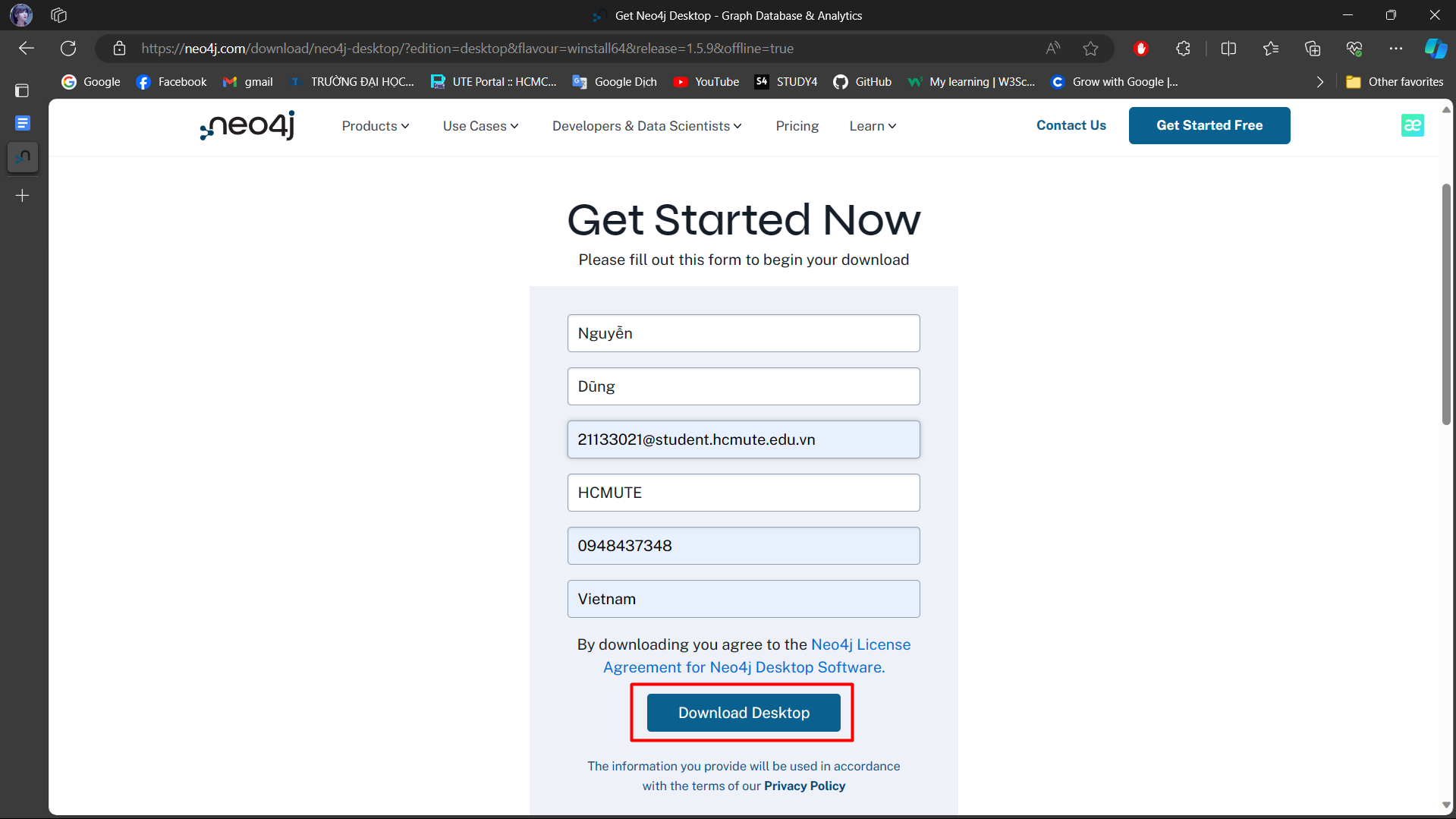
## **2.5. Neo4j - Thiết lập môi trường**

### 2.5.1. Cài đặt Neo4j Desktop trên Windows

Bước 1: Truy cập trang Download chính chủ của Neo4j để tải bản mới nhất: [Neo4j Desktop Download | Free Graph Database Download](https://neo4j.com/download/)

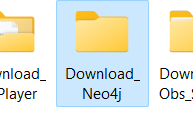


Bước 2: Nhập thông tin mà trang yêu cầu -> Download

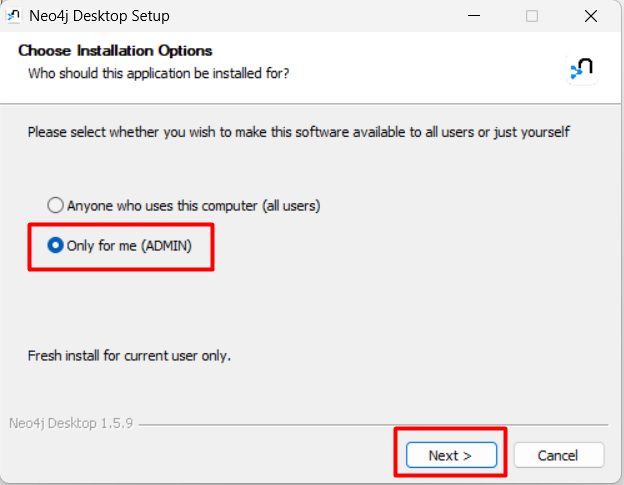


Bước 3: Tiến hành cài đặt

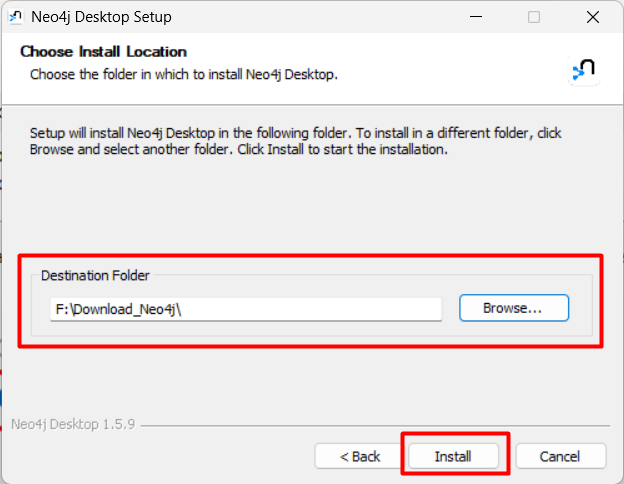
* Tạo trước 1 Folder, ví dụ: Download\_Neo4j



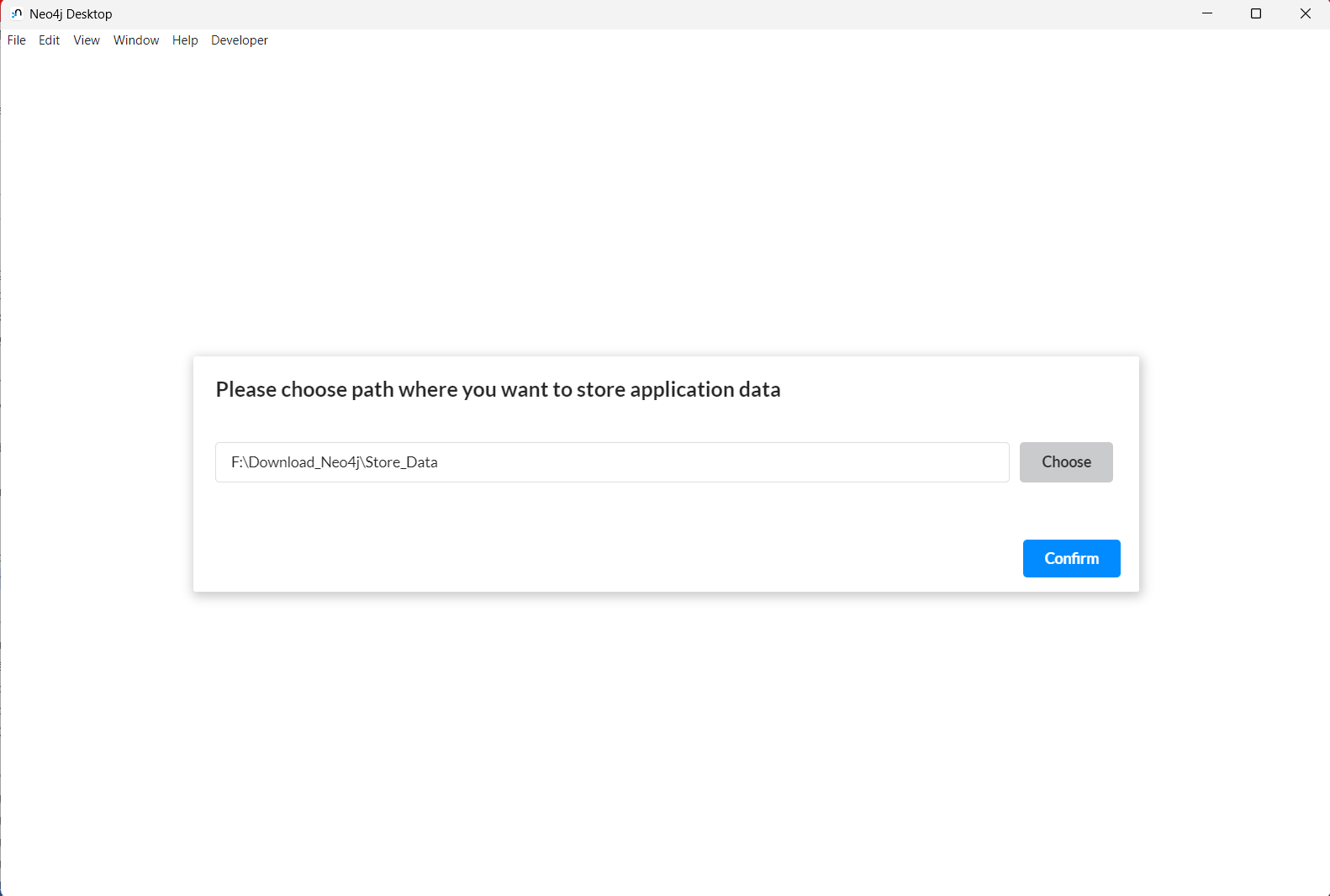
* Mở file .exe Neo4j đã tải về và cài đặt
* Chọn quyền sử dụng là “Only for Admin”



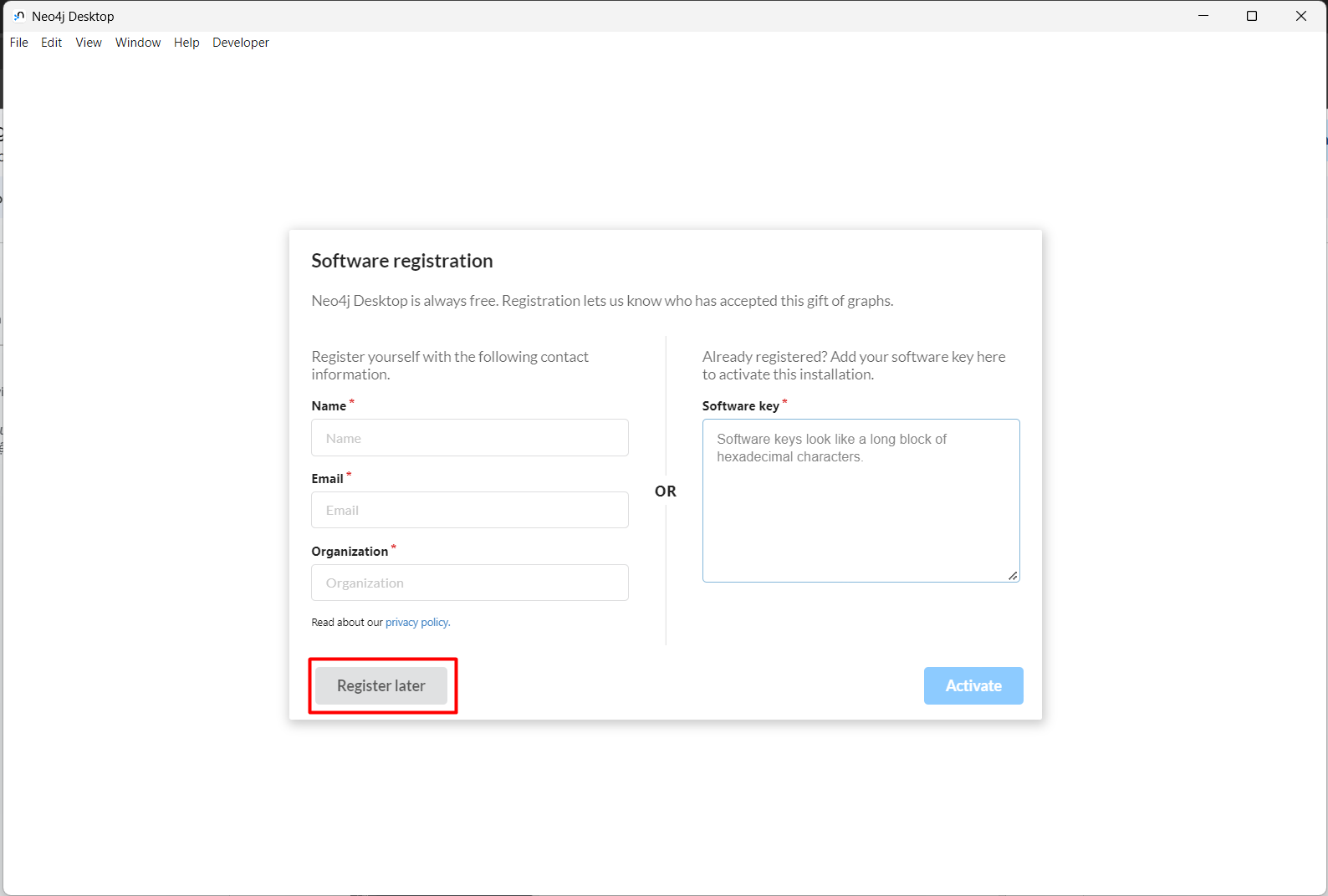
* Chọn nơi lưu trữ:



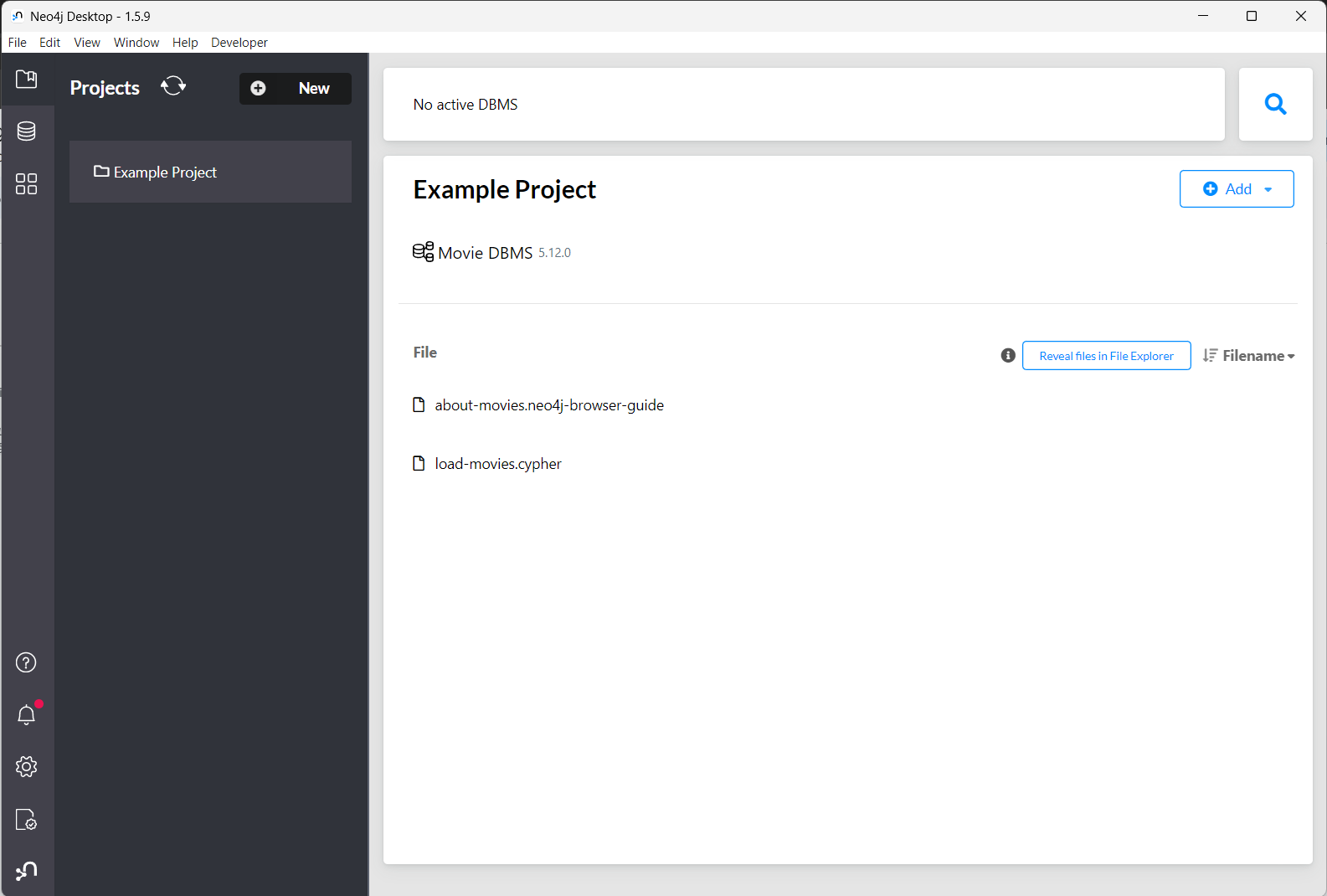
* Chọn nơi lưu dữ liệu application



* Click Register Later

****

* Cài đặt xong, mở lên ta sẽ được giao diện như sau:



### 2.5.2. Cài đặt Neo4j Desktop trên Linux

Bước 1: Truy cập trang Download chính chủ của Neo4j để tải bản mới nhất: [Neo4j Desktop Download | Free Graph Database Download](https://neo4j.com/download/)

A screenshot of a computer

Description automatically generated

Bước 2: Nhập thông tin mà trang yêu cầu -> Download

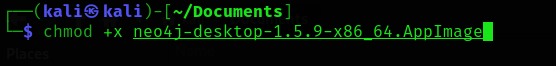
A screenshot of a computer

Description automatically generated

Bước 3: Tiến hành cài đặt

- Mở file .AppImage Neo4j đã tải về đặt quyền và cài đặt

\* Đặc quyền:

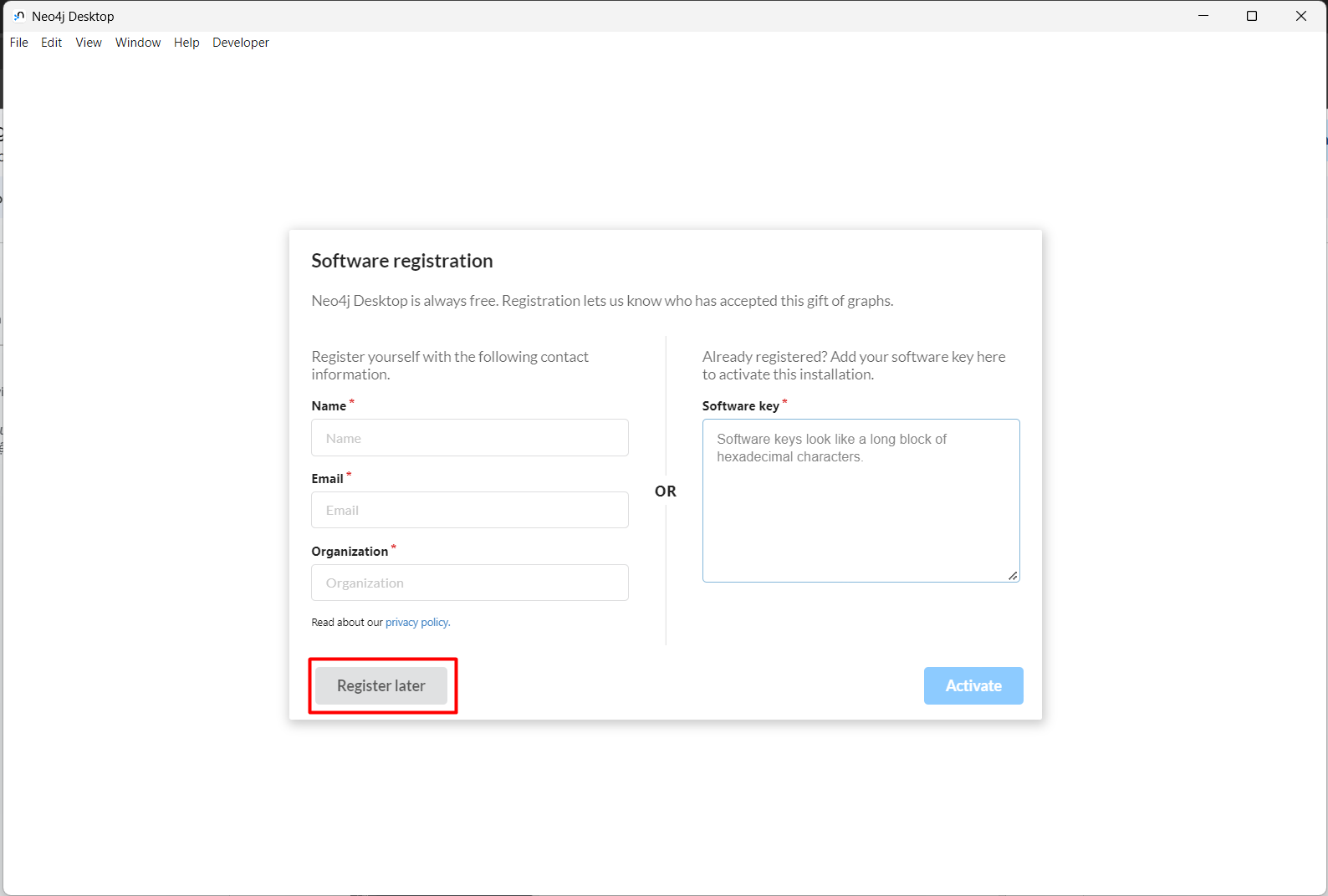


\* Chạy dòng lệnh cài đặt:

A black background with green text

Description automatically generated

- Click Register Later hoặc copy key có sẵn từ trang web

****

A screenshot of a computer

Description automatically generated

- Nhấn mật khẩu để unlock.

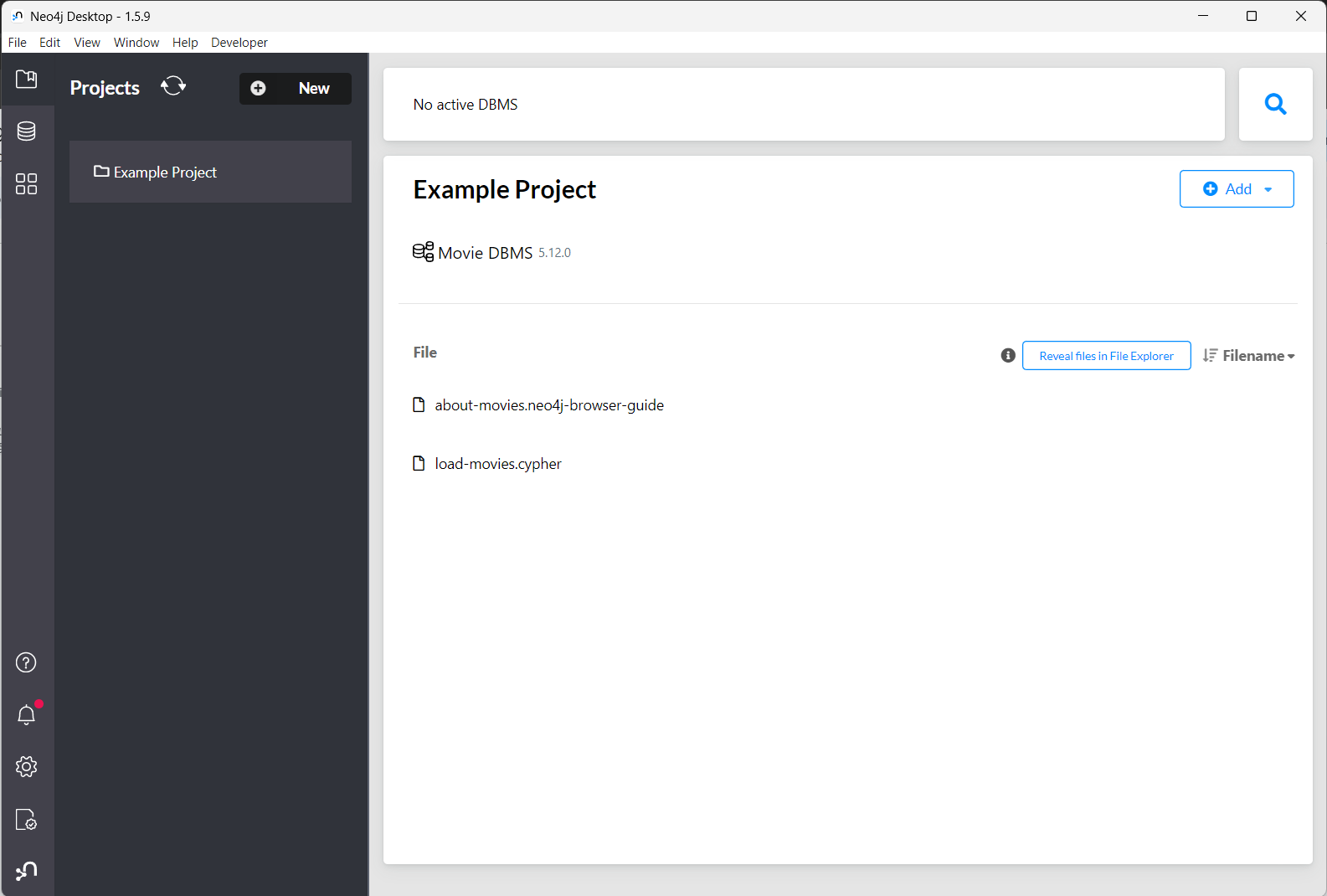
- Cài đặt xong, mở lên ta sẽ được giao diện như sau:

A screenshot of a computer

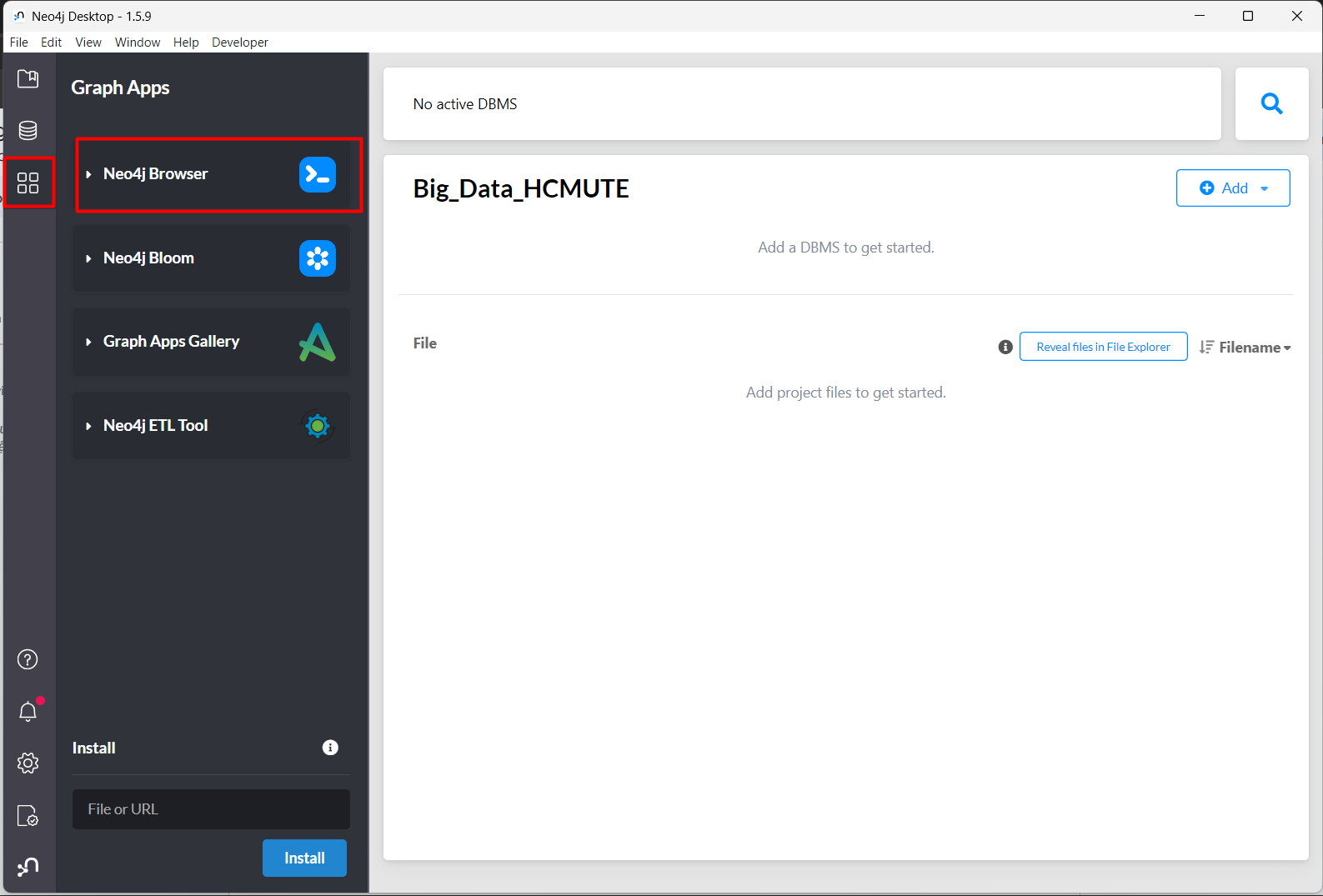
Description automatically generated

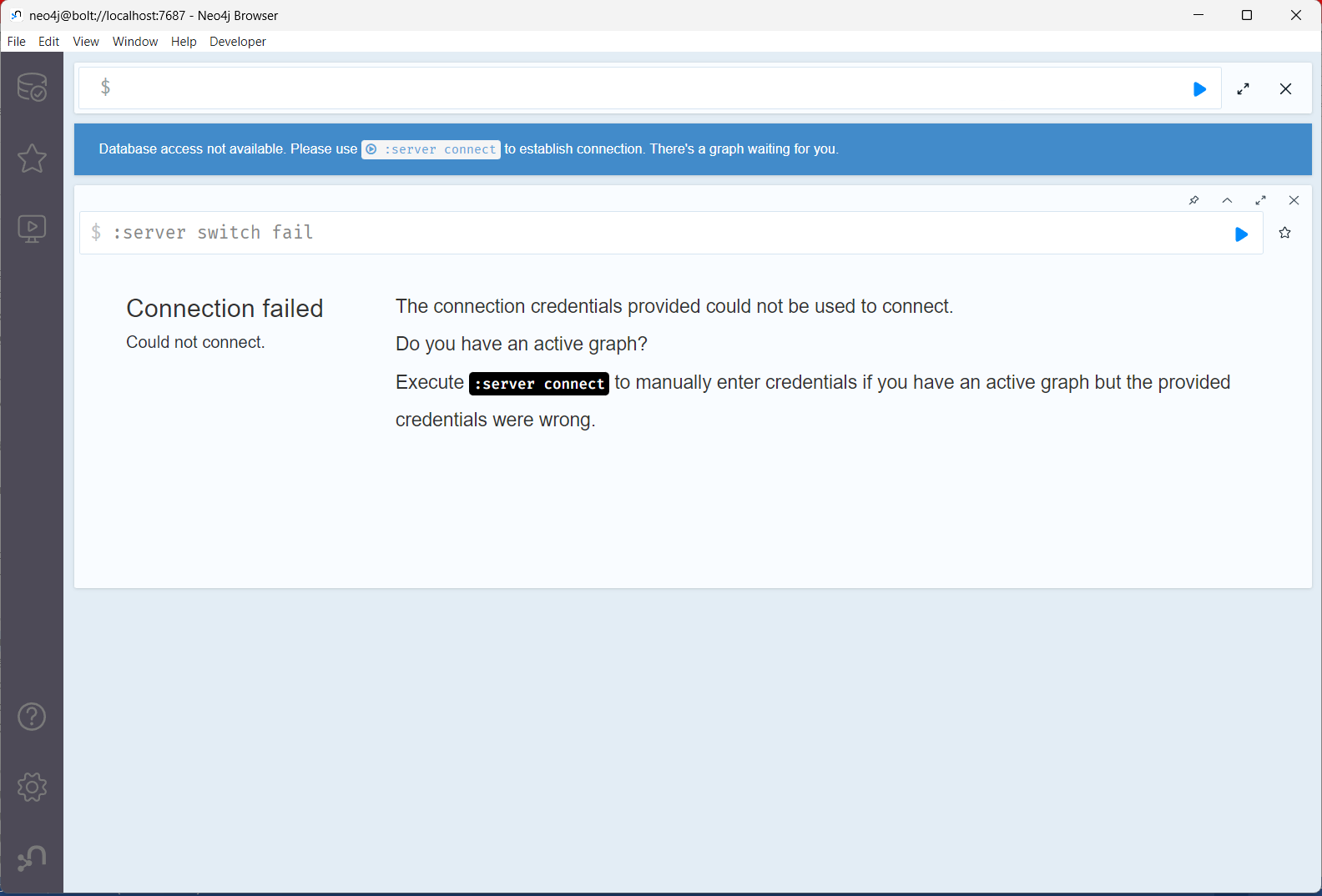
## **2.6. Làm việc với Neo4j**

### 2.6.1. Neo4j Desktop



### 2.6.2. Neo4j Browser



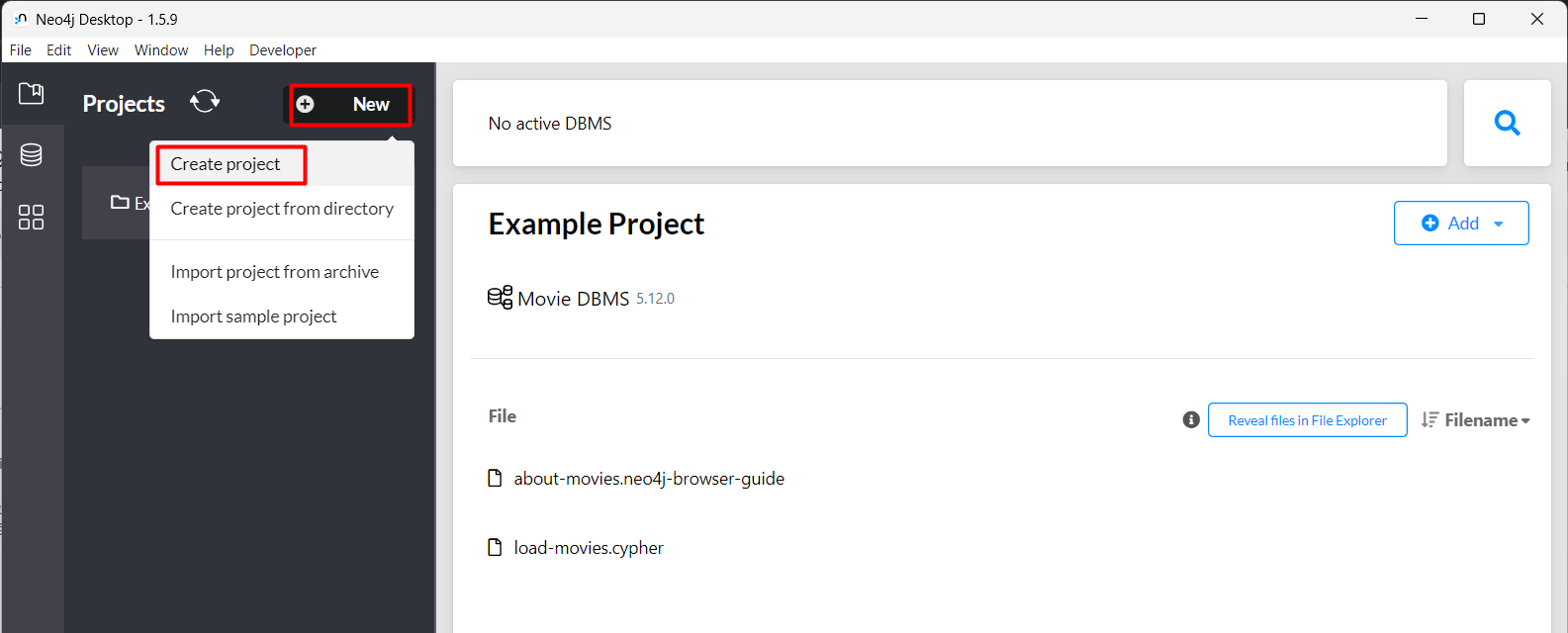


## **2.7. Tạo Local DBMS trong Neo4j Desktop và Import File**

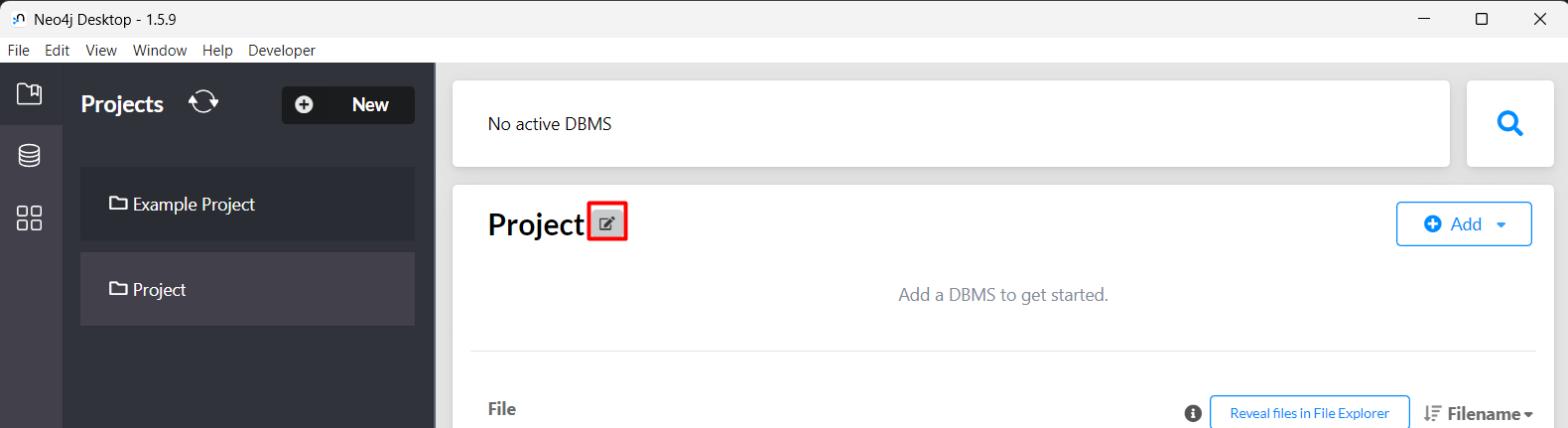
### 2.7.1. Tạo Project và Local DBMS

Đầu tiên, ta tạo một Project để lưu trữ:

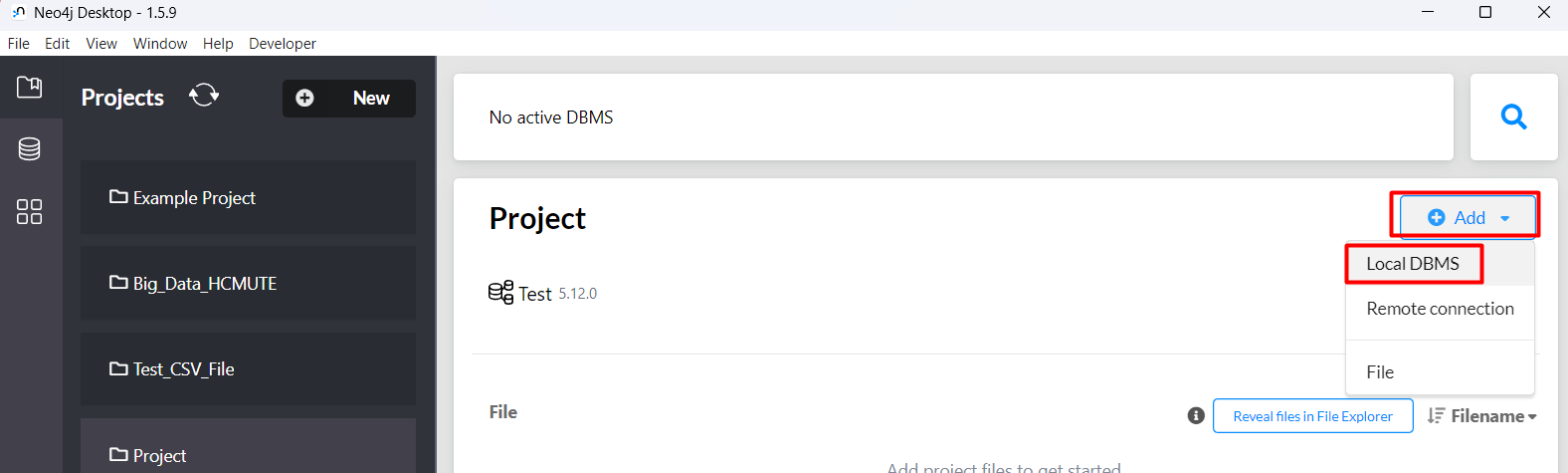
* Vào Neo4j Desktop -> New -> Create Project



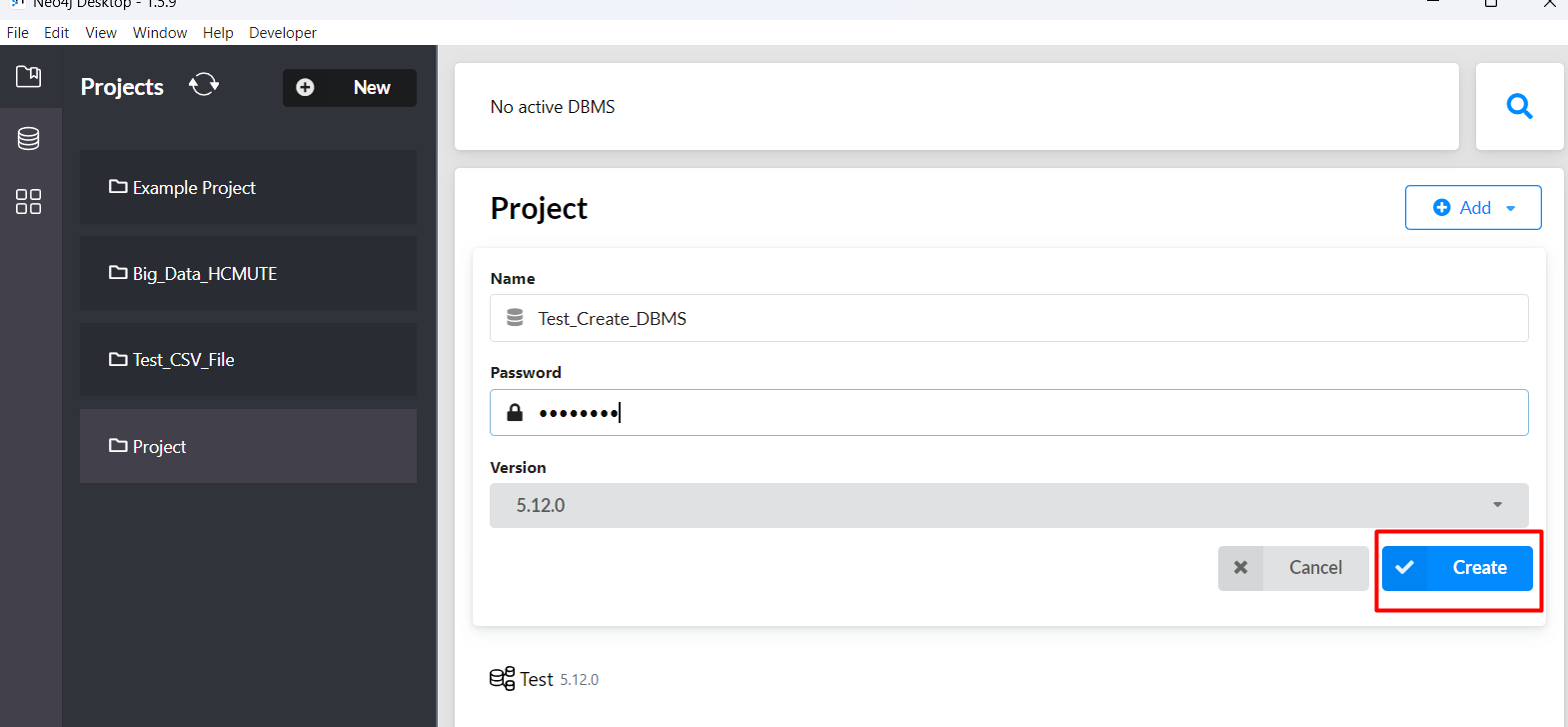
* Đổi tên, ví dụ: Big\_Data\_HCMUTE



* Tạo DBMS: chọn Add -> Local DBMS:



* Nhập Name và Password (lưu ý password trên 8 ký tự) -> Create

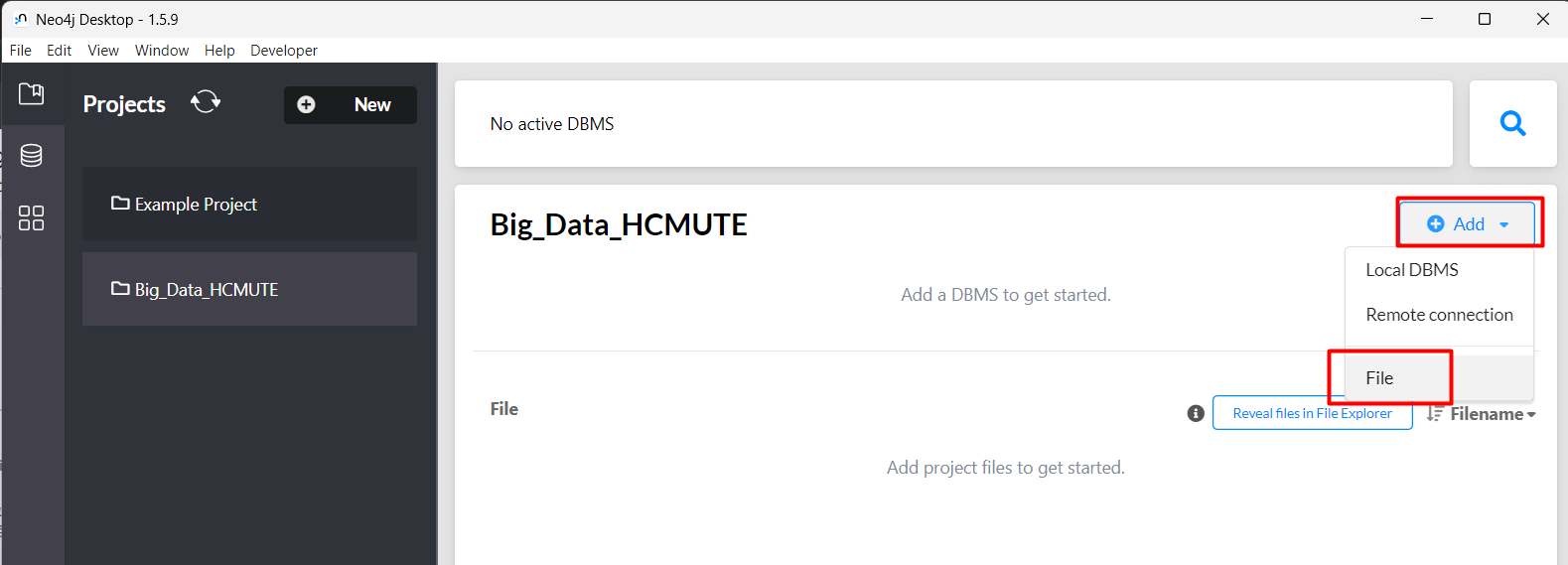


* Chọn Start

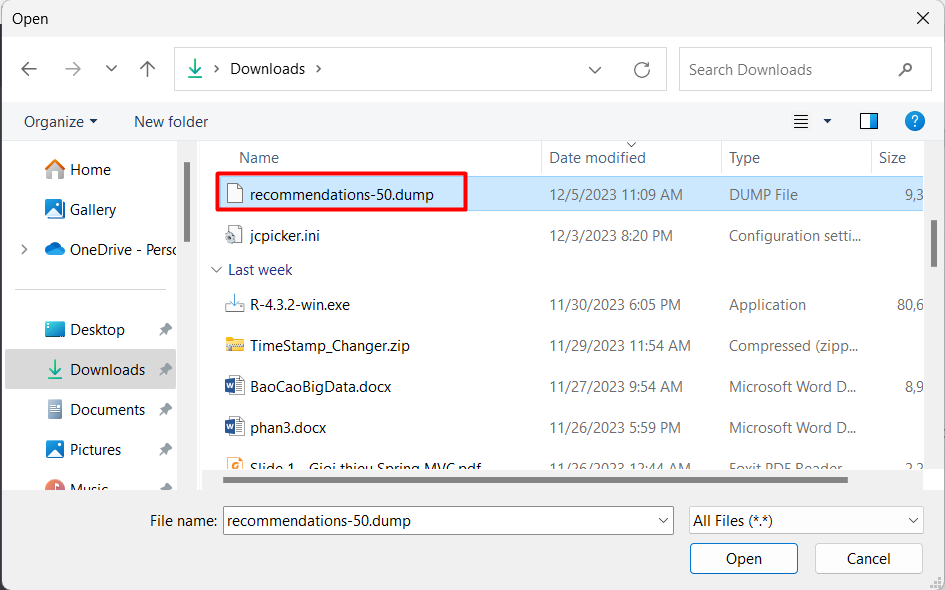


### 2.7.2. Import dump file

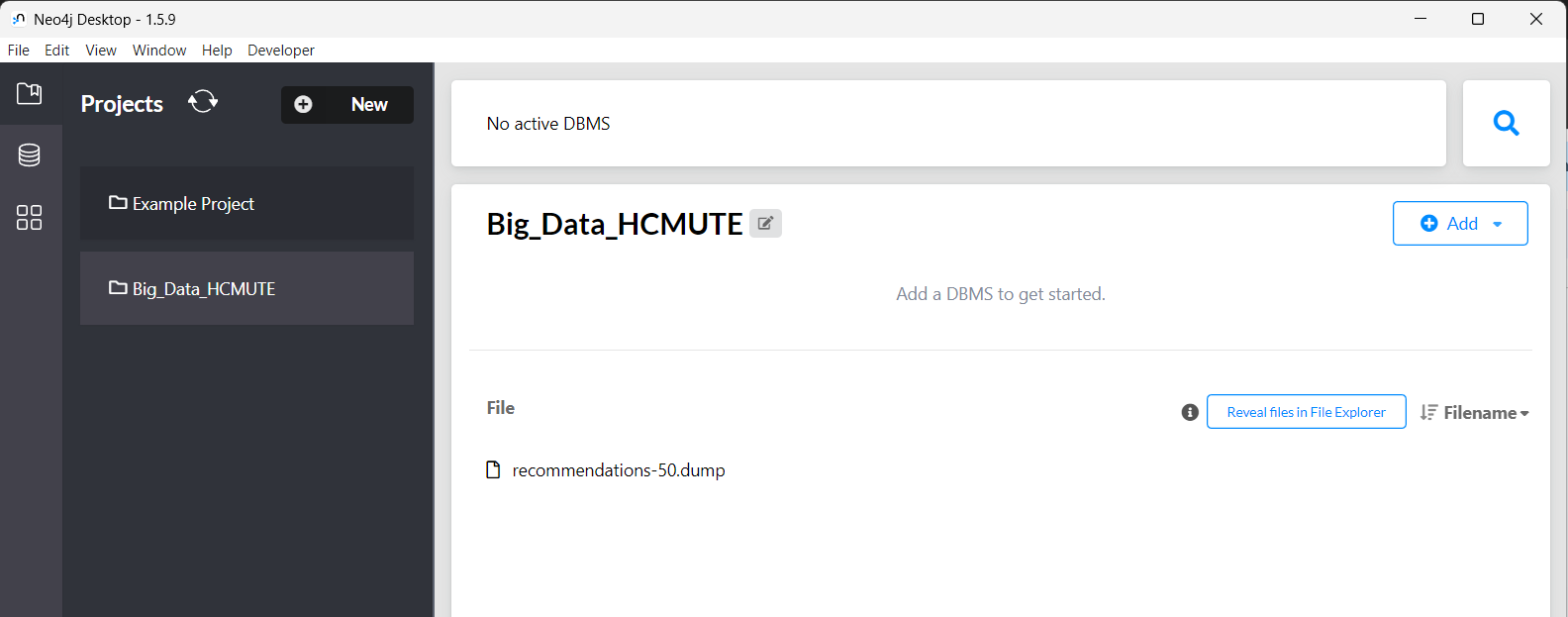
* Chọn Add -> File



* Chọn File muốn import

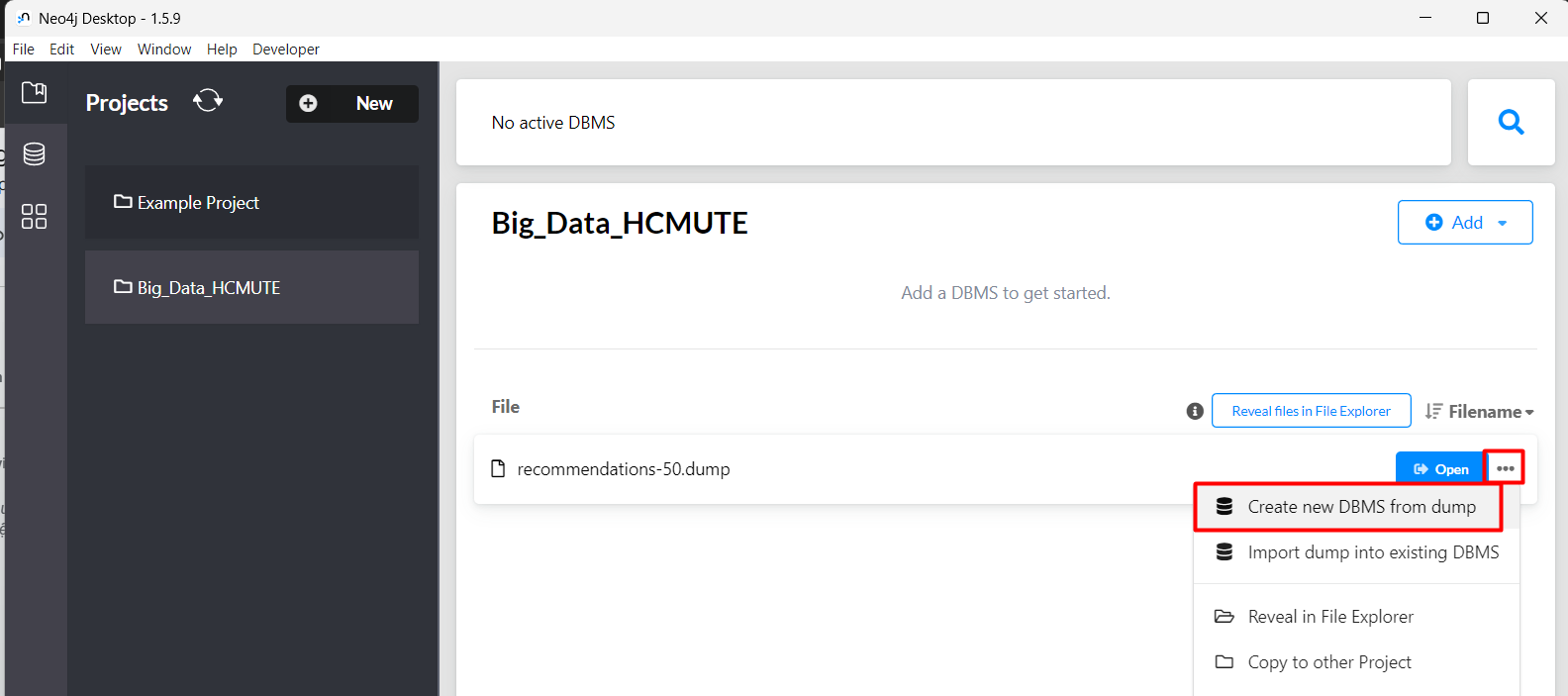


File đã import sẽ nằm ở mục File

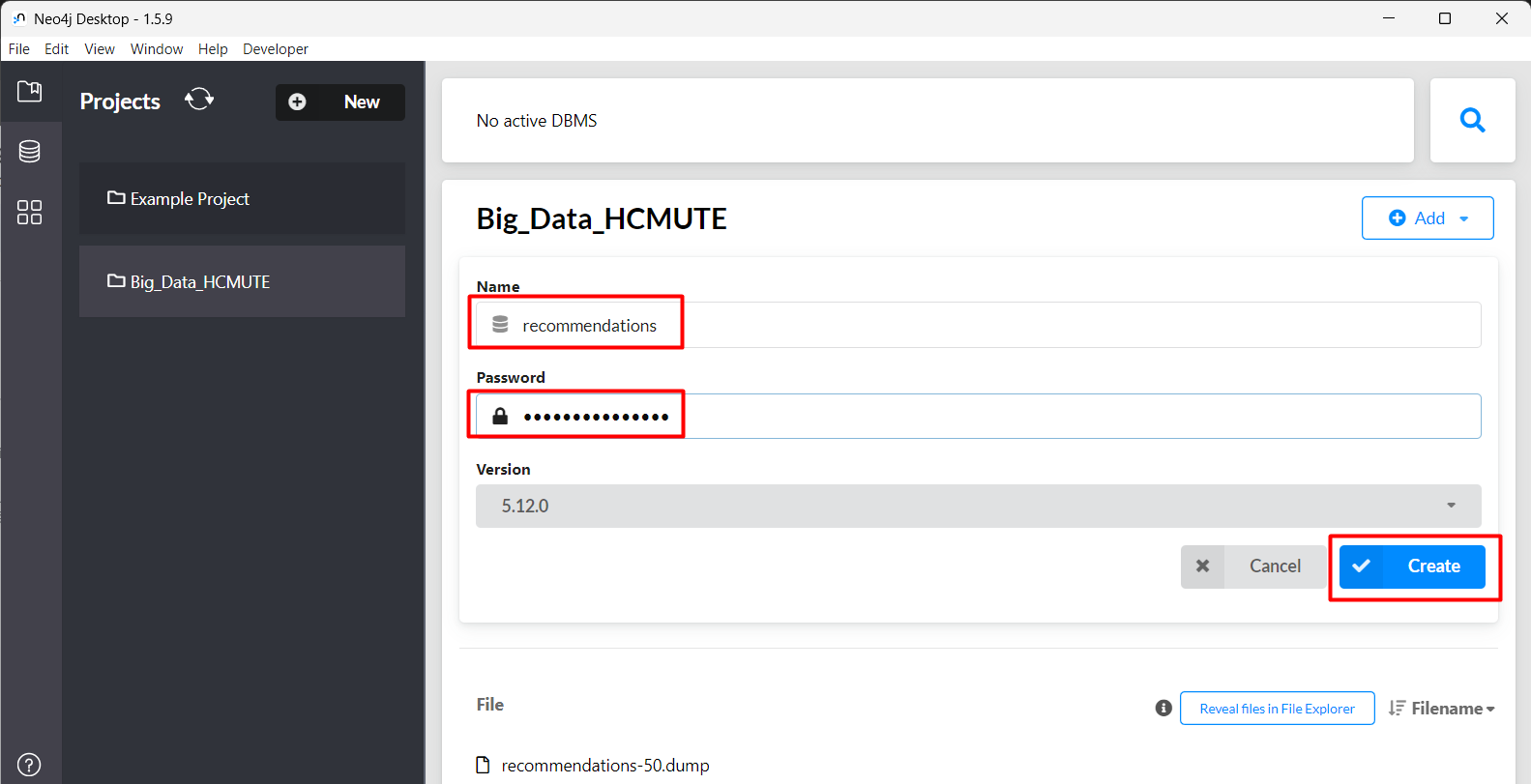


Tạo 1 DBMS mới từ file dump:

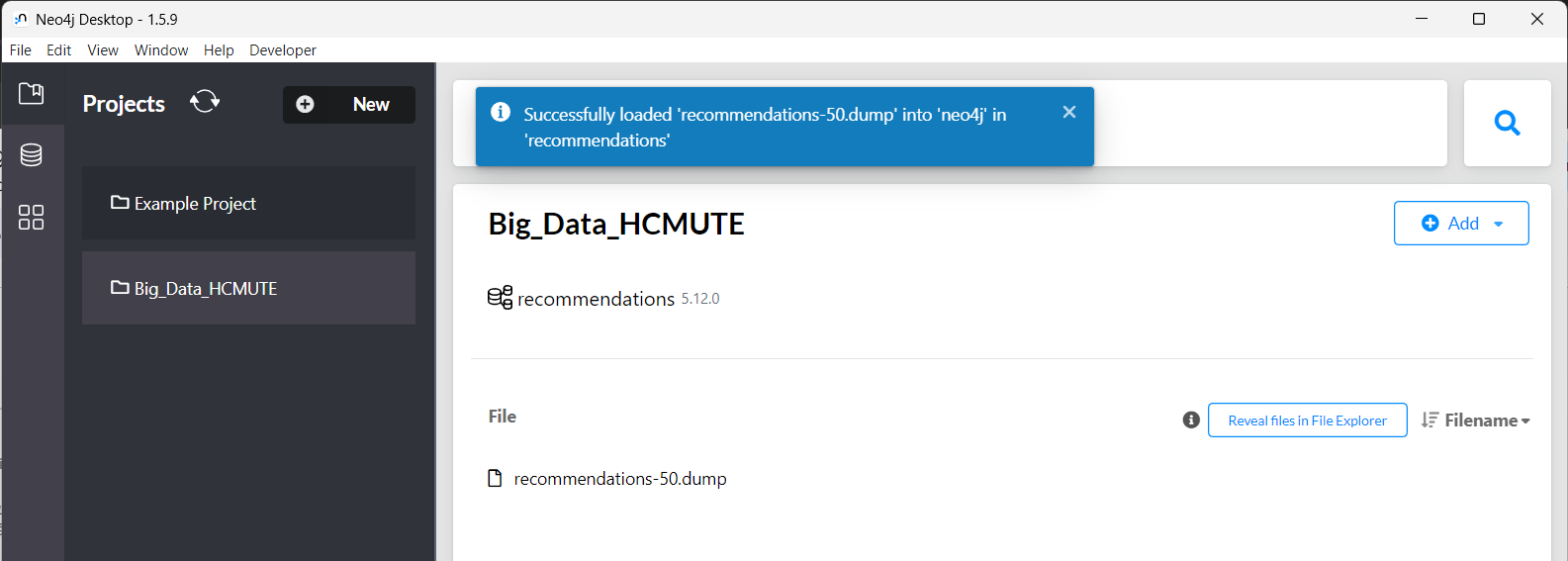
* Click vào biểu tượng “...” bên cạnh Open -> Create new DBMS from dump



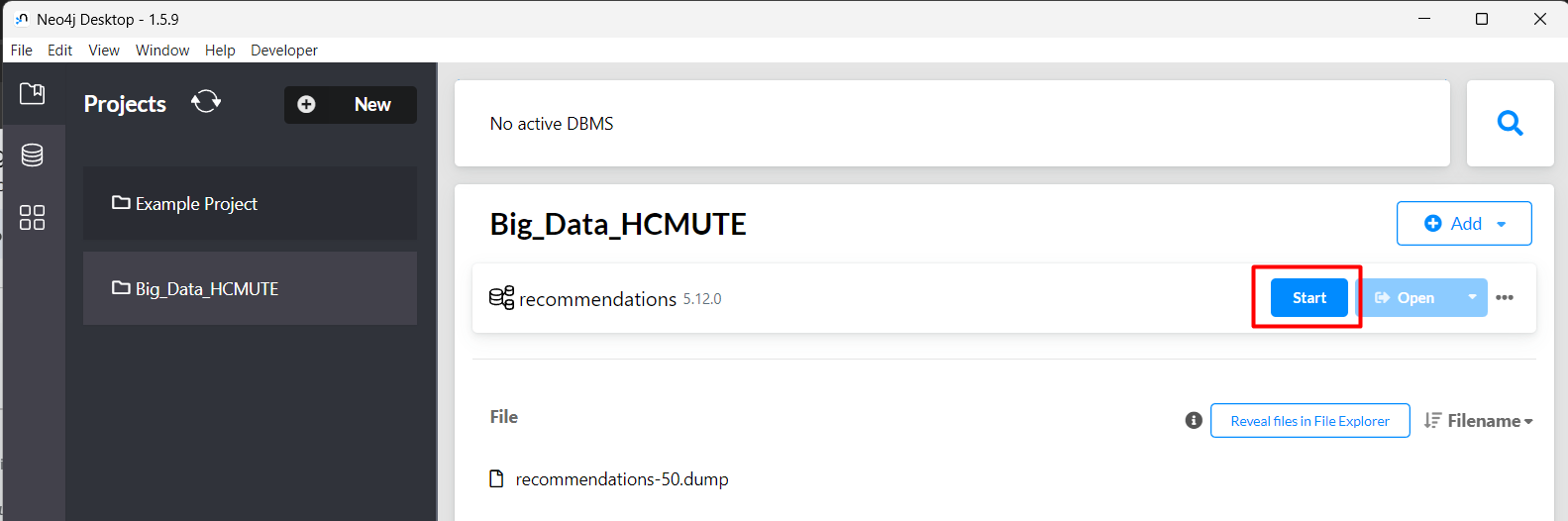
* Nhập Tên đăng nhập và Mật khẩu cho DBMS:



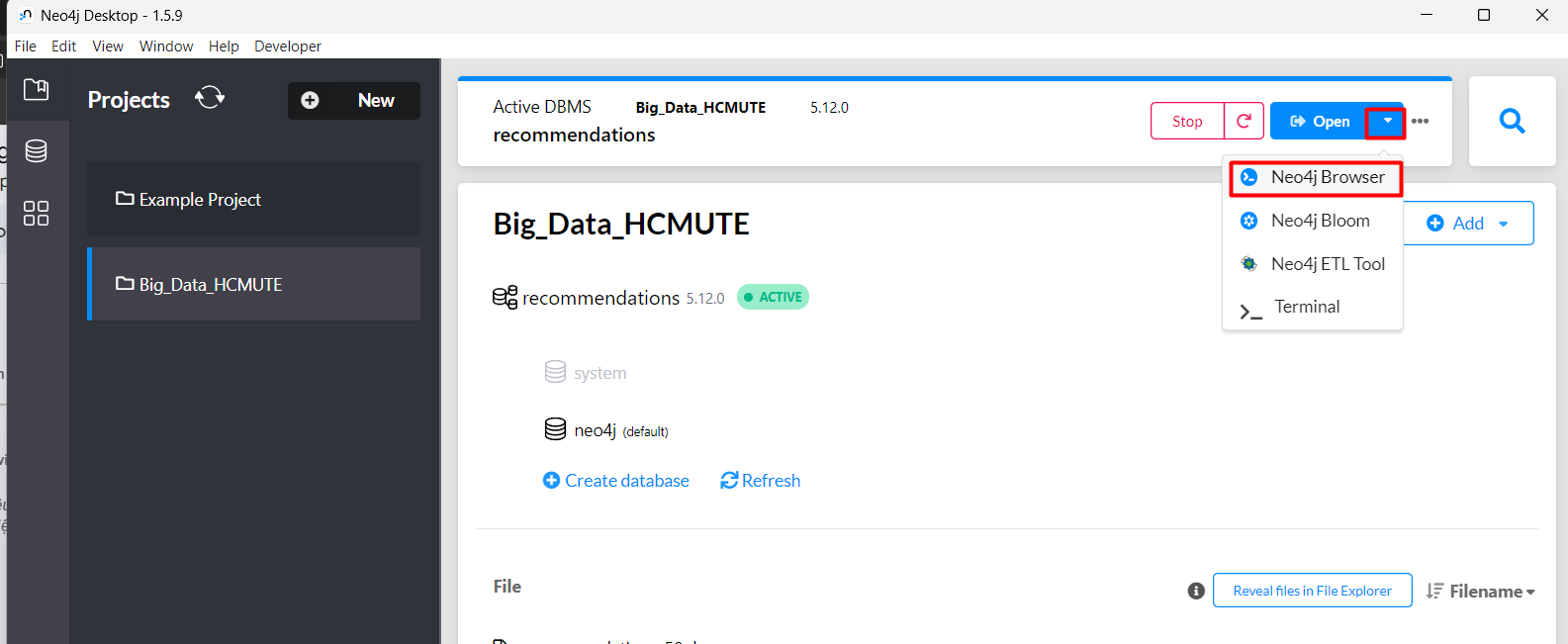
Vậy là ta đã import dump file thành công



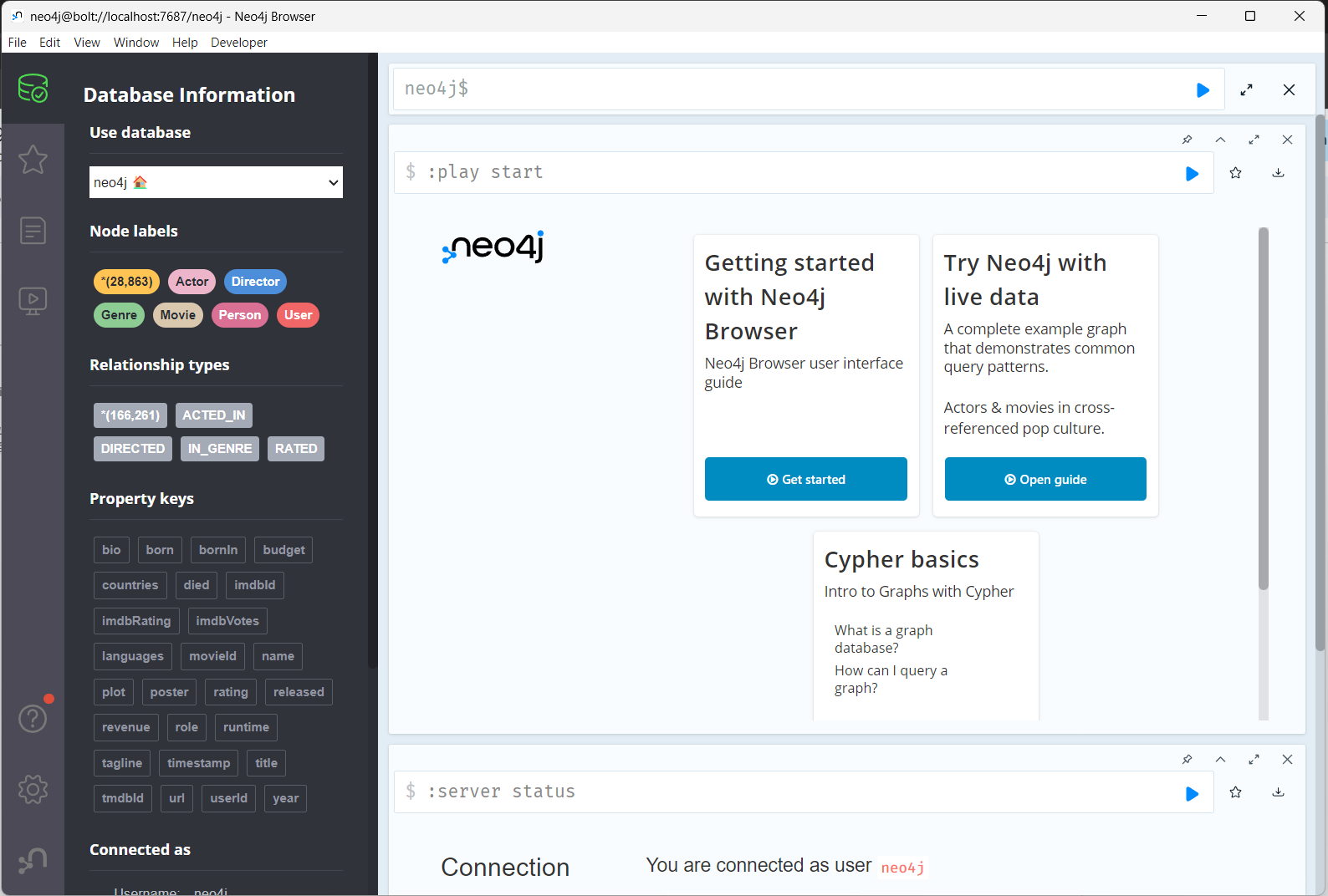
Để mở, ta click vào Start



Bên cạnh Open, click và chọn Neo4j Browser

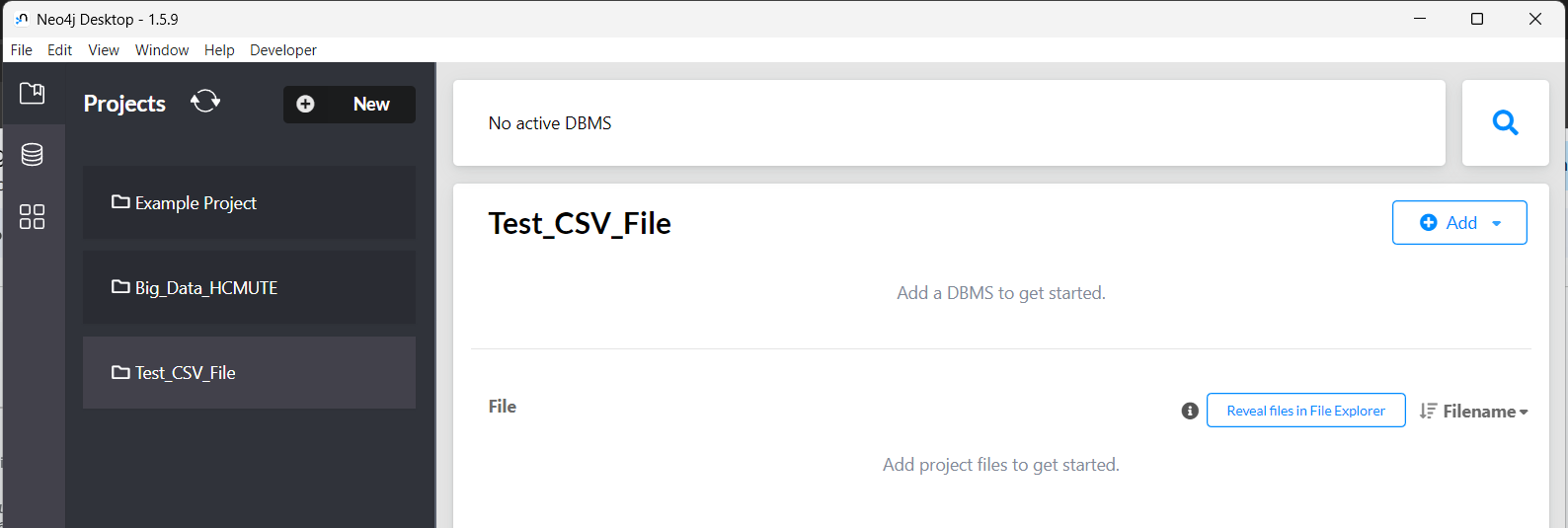


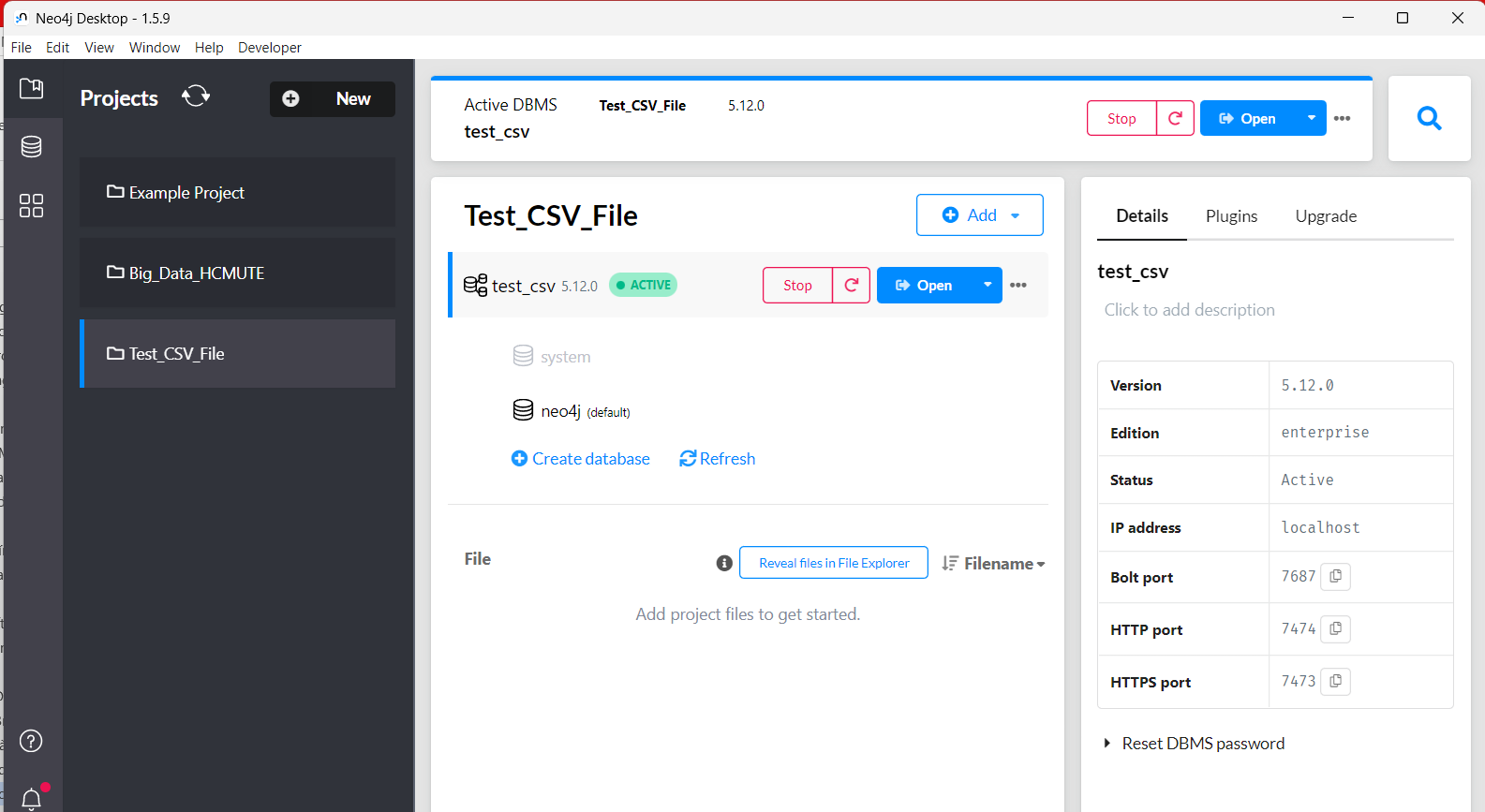
Thành công sẽ được giao diện Neo4j Browser như dưới



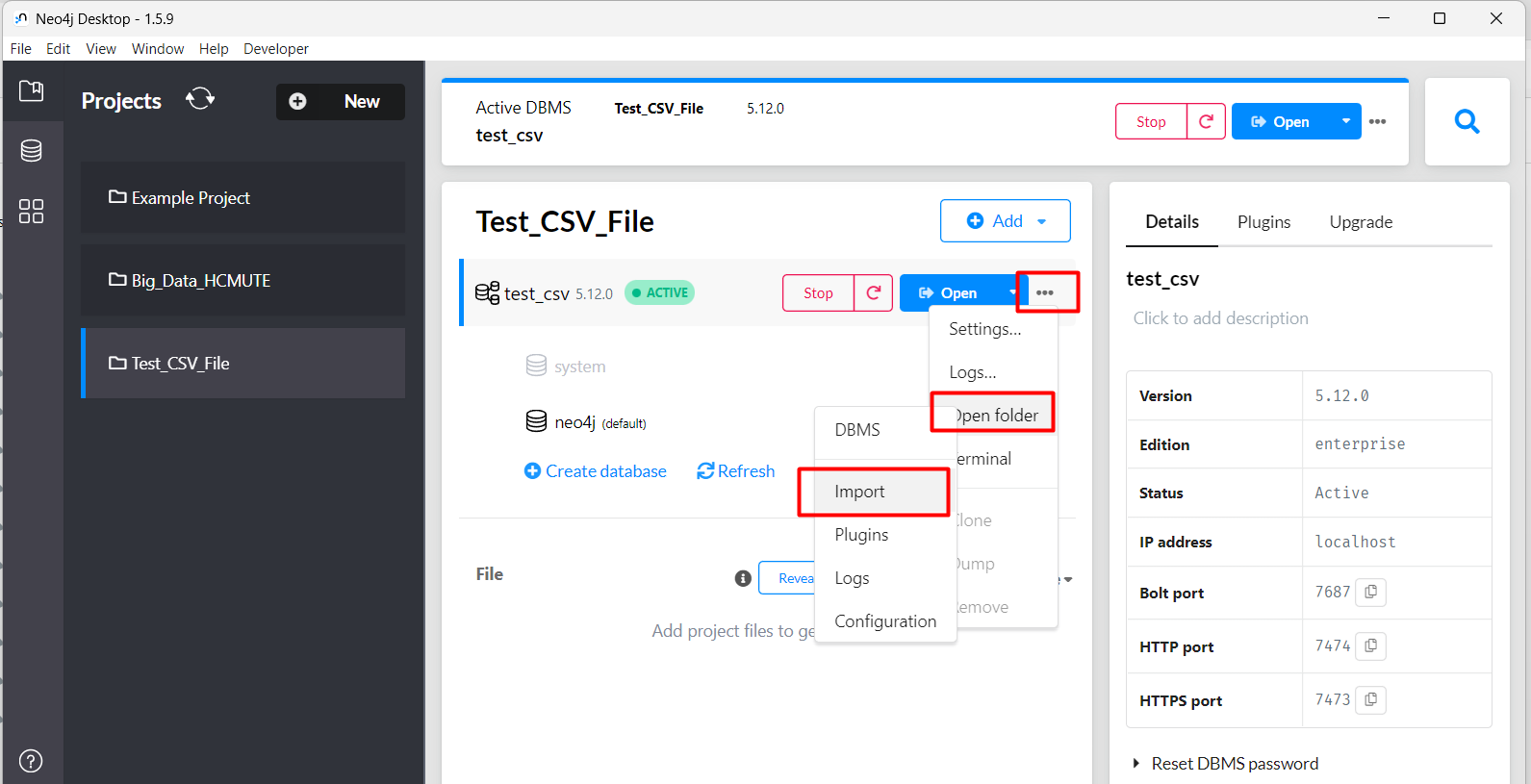
### 2.7.3. Import csv file

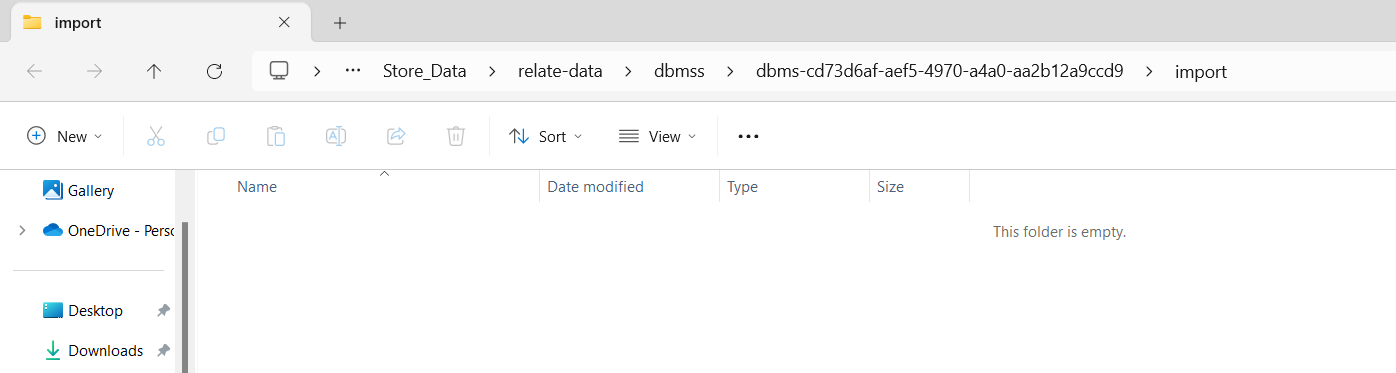
Tạo 1 Project và DBMS mới, sau đó tìm đến Open Folder



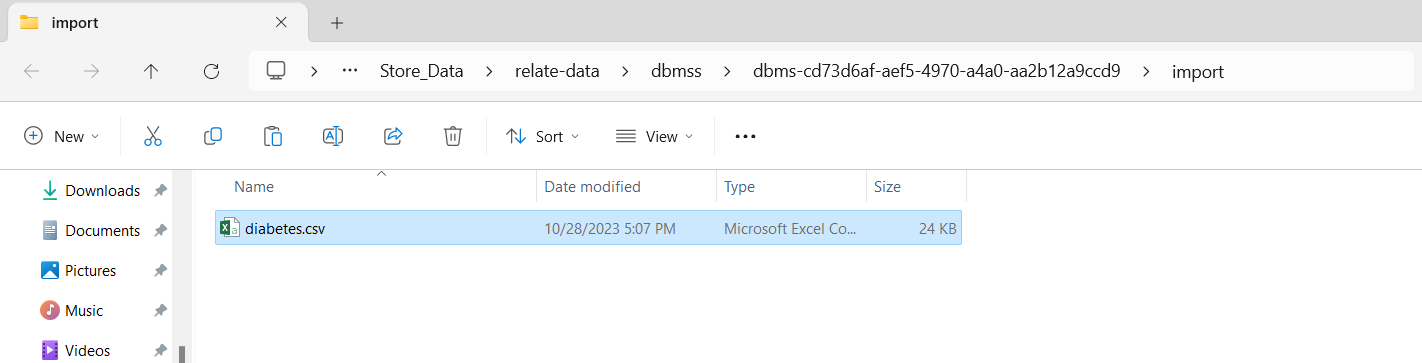


Tìm đường dẫn đến thư mục Import



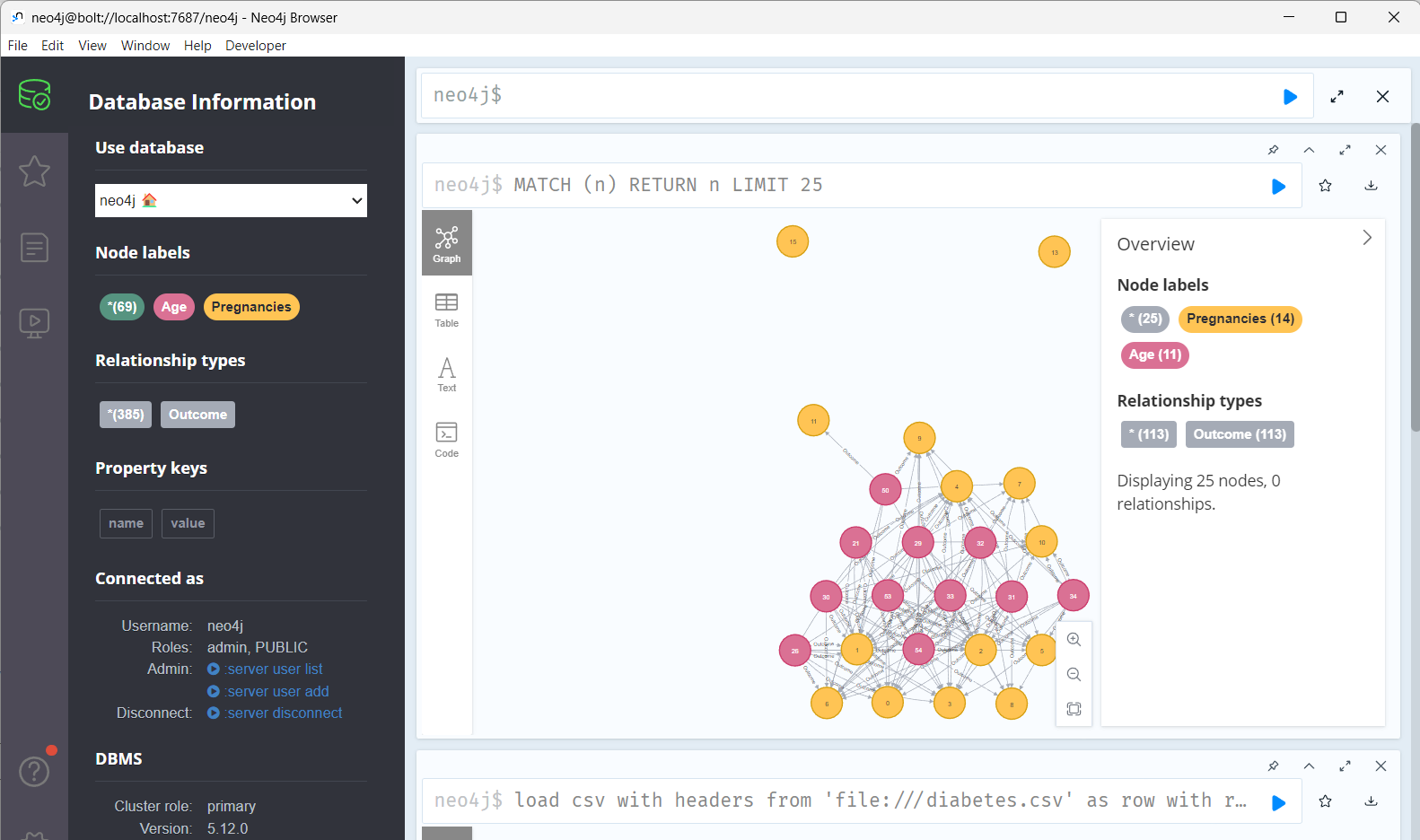


Copy file .csv vào



Ở giao diện Neo4j Browser, dùng lệnh load csv with headers

|  |
| --- |
| load csv with headers from 'file:///diabetes.csv' as row with row where row.Pregnancies is not null  merge (p:Pregnancies {name: row.Pregnancies})  merge (a:Age {name: row.Age})  merge (a) - [:Outcome {value:row.Outcome}]-> (p) |



# **CHƯƠNG 3. Neo4j – CYPHER QUERY LANGUAGE**

Cypher là ngôn ngữ truy vấn biểu đồ khai báo cho phép truy vấn, cập nhật và quản lý biểu đồ một cách rõ ràng và hiệu quả. Nó được thiết kế để phù hợp cho cả nhà phát triển và chuyên gia vận hành. Cypher được thiết kế đơn giản nhưng mạnh mẽ; các truy vấn cơ sở dữ liệu rất phức tạp có thể được diễn đạt dễ dàng, cho phép bạn tập trung vào miền của mình thay vì bị mất quyền truy cập cơ sở dữ liệu.  
 Từ khóa Cypher không phân biệt chữ hoa chữ thường.

Cypher phân biệt chữ hoa chữ thường đối với các biến.

Có các quy tắc đặt tên đặc biệt cho tên cơ sở dữ liệu.

Có các quy tắc đặt tên đặc biệt cho bí danh cơ sở dữ liệu.

## **3.1. Neo4j - Cypher Query Language**

* Là ngôn ngữ truy vấn cho Cơ sở dữ liệu đồ thị Neo4j.
* Là một ngôn ngữ khớp mẫu khai báo.
* Giống SQL như cú pháp.
* Cú pháp rất đơn giản và ở định dạng con người có thể đọc được.

## **3.2. MATCH**

|  |  |
| --- | --- |
| **Cú pháp** | **Ý nghĩa** |
| MATCH (n) RETURN n | Truy xuất tất cả các nút (Node) trong cơ sở dữ liệu Neo4j. |
| Kết quả: | |
| MATCH (node:label)<-[: Relationship]-(n)  RETURN n  Ví dụ:  MATCH (m:Movie)<-[:ACTED\_IN]-(a:Actor)  RETURN a | Truy xuất các nút (Node) dựa trên mối quan hệ (Relationship). |
| Kết quả: | |

## **3.3. RETURN**

Mệnh đề RETURN được sử dụng các nút (node) trả về, mối quan hệ (relationship) và thuộc tính (properties) trong Neo4j.

|  |  |
| --- | --- |
| **Cú pháp** | **Ý nghĩa** |
| CREATE (node:label {properties})  RETURN node  Ví dụ:  CREATE (n:User {name:"Han", userId: '21133032'}) return n | Trả về một nút (node). |
| Kết quả: | |
| CREATE (node1:label1 {properties1})  CREATE (node2:label2 {properties2})  RETURN node1, node2  Ví dụ:  CREATE (n1:User {name:"NgocTran", userId: '21133033'})  CREATE (n2:User {name:"TrongDung", userId:'25614785'})  RETURN n1,n2 | Trả về nhiều nút (node). |
| Kết quả: | |
| CREATE (node1)-[Relationship:Relationship\_type]->(node2)  RETURN Relationship  Ví dụ:  create (nxb:NhàXuấtBản{name:'Kim Đồng'})  create (s:Sách{name:'Doraemon'})  create (nxb)-[xb:XuấtBản]->(s)  return nxb,xb,s | Trả về các mối quan hệ (relationship). |
| Kết quả: | |
| Match (node:label {properties . . . . . . . . . . })  Return node.property  Ví dụ:  MATCH (n:User {name:"NgocTran"})  RETURN n.userId | Trả về các thuộc tính (properties) |
| Kết quả: | |
| MATCH (n:Label)-[r]->(m:Label)  RETURN \*  Ví dụ:  MATCH (u:Actor)-[r:ACTED\_IN]->(m:Movie)  RETURN \* | Trả về giá trị của tất cả các biến. |
| Kết quả: | |
| MATCH (n:Label)-[r]->(m:Label)  RETURN n AS node, r AS rel  Ví dụ:  MATCH (u:Actor)-[r:ACTED\_IN]->(m:Movie)  RETURN u AS DienVien, m AS Phim | Trả về cột cụ thể có bí danh. |
| Kết quả: | |
| MATCH (n:Person)-[r:KNOWS]-(m:Person)  RETURN DISTINCT n AS node  Ví dụ:  MATCH (u:Actor)-[r:ACTED\_IN]->(m:Movie)  RETURN DISTINCT u AS DienVien | Trả về các hàng duy nhất. |
| Kết quả: | |
| MATCH (n:Label)-[r]->(m:Label)  RETURN n AS node, r AS rel  ORDER BY n.name  Ví dụ:  MATCH (u:Actor)-[r:ACTED\_IN]->(m:Movie)  RETURN u AS DienVien, m AS Phim  ORDER BY u.name | Sắp xếp kết quả theo thứ tự tăng dần. |
| Kết quả: | |
| MATCH (n:Label)-[r]->(m:Label)  RETURN n AS node, r AS rel  ORDER BY n.name DESC  Ví dụ:  MATCH (u:Actor)-[r:ACTED\_IN]->(m:Movie)  RETURN u AS DienVien, m AS Phim  ORDER BY u.name DESC | Sắp xếp kết quả theo thứ tự giảm dần. |
| Kết quả: | |
| MATCH (n:Label)-[r]->(m:Label)  RETURN n AS node, r AS rel  SKIP 10  Ví dụ:  MATCH (u:Actor)-[r:ACTED\_IN]->(m:Movie)  RETURN u AS DienVien, m AS Phim  SKIP 10 | Bỏ qua 10 hàng đầu tiên để có tập kết quả. |
| Kết quả: | |
| MATCH (n:Label)-[r]->(m:Label)  RETURN n AS node, r AS rel  LIMIT 10  Ví dụ:  MATCH (u:Actor)-[r:ACTED\_IN]->(m:Movie)  RETURN u AS DienVien, m AS Phim  LIMIT 10 | Giới hạn số hàng tối đa là 10 cho tập kết quả. |
| Kết quả: | |

## **3.4. WHERE**

Giống như SQL, Neo4j CQL đã cung cấp mệnh đề WHERE trong lệnh SQL MATCH để lọc kết quả của MATCH Query.

|  |  |
| --- | --- |
| **Cú pháp** | **Ý nghĩa** |
| MATCH (label)  WHERE label.country = "property"  RETURN label  Ví dụ:  MATCH (User)  WHERE User.name = "TrongDung"  RETURN User | Sử dụng mệnh đề WHERE trong Neo4j với một điều kiện |
| Kết quả: | |
| MATCH (emp:Employee)  WHERE emp.name = 'Abc' AND emp.name = 'Xyz'  RETURN emp  Ví dụ:  MATCH (User)  WHERE User.name = "TrongDung" and User.userId='25614785'  RETURN User | Sử dụng mệnh đề WHERE trong Neo4j với nhiều điều kiện. |
| Kết quả: | |

## **3.5. CREATE**

|  |  |
| --- | --- |
| **Cú pháp** | **Ý nghĩa** |
| CREATE (node\_name)  Ví dụ:  CREATE (han) | Tạo node đơn. |
| Kết quả:  A screenshot of a computer  Description automatically generated | |
| CREATE (node1),(node2)  Ví dụ:  CREATE (han),(dung) | Tạo nhiều node. |
| Kết quả:  A screenshot of a computer  Description automatically generated | |
| CREATE (node:label)  Ví dụ:  CREATE (n:User) | Tạo node có label |
| Kết quả:  A screenshot of a computer  Description automatically generated | |
| CREATE (node:label1:label2:. . . . label)  Ví dụ:  CREATE (n:User:Actor) | Tạo node có nhiều label |
| Kết quả:  A screenshot of a computer  Description automatically generated | |
| CREATE (n:Label {name: $value})  Ví dụ:  CREATE (n:User {name:”ngoctrang”}) | Tạo node có label và properties nhất định. |
| Kết quả:  A screenshot of a computer  Description automatically generated | |
| CREATE (n:Label)-[r:TYPE]->(m:Label)  Ví dụ:  CREATE (nxbTT:NhàXuấtBản {name:'Tuổi Trẻ'})  CREATE (s4:Sách {name:'Kính Vạn Hoa'})  CREATE (nxbTT)-[r:XuấtBản]->(s4) | Tạo Relationship với kiểu và hướng quan hệ nhất định, liên kết với biến r với nó. |
| Kết quả:  A screenshot of a computer  Description automatically generated | |
| CREATE (n:Label)-[:TYPE {name: $value}]->(m:Label)  Ví dụ:  CREATE (nxbTT:NhàXuấtBản {name:'Tuổi Trẻ'})  CREATE (s4:Sách {name:'Kính Vạn Hoa'})  CREATE (s5:Sách {name:'Ngữ Văn'})  CREATE (s6:Sách {name:'Toán'})  CREATE (nxbTT)-[:XuấtBản{year:2003}]->(s4)  CREATE (nxbTT)-[:XuấtBản{year:2004}]->(s5)  CREATE (nxbTT)-[:XuấtBản{year:2004}]->(s6)  match(nxb:`NhàXuấtBản`)-[xb:`XuấtBản`]->(s:`Sách`)  where xb.year=2004  return nxb,xb,s | Tạo Relationship với label và properties nhất định.  Tạo mối quan hệ của NXB Tuổi Trẻ với 3 cuốn sách xuất bản theo năm 2003 và 2004. |
| Kết quả:  A screen shot of a computer  Description automatically generated  A diagram of a network  Description automatically generated | |

## **3.6. WITH**

Cú pháp WITH tương tự như RETURN, nhưng không hiển thị dữ liệu. Nó phân tách các phần truy vấn một cách rõ ràng, cho phép người dùng khai báo những biến nào sẽ được chuyển sang phần tiếp theo của truy vấn.

WITH có thể sử dụng:

* ORDER BY
* SKIP
* LIMIT
* WHERE

|  |  |
| --- | --- |
| **Cú pháp** | **Ý nghĩa** |
| Ví dụ:  CREATE (nxbKD: NhàXuấtBản {name:'Kim Đồng'})  CREATE (s1:Sách {name:'Doraemon'})  CREATE (s2:Sách {name:'Conan'})  CREATE (s3:Sách {name:'Shin'})  CREATE (nxbKD)-[:XuấtBản]->(s1)  CREATE (nxbKD)-[:XuấtBản]->(s2)  CREATE (nxbKD)-[:XuấtBản]->(s3)  CREATE (nxbTT: NhàXuấtBản {name:'Tuổi Trẻ'})  CREATE (s4:Sách {name:'Kính Vạn Hoa'})  CREATE (nxbTT)-[:XuấtBản]->(s4)  MATCH (nxb:NhàXuấtBản)-[xb:XuấtBản]-(s:Sách)  WITH nxb, count(xb) AS xuatban  WHERE xuatban > 2  RETURN nxb | Tạo 2 NXB: Kim Đồng và Tuổi Trẻ. Tạo 4 Sách: Doraemon, Conan, Shin, Kính Vạn Hoa và cho mối quan hệ Xuất Bản: Kim Đồng - Doraemon, Conan, Shin; Tuổi Trẻ: Kính Vạn Hoa.  Cú Pháp WITH: trả về Nhà Xuất Bản có số lượng sách xuất bản > 2. |
| Kết quả: | |

## **3.7. SET**

Sử dụng mệnh đề Set, ta có thể thêm thuộc tính (properties) mới vào nút (node) hoặc mối quan hệ (relationship) hiện có, đồng thời thêm hoặc cập nhật các giá trị thuộc tính (properties) hiện có.

|  |  |
| --- | --- |
| **Cú pháp** | **Ý nghĩa** |
| SET e.property1 = $value1  Ví dụ  MATCH (User) WHERE User.name='TrongDung' SET User.class='21133C' RETURN User | Cập nhật hoặc tạo thuộc tính.  Thêm thuộc tính class cho User có name là ‘TrongDung’ |
| Kết quả: | |
| SET node.property1 = value, node.property2 = value  Ví dụ:  MATCH (User) WHERE User.name='TrongDung' SET User.born='2003', User.faculty='cntt' RETURN User | Tạo nhiều thuộc tính trong một node. |
| Kết quả: | |
| MATCH (node:label {properties})  SET node.property = NULL  RETURN node  Ví dụ:  MATCH (User) WHERE User.name='TrongDung' SET User.born=NULL RETURN User | Xóa một thuộc tính hiện có bằng cách chuyển **null** dưới dạng giá trị cho thuộc tính đó. |
| Kết quả: | |
| MATCH (n:Label)  WHERE n.property = 123  SET n:Person  Ví dụ: match (nxb:`NhàXuấtBản`) where nxb.name='Tuổi Trẻ'  set nxb:NhàSách  return nxb | Thêm một Label vào node.  Thêm label NhàSách vào Nhà Xuất Bản Tuổi Trẻ. |
| Kết quả: | |
| MATCH (n:label1 {properties . . . . . . . })  SET n :label2:label3  RETURN n  Ví dụ:  MATCH (n:User{name: 'Chi Cong'})  SET n:Student:Actor  RETURN n | Đặt nhiều Label cho một Node hiện có |
| Kết quả: | |

## **3.8. MERGE**

Lệnh MERGE là sự kết hợp giữa lệnh CREATE và lệnh MATCH.

Lệnh Neo4j CQL MERGE tìm kiếm một mẫu nhất định trong biểu đồ. Nếu nó tồn tại, thì nó sẽ trả về kết quả. Nếu nó KHÔNG tồn tại trong biểu đồ, thì nó sẽ tạo một nút/mối quan hệ mới và trả về kết quả.

|  |  |
| --- | --- |
| **Cú pháp** | **Ý nghĩa** |
| MERGE (n:Label {name: $value})  ON CREATE SET n.created = timestamp()  ON MATCH SET  n.counter = coalesce(n.counter, 0) + 1,  n.accessTime = timestamp()  VD:  MERGE (n:user {name:"Alice"})  ON CREATE SET n.created = timestamp()  ON MATCH SET  n.counter = coalesce(n.counter, 0) + 1,  n.accessTime = timestamp() | MATCH một mẫu (a pattern) hoặc tạo nó nếu không tồn tại. Sử dụng ON CREATE và ON MATCH để cập nhật có điều kiện.  Tạo 1 node có giá trị name ‘Alice’, gộp node này với các giá trị created, counter và accessTime vào node. |
| Kết quả: | |
| MATCH  (a:Person {name: $value1}),  (b:Person {name: $value2})  MERGE (a)-[r:LOVES]->(b)  VD:  MATCH  (a:Person {name: "Lillian Gish"}),  (b:Person {name:"Bebe Daniels"})  MERGE (a)-[r:LOVES]->(b) | MERGE tìm hoặc tạo Relationship giữa các Nodes.  Gộp 2 Person tìm thấy vào mối quan hệ LOVES. |
| Kết quả: | |
| MERGE (node:label) RETURN node  VD:  MERGE (node:User) RETURN node | MERGE node với label.  Trả về các node User. |
| Kết quả: | |
| MERGE (node:label {key1:value, key2:value, key3:value . . . . . . . . })  VD:  MERGE (node:User{name:"ngoctrang123" }) | MERGE node với properties.  Tạo 1 User có name ‘ngoctrang123’ |
| Kết quả: | |
| MATCH (a:Person {name: $value1})  Ví dụ:  MATCH (a:Person {name: 'Friedrich Feher'})  return a | MERGE tìm hoặc tạo các paths được gắn vào node.  Tìm node Persom có name ‘Friedrich Feher’ |
| Kết quả: | |

## **3.9. DELETE**

Sử dụng mệnh đề **Delete**, ta có thể xóa các nút và mối quan hệ khỏi cơ sở dữ liệu bằng cách sử dụng mệnh đề DELETE.

|  |  |
| --- | --- |
| **Cú pháp** | **Ý nghĩa** |
| MATCH (n) DETACH DELETE n | Xóa tất cả các Node và các Relationships liên quan trong cơ sở dữ liệu. |
| Kết quả: | |
| MATCH (node:label {properties . . . . . . })  DETACH DELETE node  VD:  MATCH (node:User{name:"Oscar Harmon" })  DETACH DELETE node | Xóa một Node cụ thể, bạn cần chỉ định chi tiết của Node.  Xóa User có name ‘Oscar Harmon’ |
| Kết quả: | |
| MATCH (n:Label)-[r:RelationShip]->(m:Label)  WHERE r.id = 123  DELETE r  VD:  MATCH (a:Person)-[l:LOVES]->(b:Person)  DELETE l | Xóa một relationship. |
| Kết quả: | |
| MATCH (n:Label)  WHERE n.id = 123  DETACH DELETE n  VD:  match (nxb:`NhàXuấtBản`)  where nxb.name='Tuổi Trẻ'  detach delete nxb | Xoá một Node và kèm theo cả các RelationShip của Node đó. |
| Kết quả: | |

## **3.10. REMOVE**

Mệnh đề REMOVE được sử dụng để xóa các thuộc tính và nhãn khỏi các thành phần đồ thị (Nút hoặc Mối quan hệ).

Sự khác biệt chính giữa các lệnh Neo4j CQL DELETE và REMOVE là

* + Thao tác DELETE được sử dụng để xóa các nút và các mối quan hệ liên quan.
  + Thao tác REMOVE được sử dụng để xóa nhãn và thuộc tính.

|  |  |
| --- | --- |
| **Cú pháp** | **Ý nghĩa** |
| MATCH (node:label{properties . . . . . . . })  REMOVE node.property  RETURN node  VD:  CREATE (nxbTT:NhàXuấtBản {name:'Kim Đồng',year:'2003'})  match (nxb:`NhàXuấtBản`)  remove nxb.year return nxb | Xóa thuộc tính (properties) của Node. |
| Kết quả: | |
| MATCH (node:label {properties . . . . . . . . })  REMOVE node:label  RETURN node  VD:  MATCH (n:User{name:'ngoctrang'})  REMOVE n:User  RETURN n | Xoá nhãn (label) khỏi Node hiện có. |
| Kết quả: | |
| MATCH (node:label1:label2 {properties . . . . . . . . })  REMOVE node:label1:label2  RETURN node  VD:  MATCH (n:Actor:Person {name:"Frank Bennett"})  REMOVE n:Actor:Person  RETURN n | Xóa nhiều nhãn (label) khỏi một nút hiện có. |
| Kết quả: | |

## **3.11. FOREACH**

Mệnh đề **FOREACH** được sử dụng để cập nhật dữ liệu trong danh sách cho dù là thành phần của đường dẫn hay kết quả tổng hợp.

|  |  |
| --- | --- |
| WITH ['Alice', 'Neo'] AS names  FOREACH ( value IN names | CREATE (:Person {name: value}) ) | Chạy một thao tác thay đổi cho từng thành phần trong danh sách. |
| Kết quả: | |

## **3.12. FUNCTION**

### 3.12.1. Aggregating Functions (Các hàm tổng hợp)

Giống như SQL, Neo4j CQL đã cung cấp một số hàm tổng hợp để sử dụng trong mệnh đề RETURN. Nó tương tự như mệnh đề GROUP BY trong SQL.

Chúng ta có thể sử dụng hàm RETURN + Aggregation này trong lệnh MATCH để làm việc trên một nhóm các nút và trả về một số giá trị tổng hợp.

|  |  |
| --- | --- |
| **Cú pháp** | **Ý nghĩa** |
| MATCH (m:Movie)  RETURN avg(m.imdbRating) | Hàm **avg** trả về giá trị trung bình của các giá trị int hoặc float. |
| Kết quả: | |
| MATCH (p:Person {name: 'Keanu Reeves'})-->(x)  RETURN labels(p), p.age, count(\*) | Hàm **count** trả về số lượng các giá trị hoặc các dòng. Khi **count(\*)** được sử dụng, hàm trả về số lượng của các dòng phù hợp. |
| Kết quả: | |
| MATCH (m:Movie)  RETURN count(m.imdbRating) | Hàm **count** cũng có thể được sử dụng thông qua biểu thức. Nếu vậy, nó trả về số lượng các giá trị khác null thông qua biểu thức. |
| Kết quả: | |
| MATCH (m:Movie)  RETURN max(m.imdbRating) | Hàm **max** trả về giá trị lớn nhất trong một tập giá trị. |
| Kết quả: | |
| MATCH (m:Movie)  RETURN min(m.imdbRating) | Hàm **min** trả về giá trị nhỏ nhất trong một tập giá trị. |
| Kết quả: | |
| MATCH (m:Movie)  RETURN sum(m.imdbRating) | Hàm **sum** trả về tổng của một tập hợp các giá trị số. |
| Kết quả: | |

### 3.12.2. String functions (Các hàm Chuỗi)

Giống như SQL, Neo4J CQL đã cung cấp một tập hợp các hàm Chuỗi để sử dụng chúng trong các Truy vấn CQL để có được kết quả cần thiết.

|  |  |
| --- | --- |
| **Cú pháp** | **Ý nghĩa** |
| RETURN replace("hello", "l", "w") | Hàm **replace** trả về STRING trong đó tất cả các lần xuất hiện của STRING được chỉ định trong STRING đã cho đã được thay thế bằng STRING thay thế (được chỉ định) khác. |
| Kết quả: | |
| RETURN reverse('palindrome') | Hàm **reverse** trả về STRING trong đó thứ tự của tất cả các ký tự trong STRING đã cho đã bị đảo ngược. |
| Kết quả: | |
| RETURN split('one,two', ',') | Hàm **split** trả về một LIST<STRING> do việc phân tách STRING đã cho xung quanh các kết quả trùng khớp của dấu phân cách đã cho. |
| Kết quả: | |
| RETURN substring('hello', 1, 3), substring('hello', 2) | Hàm **substring** trả về một chuỗi con của STRING đã cho, dựa vào giá trị vị trí phần tử bắt đầu |
| Kết quả: | |
| RETURN toLower('HELLO') | Hàm **toLower** trả về STRING đã cho ở dạng chữ thường. |
| Kết quả: | |
| RETURN toUpper('hello') | Hàm **toUpper** trả về STRING đã cho ở dạng chữ hoa. |
| Kết quả: | |

### 3.12.3. Temporal functions (Các hàm Thời gian)

|  |  |
| --- | --- |
| **Cú pháp** | **Ý nghĩa** |
| RETURN date() AS currentDate | Hàm **date** trả về giá trị DATE hiện tại. Nếu không có tham số múi giờ nào được chỉ định thì múi giờ địa phương sẽ được sử dụng. |
| Kết quả: | |
| RETURN datetime() AS currentDateTime | Hàm **datetime** trả về giá trị ZONED DATETIME hiện tại. Nếu không có tham số múi giờ nào được chỉ định thì múi giờ mặc định sẽ được sử dụng. |
| Kết quả: | |
| RETURN time() AS now | Hàm **time** trả về giá trị ZONED TIME hiện tại. Nếu không có tham số múi giờ nào được chỉ định thì múi giờ địa phương sẽ được sử dụng. |
| Kết quả: | |

## **3.13. UNIQUE CONSTRAINT**

Cơ sở dữ liệu Neo4j cũng hỗ trợ ràng buộc UNIQUE đối với các thuộc tính nút hoặc mối quan hệ. Ràng buộc UNIQUE được sử dụng để tránh các bản ghi trùng lặp và để thực thi quy tắc toàn vẹn dữ liệu.

|  |  |
| --- | --- |
| **Cú pháp** | **Ý nghĩa** |
| SHOW CONSTRAINT | Xem tất cả ràng buộc trong Neo4j |
| Kết quả: | |
| CREATE CONSTRAINT constraint\_name FOR (node:label) REQUIRE node.id IS UNIQUE  Ví dụ:  CREATE CONSTRAINT Test FOR (n:HS) REQUIRE n.id IS UNIQUE | Tạo các ràng buộc duy nhất trên các thuộc tính nút hoặc mối quan hệ. |
| Kết quả: | |
| DROP CONSTRAINT constraint\_name Ví dụ:  DROP CONSTRAINT Test | Xóa ràng buộc Duy nhất hiện có khỏi một nút hoặc thuộc tính mối quan hệ. |
| Kết quả: | |

# **CHƯƠNG 4. XÂY DỰNG ỨNG DỤNG KẾT NỐI NEO4J**

## **4.1. Mô tả dữ liệu sử dụng trên Neo4j**

### 4.1.1. Mô tả dữ liệu

The Open Movie Database: Nhóm đã sử dụng “Recommendations” chứa tập dữ liệu xếp hạng người dùng MovieLens cùng với dữ liệu phim và diễn viên/ đạo diễn từ <https://themoviedb.org>.

Dataset bao gồm:

- 28,863 nodes

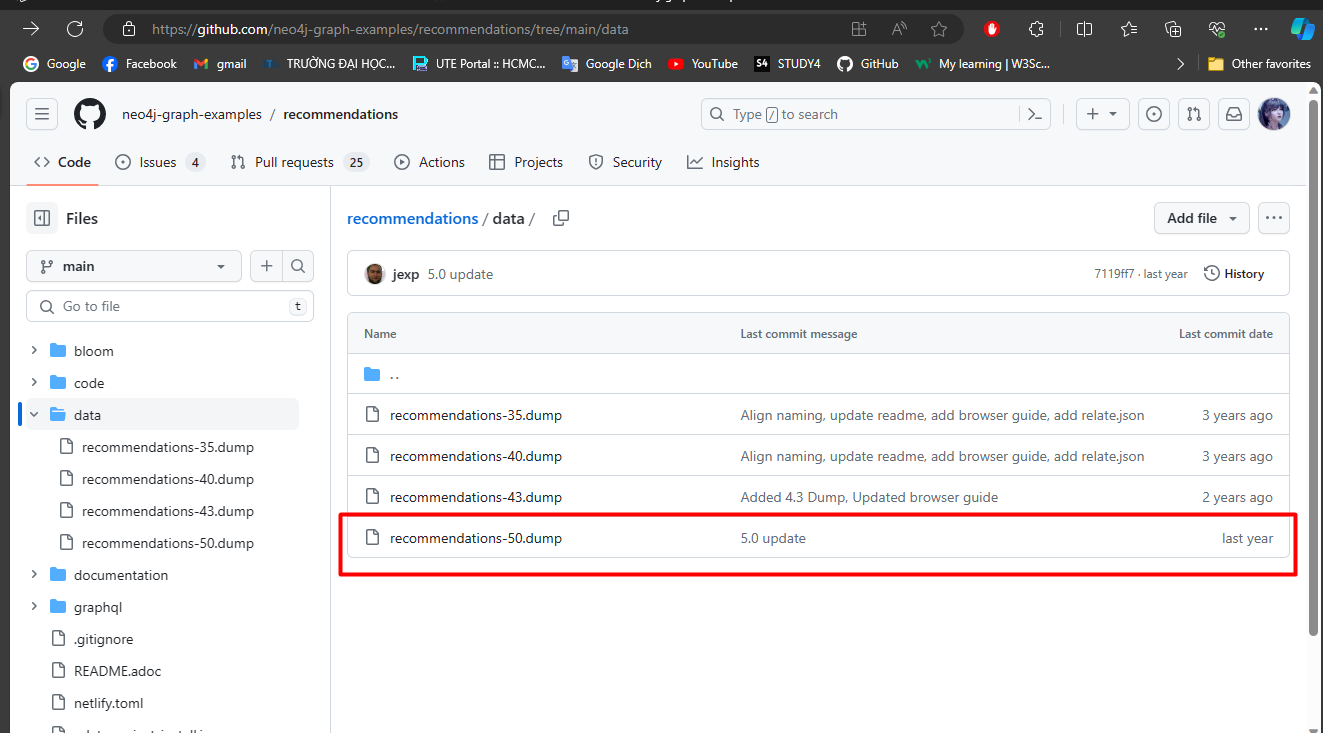
- 166,261 relationships

Trong đó:

- Node label bao gồm: Actor, Director, Genre, Movie, Person, User

- Relationship types bao gồm: ACTED\_IN, DIRECTED, IN\_GENRE, RATED

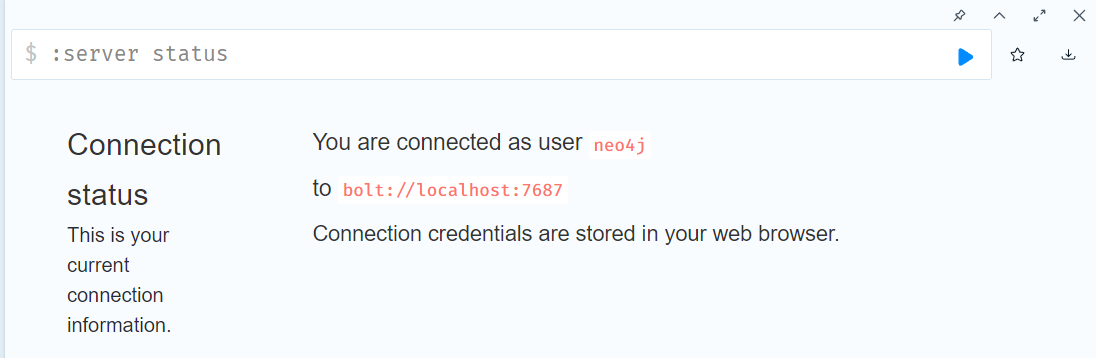
Truy cập vào đường link github để lấy dữ liệu: [recommendations/data at main · neo4j-graph-examples/recommendations (github.com)](https://github.com/neo4j-graph-examples/recommendations/tree/main/data)

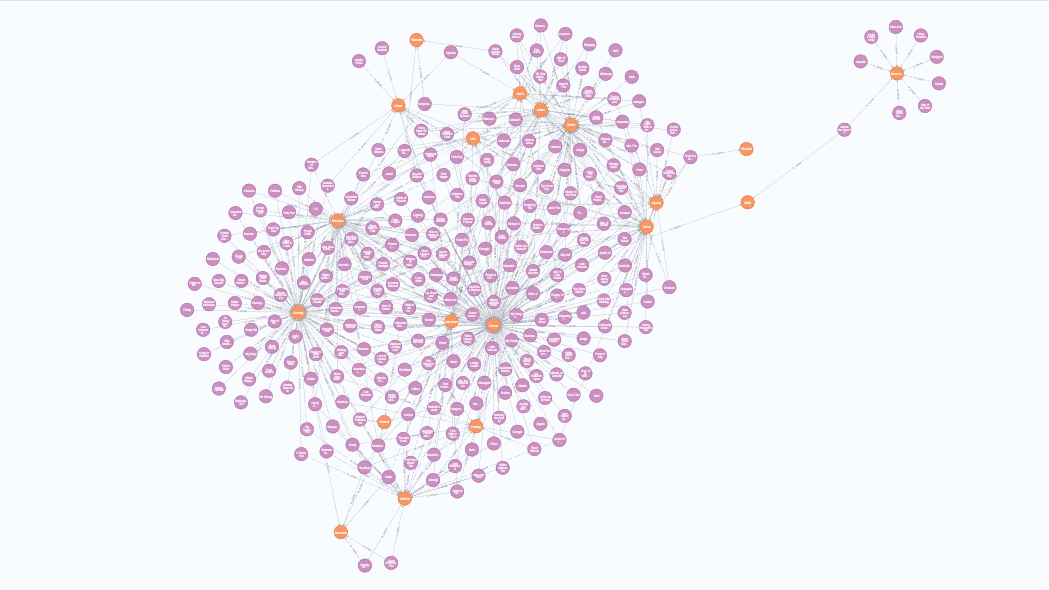


### 4.1.2. Kết nối Cơ Sở Dữ Liệu

Để kết nối vào cơ sở dữ liệu này, nhóm sử dụng một số tham số như sau:

|  |
| --- |
| #global variables  uri = "bolt://localhost:7687"  user = "neo4j"  password = "12345678" # modify this accordingly based on your own password    neo\_db = recommender.Recommender(uri, user, password) |



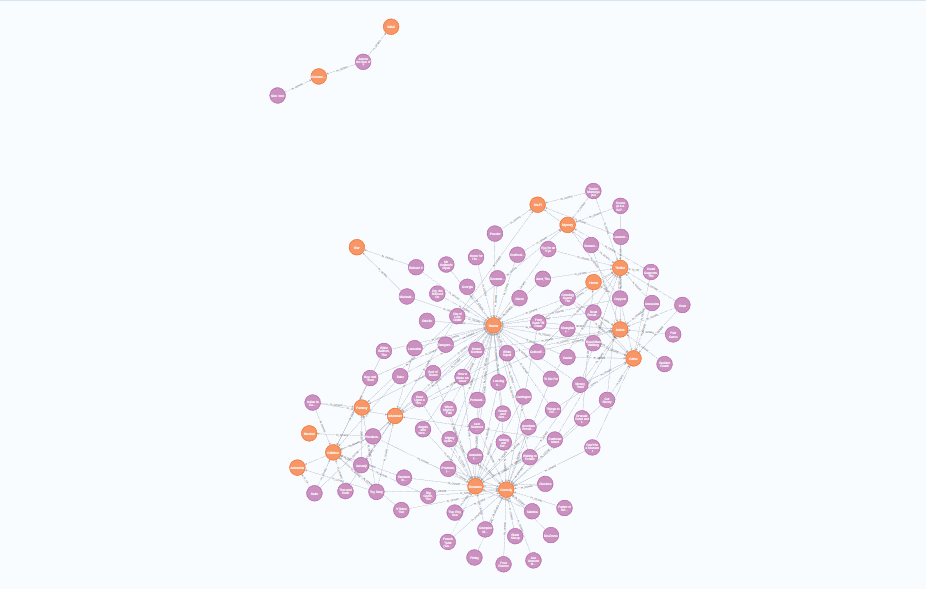


## **4.2. Xây dựng ứng dụng kết nối với Neo4j**

### 4.2.1. Khai phá dữ liệu bằng đồ thị

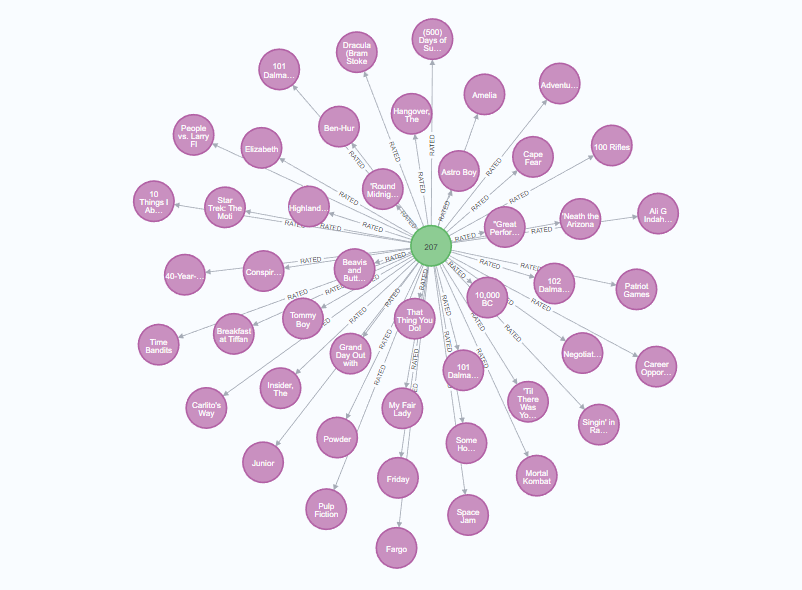
* Thông tin toàn bộ database:

|  |
| --- |
| match (n) return n limit 100 |



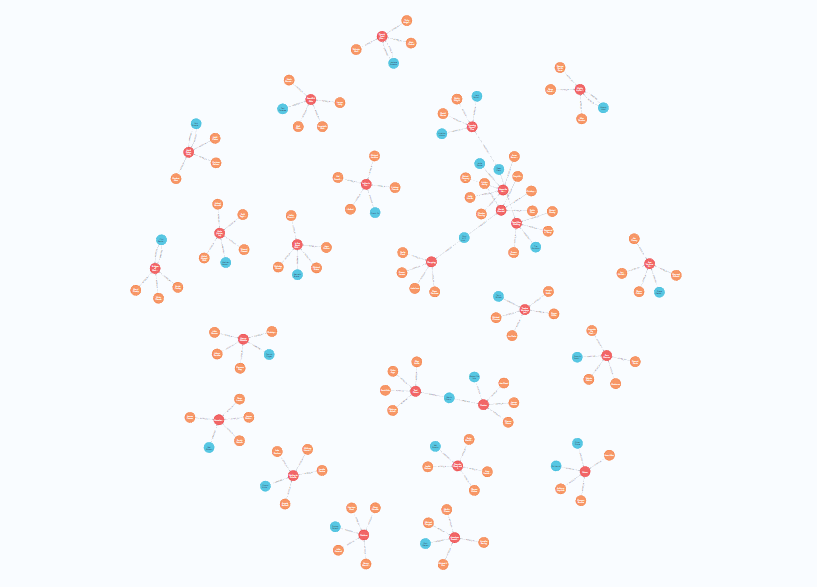
* Kiểm tra tất cả lượt rating của một user bất kỳ theo name của user:

|  |
| --- |
| // Show all ratings by Patrick Hutchinson  MATCH (u:User {name: "Patrick Hutchinson"})  MATCH (u)-[r:RATED]->(m:Movie)  RETURN \* LIMIT 100; |



* Kiểm tra thông tin của một bộ film với thông tin của actor và director

|  |
| --- |
| MATCH p=(:Actor)-[:ACTED\_IN]-(:Movie)-[:DIRECTED]-(:Actor)  RETURN p LIMIT 100; |



### 4.2.2. Mô tả ứng dụng

Ứng dụng được viết bằng ngôn ngữ Python, sử dụng framework FastAPI với các thư viện neo4j (GraphDatabase), và neo4j.exceptions (ServiceUnavailable)

#### a. Framework FastAPI

FastAPI là framework hiện đại, hiệu suất cao để xây dựng web API với Python 3.7+ dựa trên các tiêu chuẩn của Python. Nó giúp các nhà phát triển code ít hơn, triển khai nhanh hơn và hỗ trợ docs API đầy đủ, cũng cải thiện tốc độ khi tích hợp API vì đã support docs API đầy đủ. Sẽ là một sự lựa chọn hoàn hảo cho một dự án cần triển khai trong thời gian ngắn nhưng vẫn đảm bảo đầy đủ các yếu tố đi kèm.

Cài đặt thư viện chính:

|  |
| --- |
| pip install fastapi |

Cần ASGI server khi deploy sản phẩm như Uvicorn

|  |
| --- |
| pip install uvicorn |

Do based trên OpenAI mà trước đó có tên là Swagger nên FastAPI cung cấp doc có giao diện dễ nhìn, dễ sử dụng. Ví dụ minh họa: Bật doc bằng local url <http://127.0.0.0:8000/docs>

**b. neo4j (GraphDatabase)**

Graph database là một hệ quản trị cơ sở dữ liệu được thiết kế đặc biệt để lưu trữ và truy vấn dữ liệu theo mô hình đồ thị. Trong mô hình đồ thị, dữ liệu được biểu diễn dưới dạng các đỉnh (nodes) và các cạnh (edges) để mô tả các mối quan hệ giữa các đối tượng.

Trong Python, có một số thư viện đáng chú ý để làm việc với graph database, trong đó Neo4j thư viện khá phổ biến. Thư viện Neo4j Python cung cấp một API phong phú giúp bạn tạo, truy vấn và cập nhật dữ liệu trên Neo4j.

#### c. neo4j.exceptions (ServiceUnavailable)

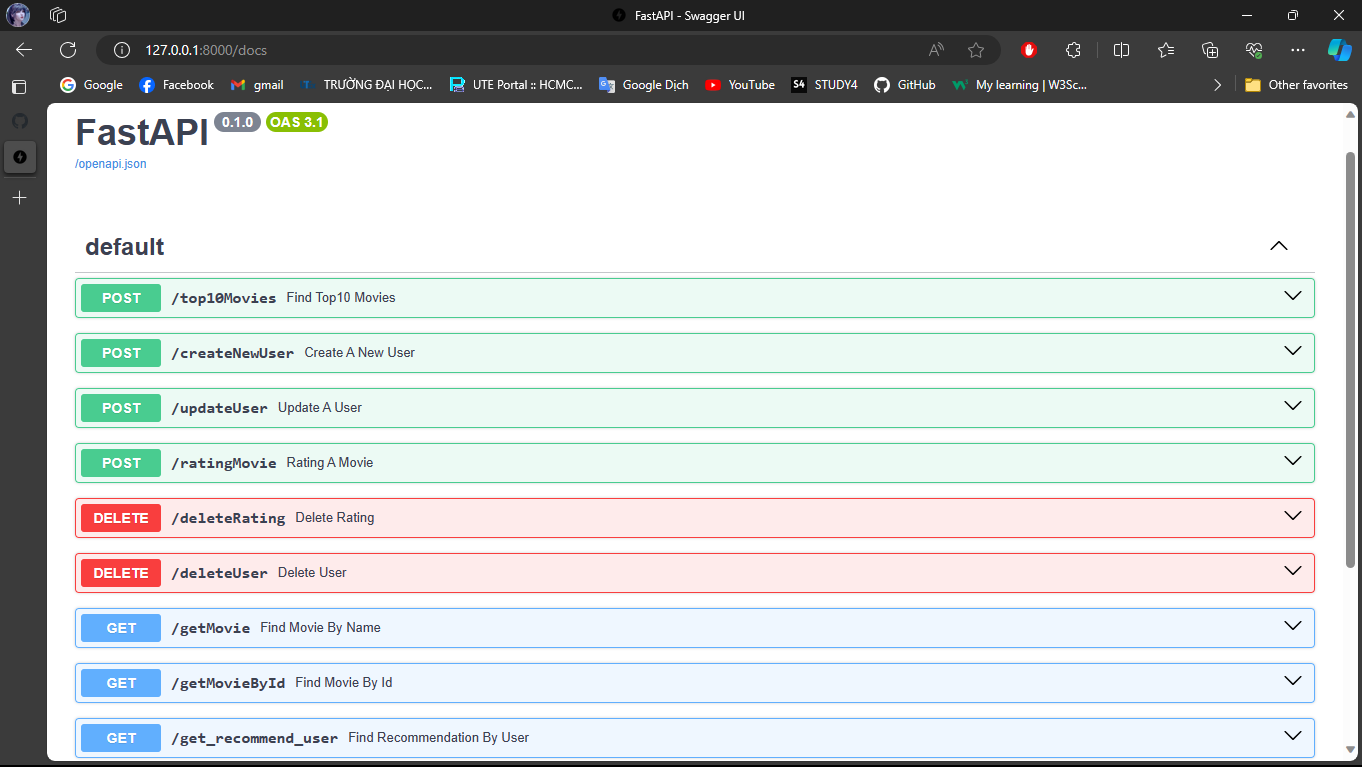
Thư viện neo4j.exceptions trong Python là một phần của gói neo4j và cung cấp các lớp ngoại lệ (exceptions) liên quan đến việc làm việc với Neo4j, một hệ quản trị cơ sở dữ liệu đồ thị phổ biến.

Trong đó, ServiceUnavailable là một lớp ngoại lệ đặc biệt trong neo4j.exceptions và được sử dụng khi có sự cố xảy ra trong quá trình kết nối với Neo4j server hoặc khi Neo4j server không khả dụng.

Khi sử dụng neo4j trong Python, bạn có thể sử dụng ServiceUnavailable exception để xử lý các trường hợp lỗi kết nối hoặc không thể truy cập vào Neo4j server.

### 4.2.3. Xây dựng API để tương tác

|  |
| --- |
| from unittest import result  from fastapi import FastAPI  import recommender  import atexit  #global variables  uri = "bolt://localhost:7687"  user = "neo4j"  password = "12345678" # modify this accordingly based on your own password    neo\_db = recommender.Recommender(uri, user, password)  app = FastAPI()  # Top 10 phim  @app.post('/top10Movies')  async def find\_top10\_movies():      result = neo\_db.findTop10Movies()      return result  # Tạo user với tên và id  @app.post('/createNewUser')  async def create\_a\_new\_user(name, custom\_id):      result = neo\_db.createUser(name, custom\_id)      return result  # Cập Nhật tên user với id  @app.post('/updateUser')  async def update\_a\_user(name, custom\_id):      result = neo\_db.updateUser(name, custom\_id)      return result  # Tạo relationship rating giữa user và movie  @app.post('/ratingMovie')  async def rating\_a\_movie(userId, movieId, rating):      result = neo\_db.ratedMovie(userId, movieId, rating)      return result  # Delete relationship user rating movie  @app.delete('/deleteRating')  async def delete\_rating(userId, movieId):      result = neo\_db.deleteRating(userId, movieId)      return result  # Delete User (Node)  @app.delete('/deleteUser')  async def delete\_User(userId):      result = neo\_db.deleteUser(userId)      return result  # Tìm bộ phim có chưa chuỗi được nhập vào  @app.get('/getMovie')  async def find\_movie\_by\_name(movie\_name):      result = neo\_db.findMovie(movie\_name)      return result  # Tìm bộ phim, đạo diễn và diễn phim bằng id phim  @app.get('/getMovieById')  async def find\_movie\_by\_Id(id\_movie):      result = neo\_db.findMovieById(id\_movie)      return result  # giới thiệu phim mà user chưa xem  @app.get('/get\_recommend\_user')  async def find\_recommendation\_by\_user(user\_name):      result = neo\_db.findRecommendationUser(user\_name)      return result |



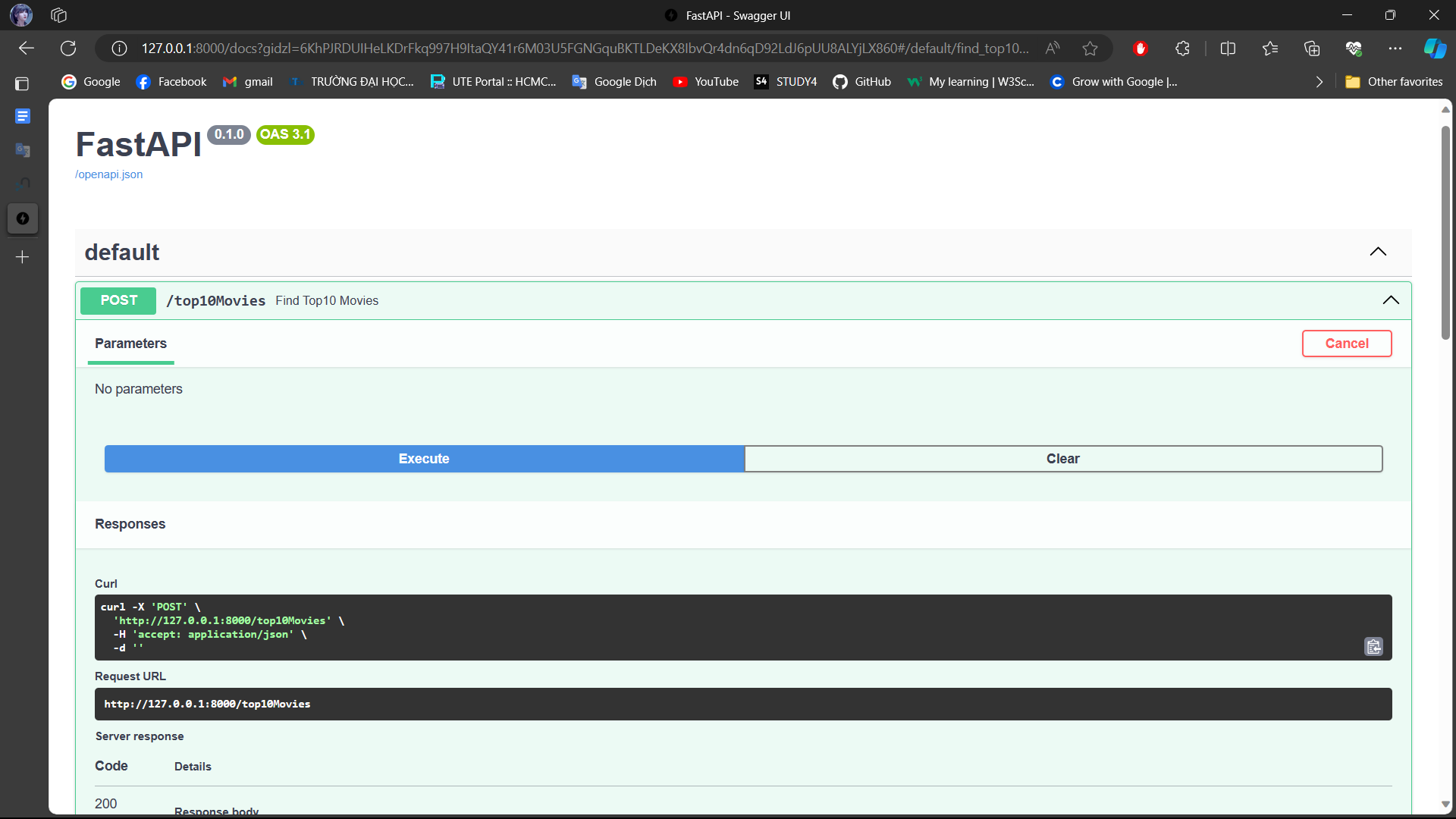
#### a. API Tìm Danh Sách Top10 phim sắp xếp theo Doanh Thu giảm dần

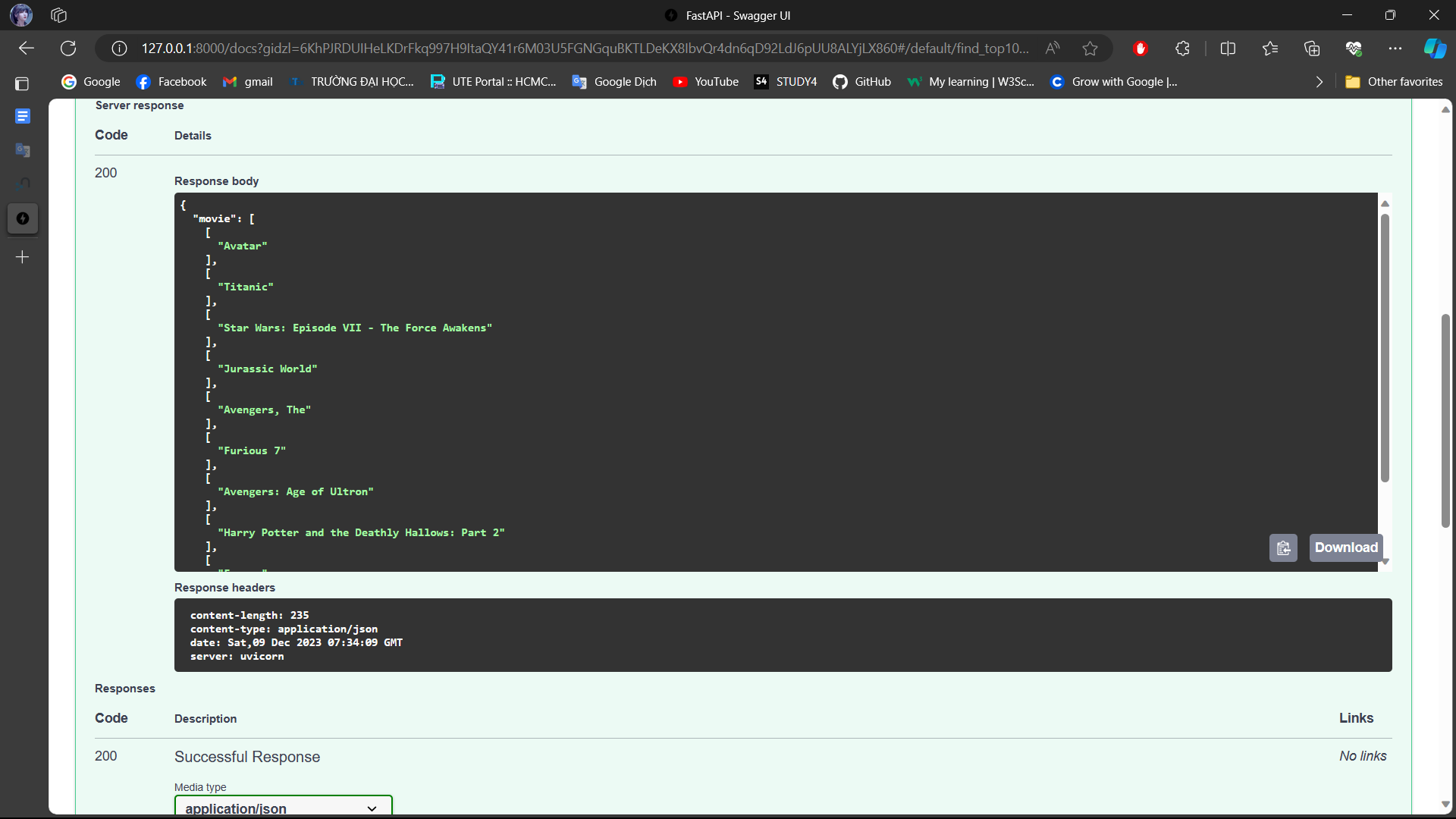
Dùng để truy vấn xem 10 bộ phim có doanh thu cao nhất trong database (xếp từ cao đến thấp).

Code:

|  |
| --- |
| # Tìm top10 phim được xếp theo doanh thu giảm dần      def findTop10Movies(self):          with self.driver.session() as session:              result = session.read\_transaction(self.\_\_findTop10AndReturnMovie)              return {"movie": result}        @staticmethod      #Tìm top10 phim theo doanh thu giảm dần      def \_\_findTop10AndReturnMovie(tx):          query = (              """MATCH (n:Movie)                  RETURN n.title                  ORDER BY -n.revenue  LIMIT 10"""          )          try:              result = tx.run(query)              return [ {record["n.title"]} for record in result]          except ServiceUnavailable as exception:              logging.error("{query} raised an error: \n {exception}".format(query=query, exception=exception))              raise |

Kết quả:





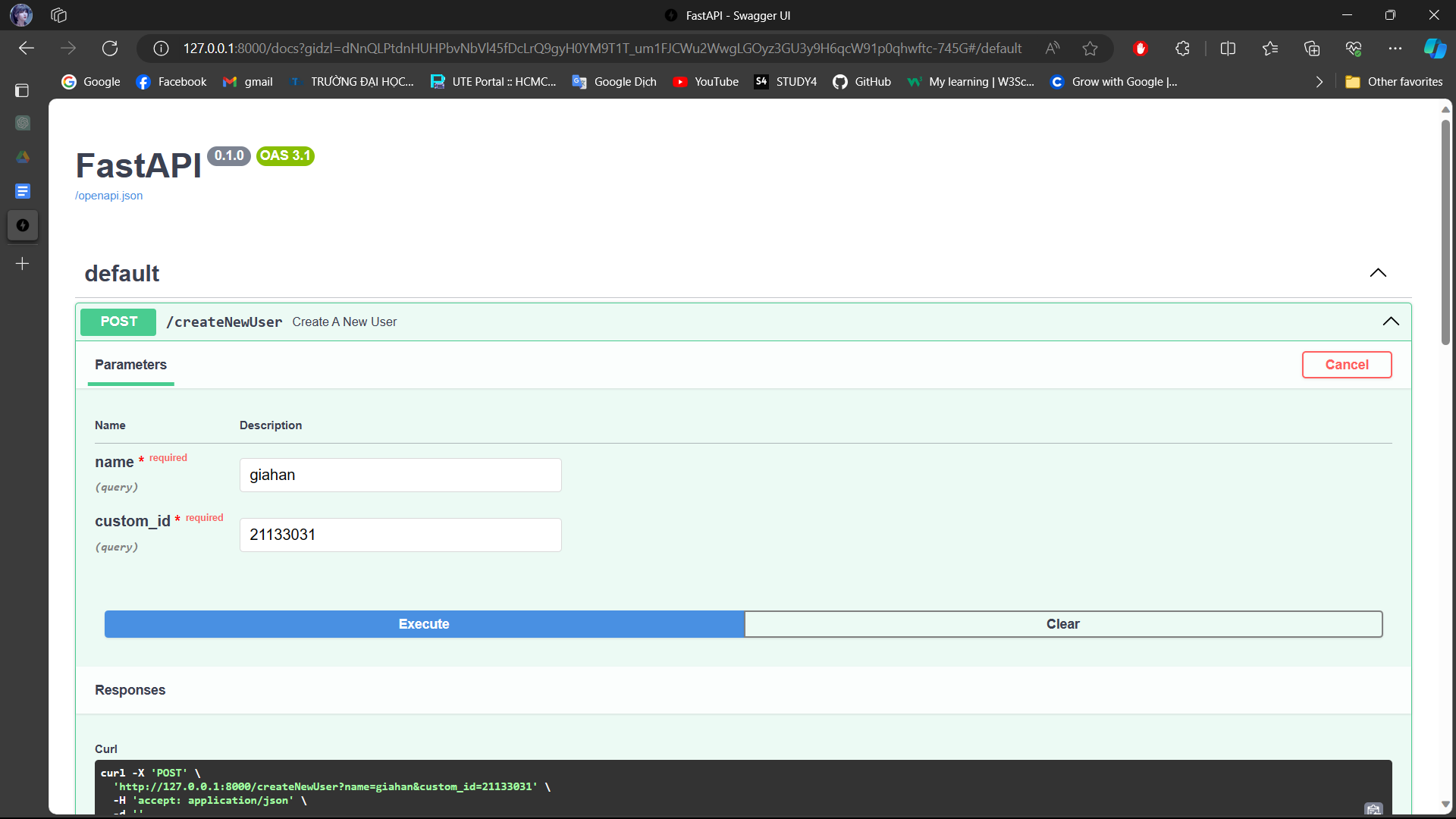
#### b. API Tạo thông tin một User mới

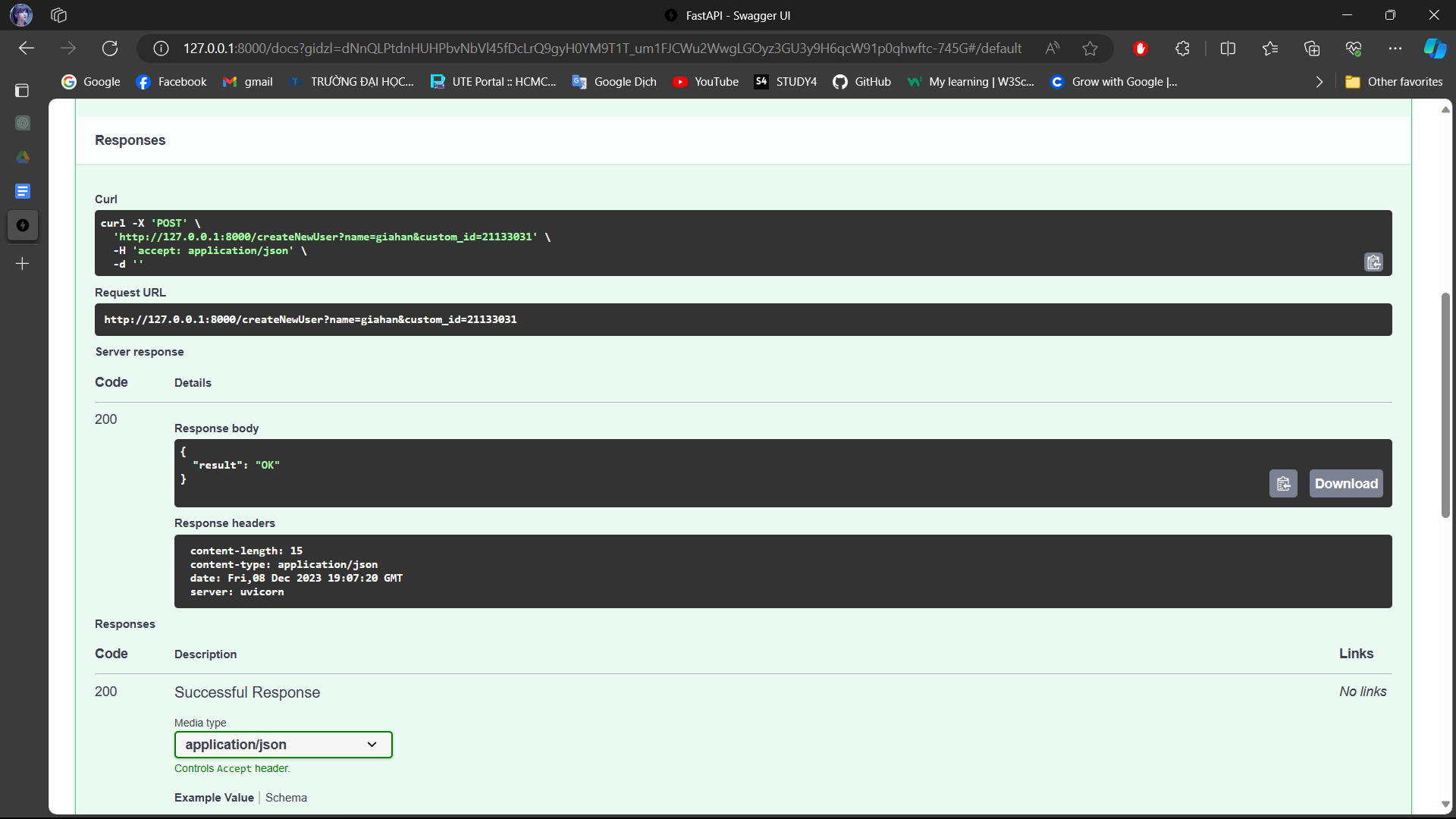
Dùng để tạo một node User mới trong database khi cung cấp một số thuộc tính như name và ID.

Code:

|  |
| --- |
| # Tạo 1 User mới      def createUser(self, name, id):          with self.driver.session() as session:              session.write\_transaction(self.\_\_createUser, name, id)              return {"result": "OK"}        @staticmethod      def \_\_createUser(tx, name\_user, userID):          query = ('''CREATE (:User {name: $name\_user, userId: $userID})''')          try:              tx.run(query, name\_user = name\_user, userID = userID).data()              return "Created"          except ServiceUnavailable as exception:              logging.error("{query} raised an error: \n {exception}".format(query=query, exception=exception))              raise |

Kết quả:





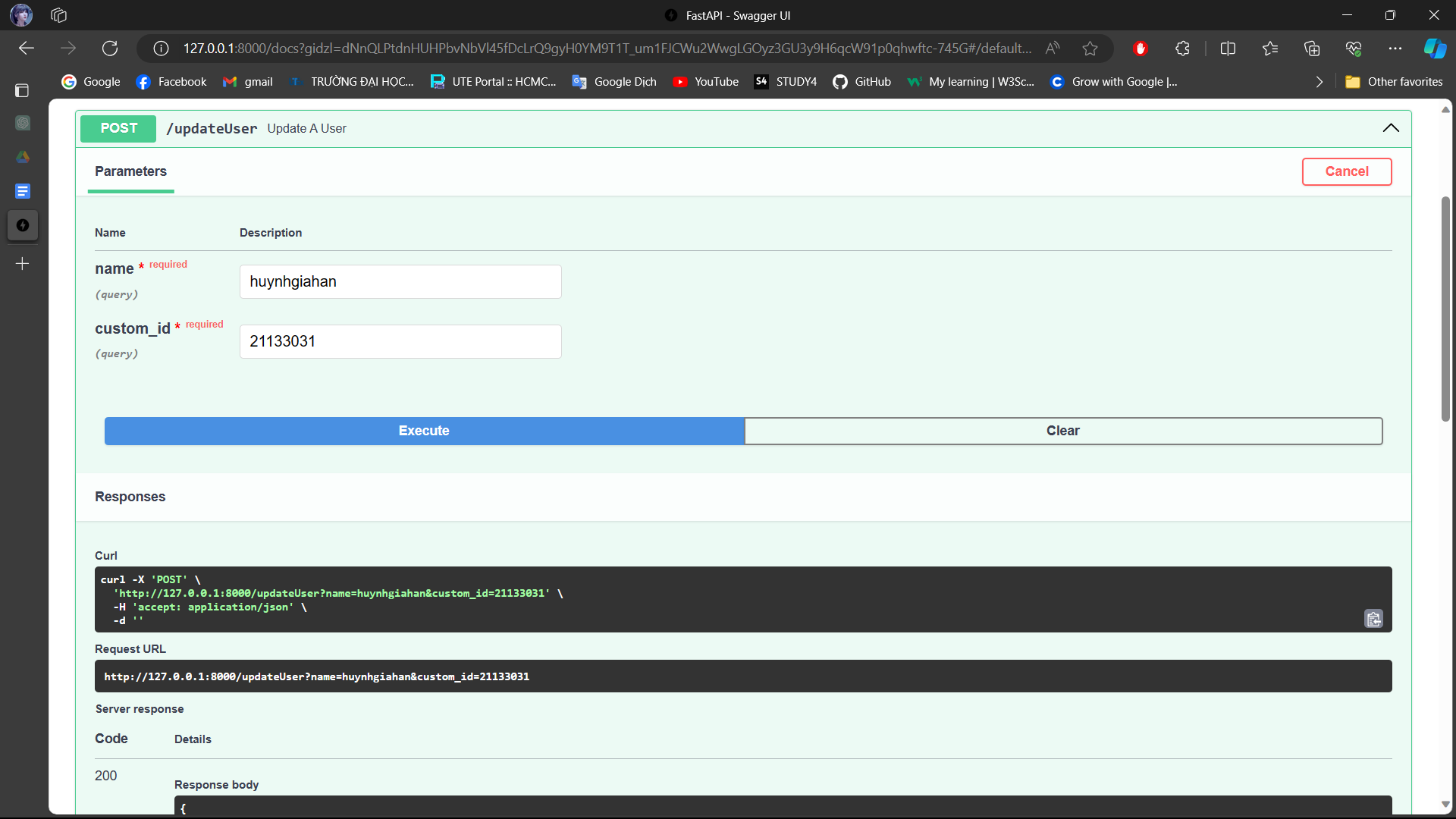
#### c. API Chỉnh sửa thông tin name của User

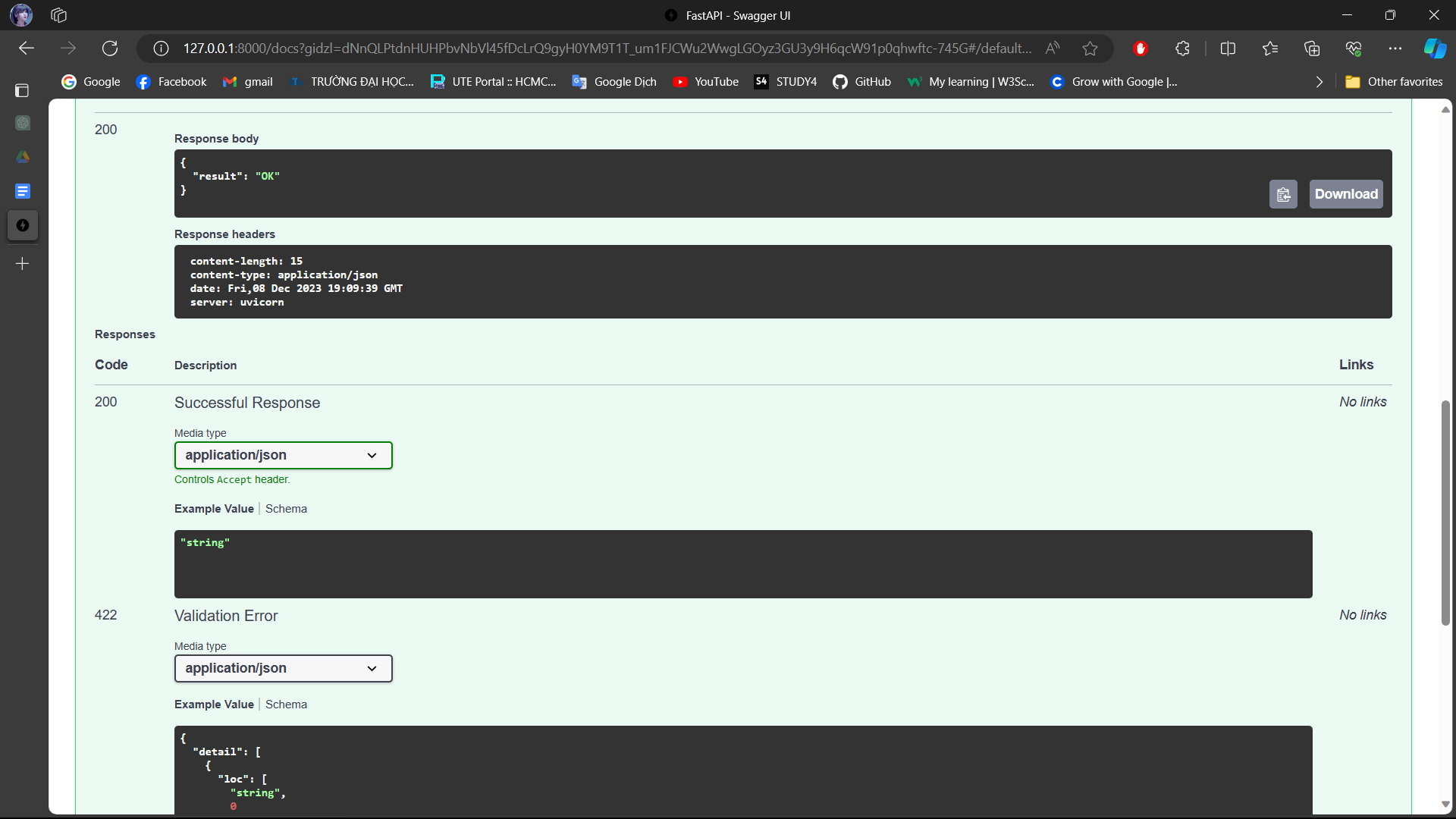
Khi người dùng cần chỉnh sửa thông tin tên của mình trên database, chỉ cần nhập tên mới và nhập lại mã User.

Code:

|  |
| --- |
| # Chỉnh Sửa thông tin User      def updateUser(self, name, id):          with self.driver.session() as session:              session.write\_transaction(self.\_\_updateUser, name, id)              return {"result": "OK"}        @staticmethod      def \_\_updateUser(tx, name\_user, userID):          query = ('''MATCH (n:User)                      WHERE n.userId = $userID                      SET n.name = $name\_user ''')          try:              tx.run(query, name\_user = name\_user, userID = userID).data()              return "Updated"          except ServiceUnavailable as exception:              logging.error("{query} raised an error: \n {exception}".format(query=query, exception=exception))              raise |

Kết quả:





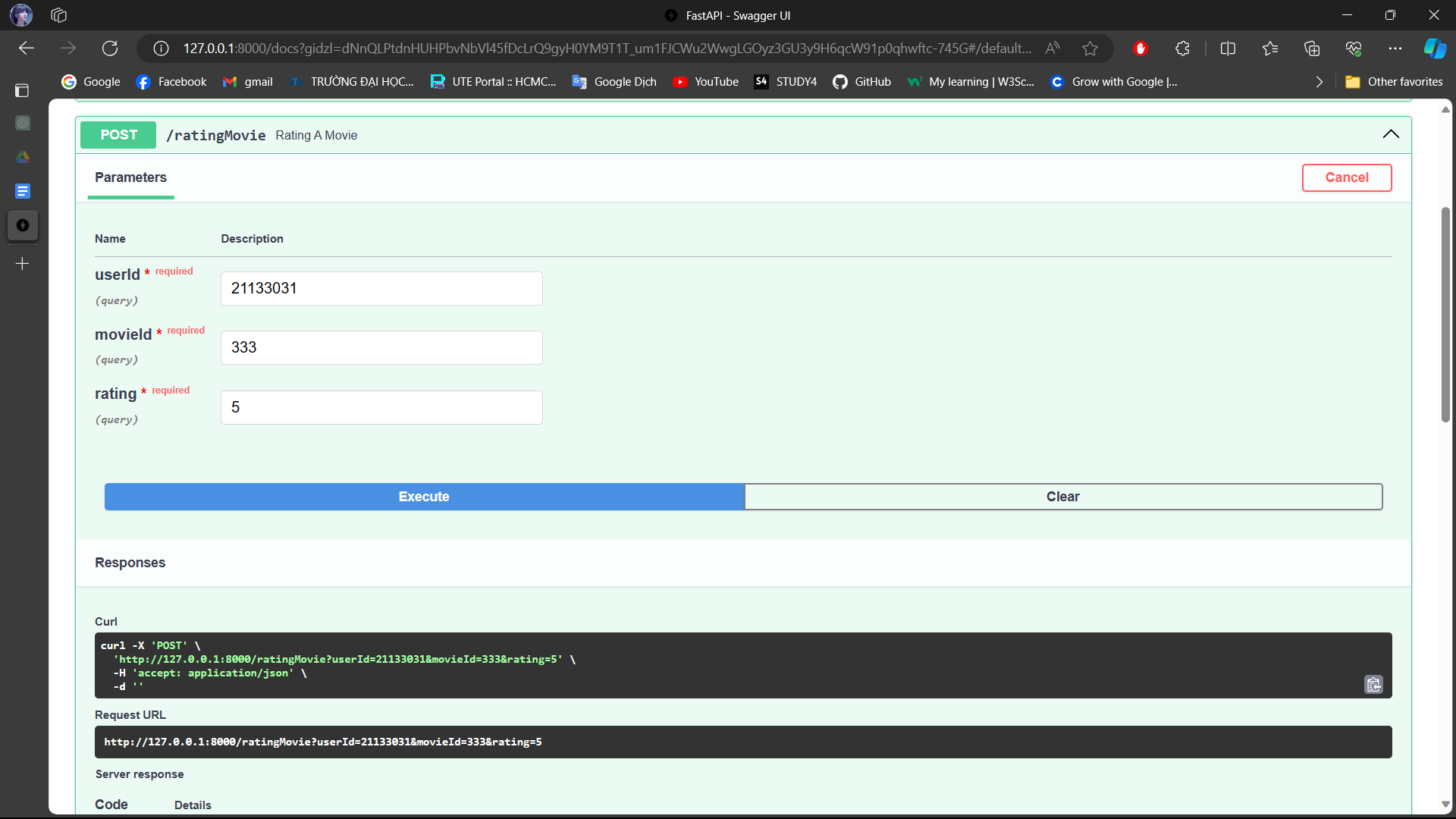
#### d. API Tạo đánh giá cho một bộ phim từ một User

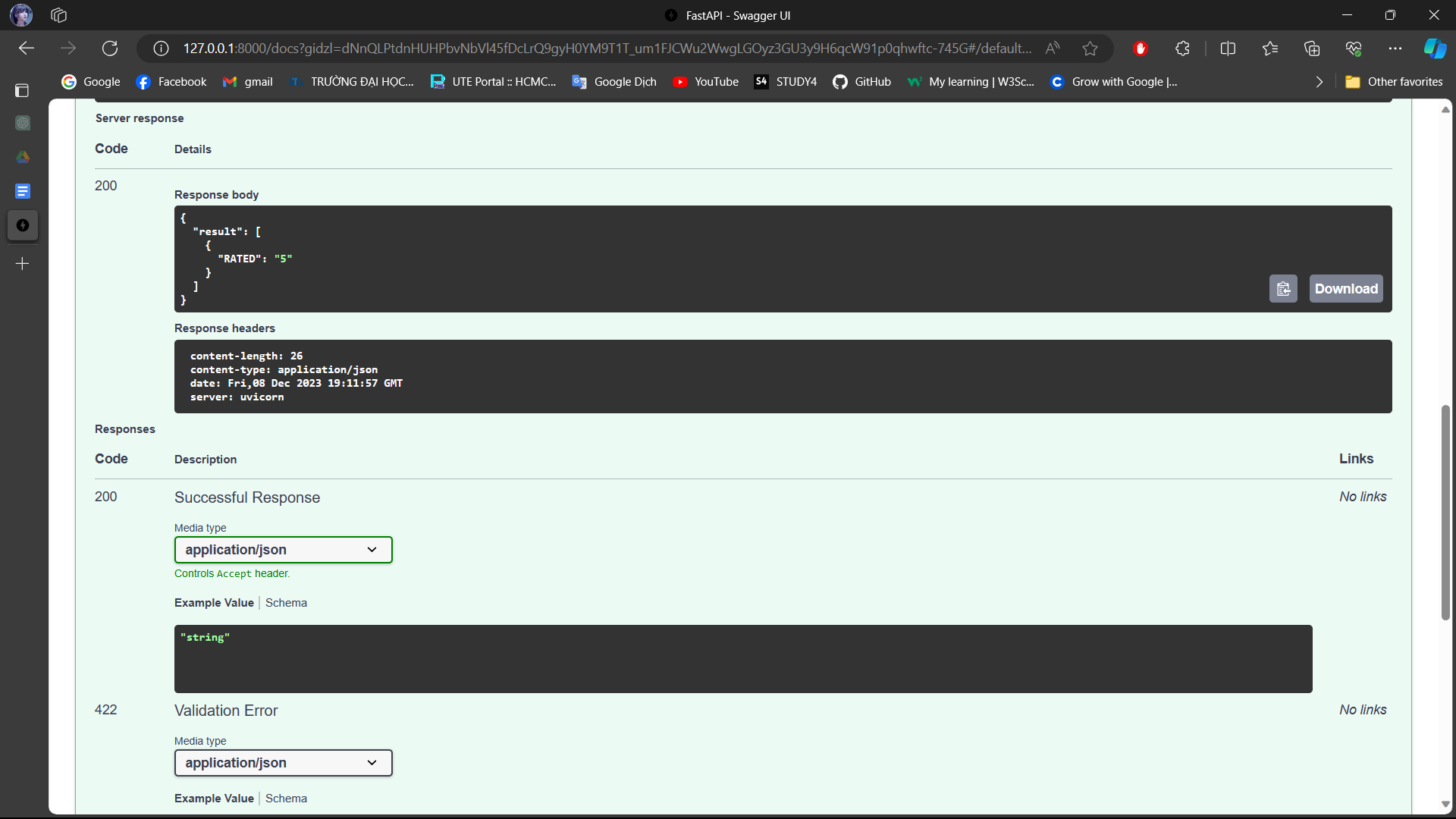
Khi người dung cần đánh giá một bộ phim, chỉ cần cung cấp userId, movieId và rating để có thể them đánh giá vào database.

Code:

|  |
| --- |
| # Tạo quan hệ User đánh giá một bộ phim theo Id phim      def ratedMovie(self, userId, movieId, rating):          with self.driver.session() as session:              result = session.write\_transaction(self.\_\_makeRatingMovie, userId, movieId, rating)              return {"result": result}        @staticmethod      def \_\_makeRatingMovie(tx, userId, movieId, rating):          # current date and time          now = datetime.now()          # time is now          timestamp = round(datetime.timestamp(now))          query = ("""MATCH(u:User),(m:Movie)                  WHERE u.userId = $userId AND m.movieId = $movieId                  CREATE (u)-[r:RATED {rating: $rating  , timestamp:$stamp}]->(m)                  RETURN type(r) as relation , r.rating as rating""")          try:              result = tx.run(query, userId=userId, movieId=movieId, rating = rating, stamp = timestamp)              return [ {record["relation"]: record["rating"]} for record in result]          except ServiceUnavailable as exception:              logging.error("{query} raised an error: \n {exception}".format(query=query, exception=exception))              raise |

Kết quả:





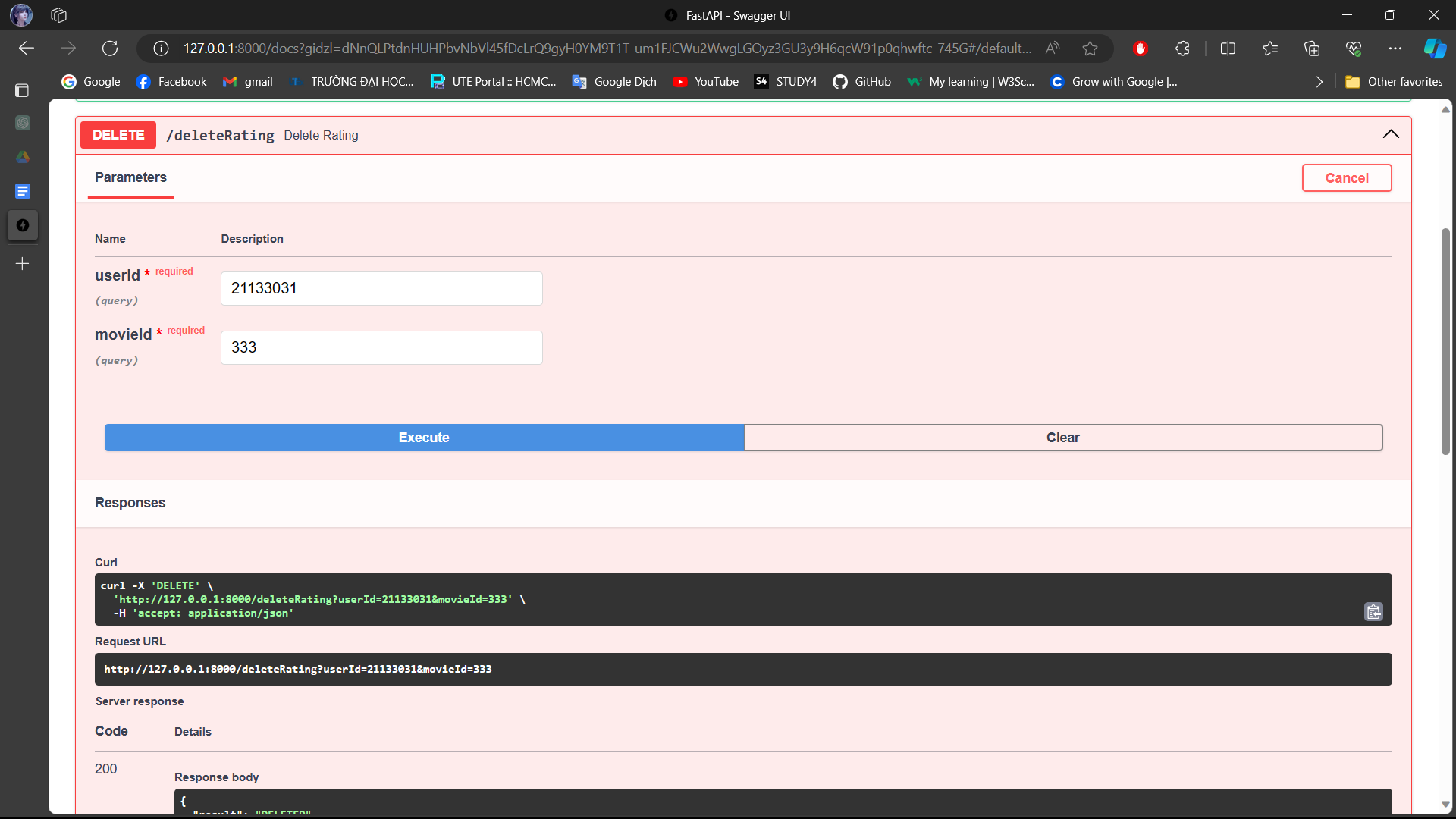
#### e. API Xóa Rating của một User từ một bộ phim

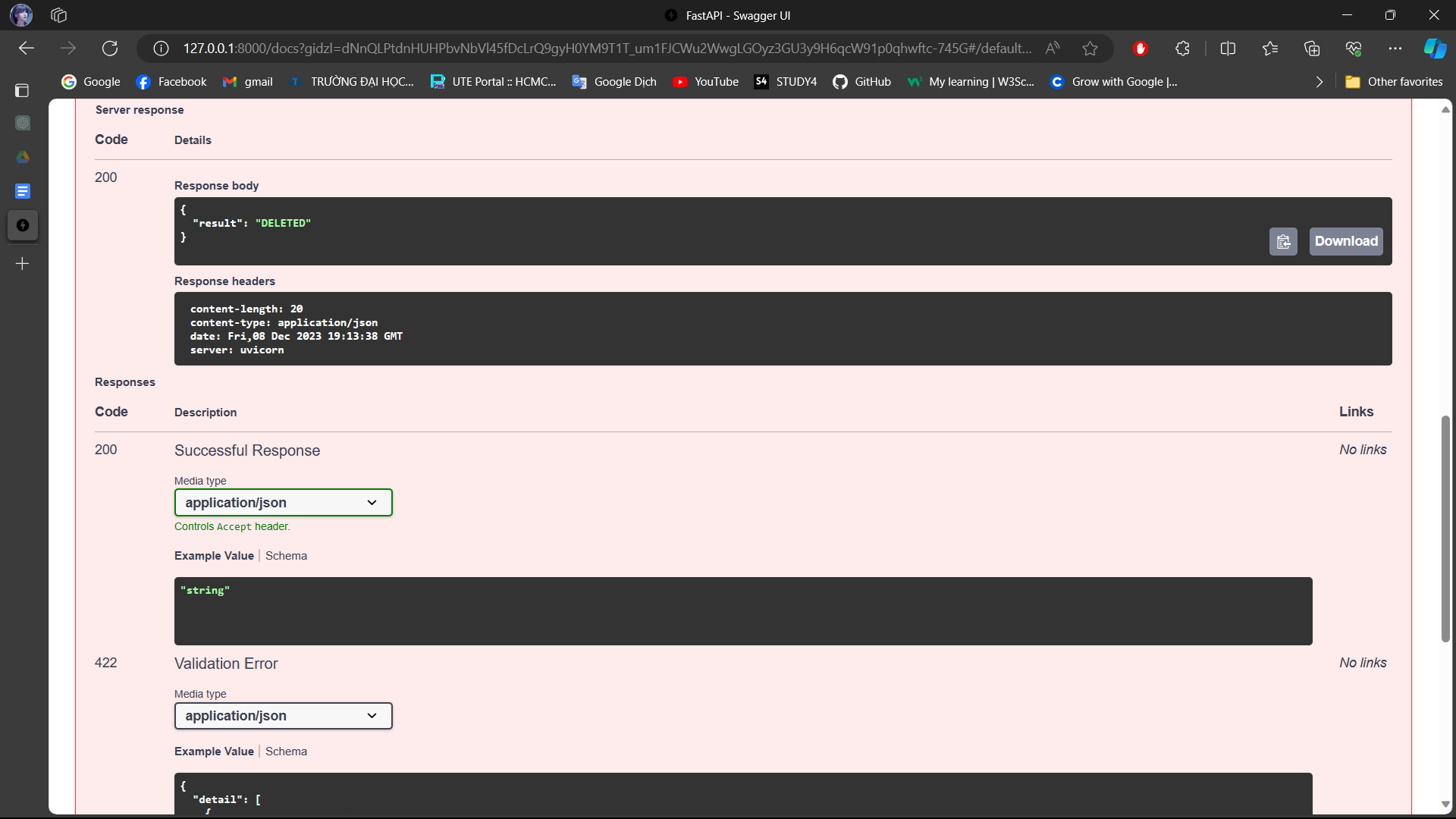
Người dùng (user) cần cung cấp userId và movieId để có thể xóa lượt rating tới bộ phim đó.

Code:

|  |
| --- |
| # Xóa quan hệ đánh giá 1 bộ phim của User      def deleteRating(self, userId, movieId):          with self.driver.session() as session:              result = session.write\_transaction(self.\_\_deleteRating, userId, movieId)              return {"result": "DELETED"}        @staticmethod      def \_\_deleteRating(tx, userId, movieId):          query = ("""MATCH (u:User)-[r:RATED]->(m:Movie)                      WHERE u.userId = $userId AND m.movieId = $movieId                      DELETE r                  """)          try:              result = tx.run(query, userId=userId, movieId = movieId)              return [record for record in result]          except ServiceUnavailable as exception:              logging.error("delete rating: {query} raised an error: \n {exception}".format(query=query, exception=exception))              raise |

Kết quả:





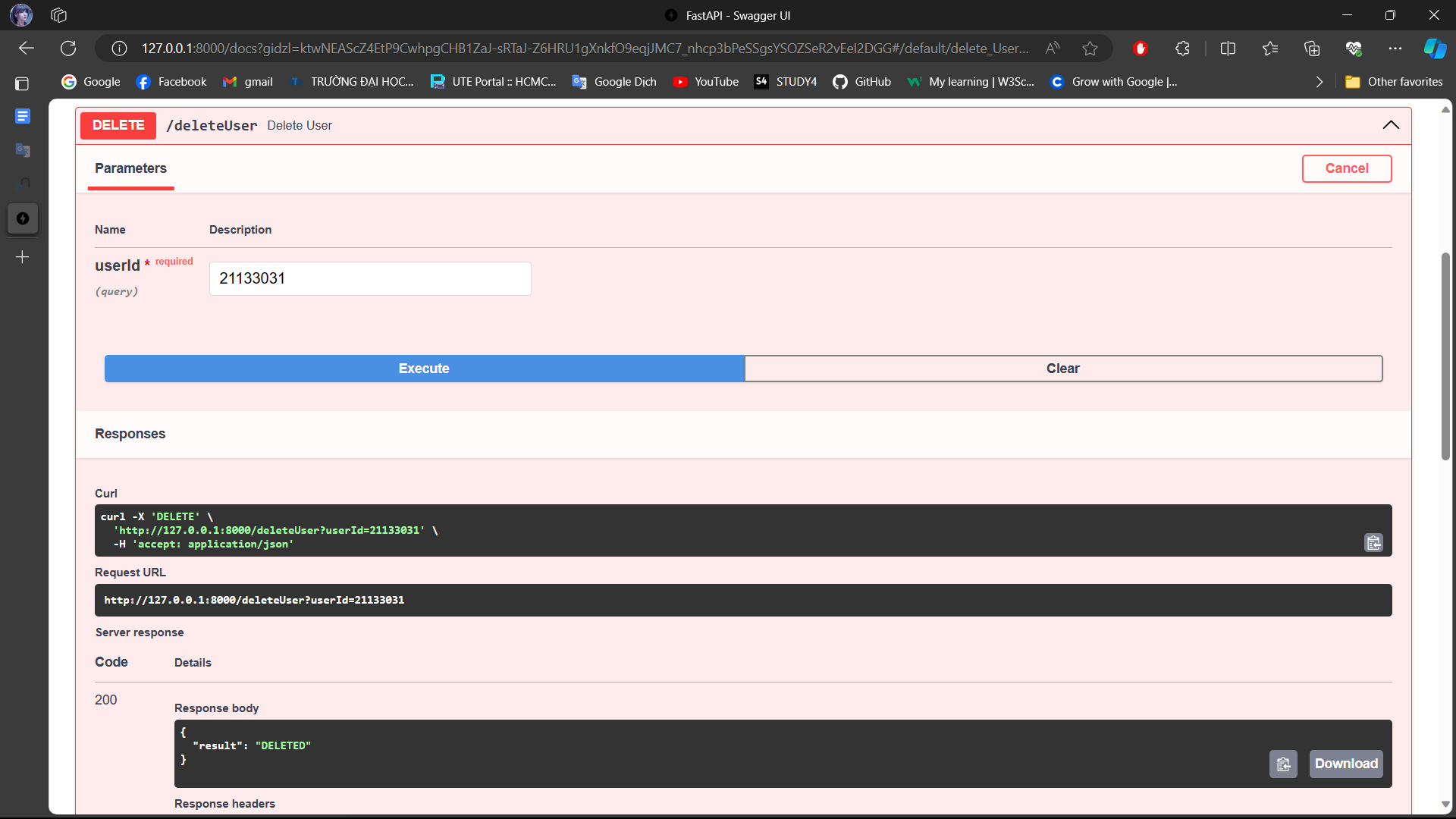
#### f. API Xóa một User dựa vào Id của User đó

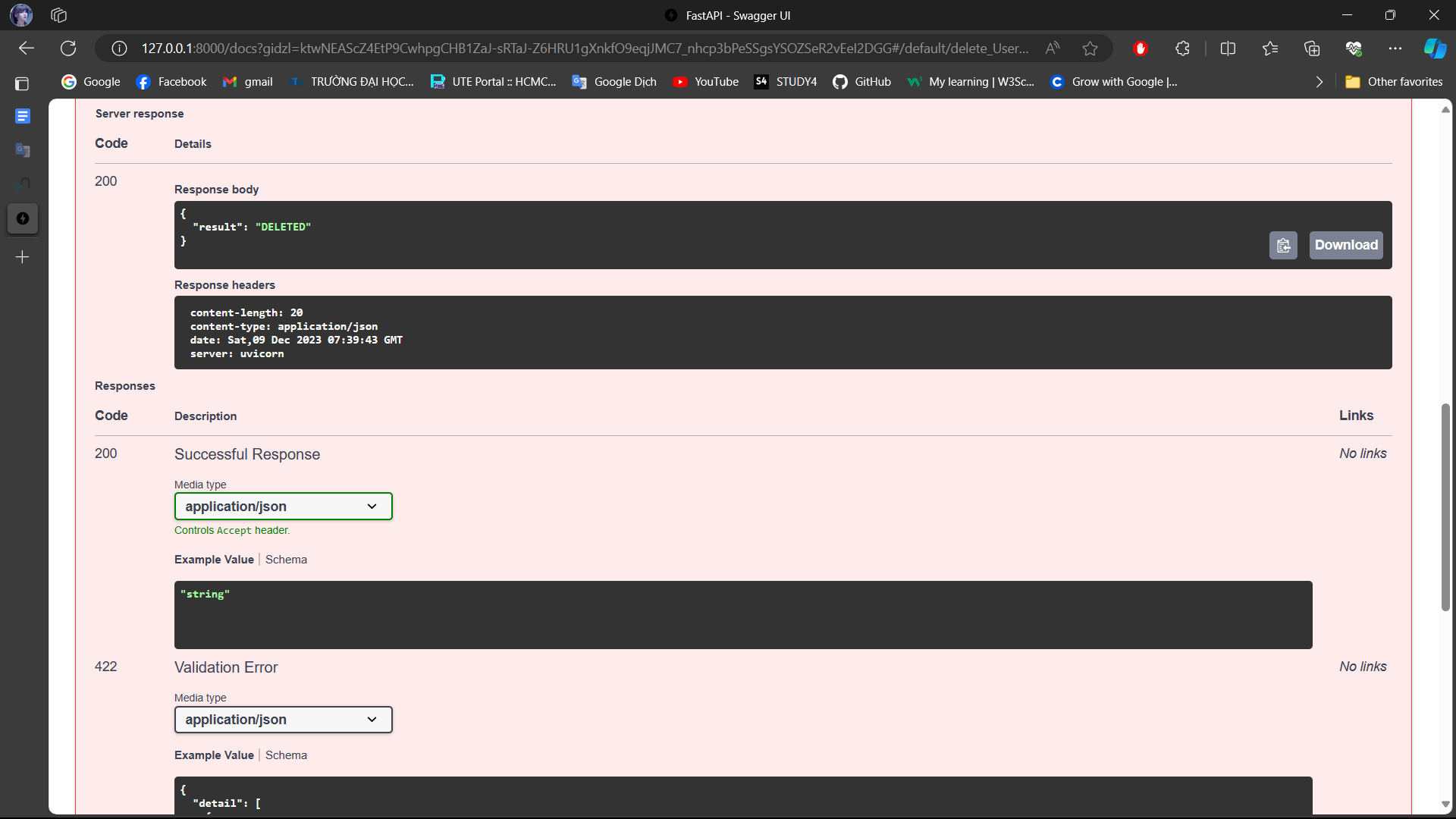
Dùng để xóa node User ra khỏi database dựa trên thuộc tính UserID của User cần xóa.

Code:

|  |
| --- |
| # Xóa 1 User      def deleteUser(self, userId):          with self.driver.session() as session:              result = session.write\_transaction(self.\_\_deleteUser, userId)              return {"result": "DELETED"}        @staticmethod      def \_\_deleteUser(tx, userId):          query = ("""MATCH (n:User) WHERE n.userId= $userId DETACH DELETE n """)          try:              result = tx.run(query, userId=userId)              return [record for record in result]          except ServiceUnavailable as exception:              logging.error("delete rating: {query} raised an error: \n {exception}".format(query=query, exception=exception))              raise |

Kết quả:





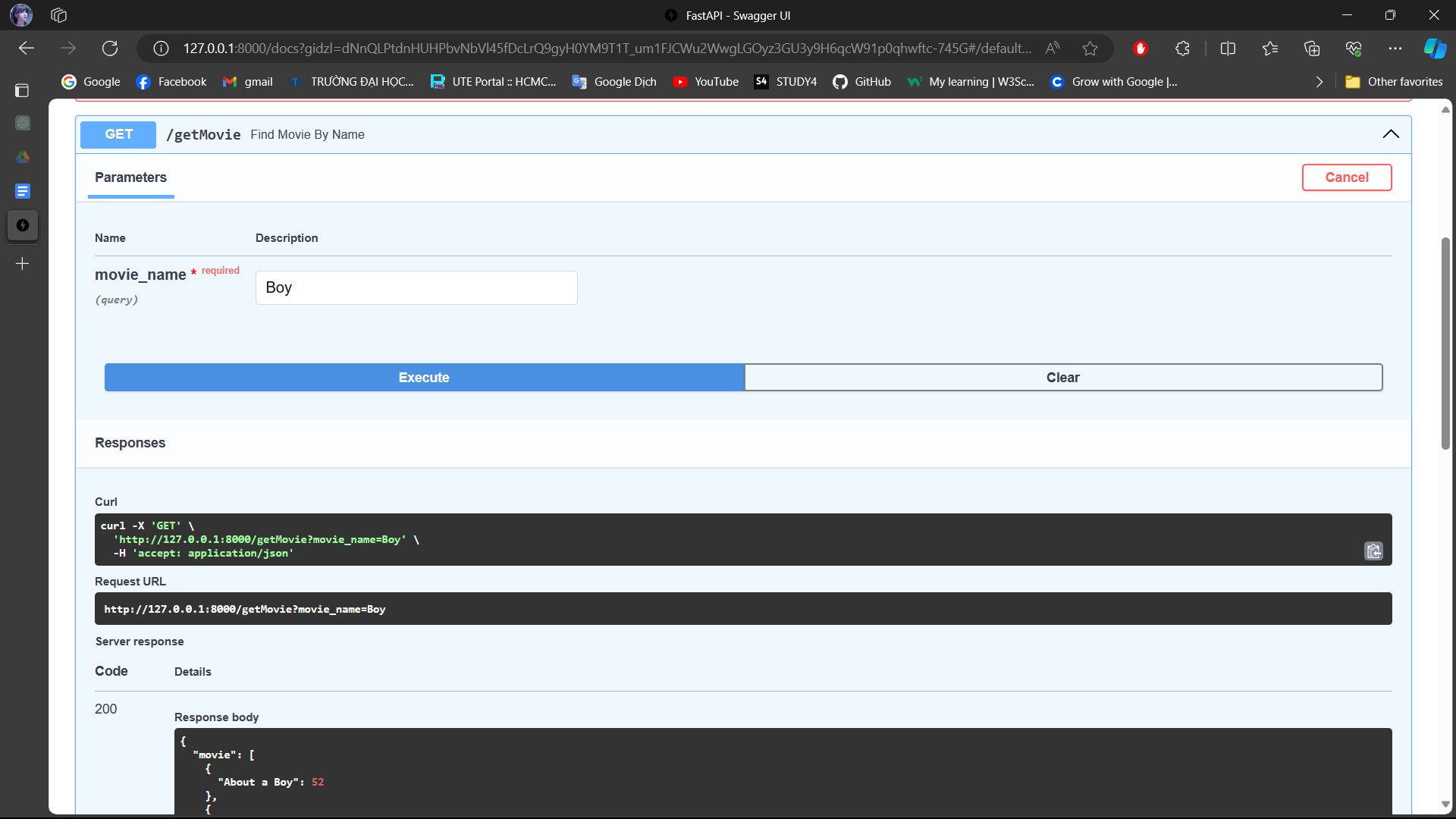
#### g. API Tìm thông tin Movies theo từ khóa và xem được số lượt đánh giá của nó

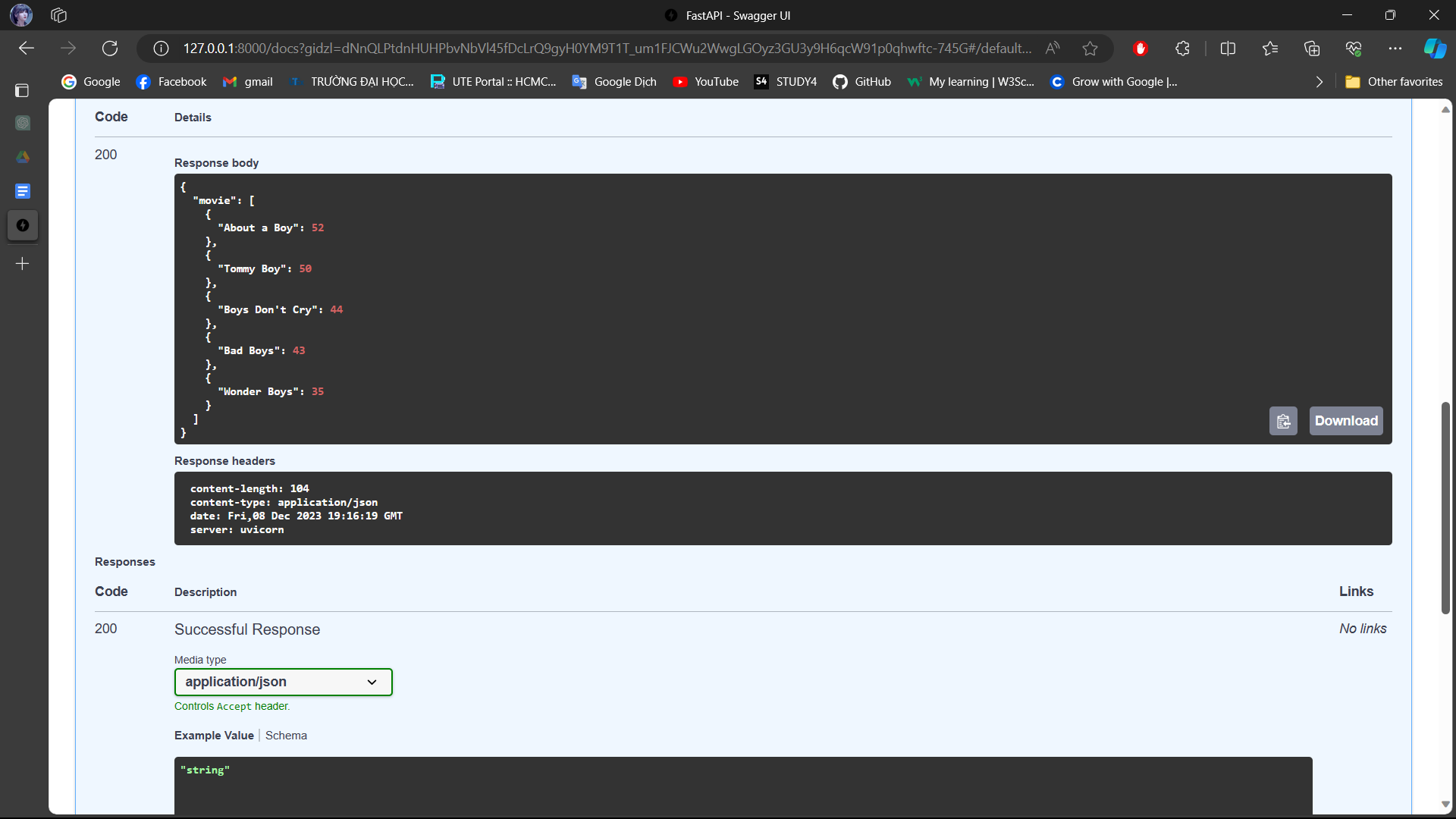
Trong phần này, câu lệnh tìm kiếm tên của các bộ phim với một từ khóa nhất định, kết quả trả về là tên hoàn chỉnh của bộ phim và số lượng review của nó.

Code:

|  |
| --- |
| # Tìm phim theo 1 chuỗi      def findMovie(self, movie\_name):          with self.driver.session() as session:              result = session.read\_transaction(self.\_\_findAndReturnMovie, movie\_name)              return {"movie": result}        @staticmethod      def \_\_findAndReturnMovie(tx, movie\_name):          query = (              """MATCH (m:Movie)<-[:RATED]-(u:User)              WHERE m.title CONTAINS $movie\_name              WITH m, COUNT(\*) AS reviews              RETURN m.title AS movie, reviews              ORDER BY reviews DESC LIMIT 5;"""          )          try:              result = tx.run(query, movie\_name=movie\_name)              return [ {record["movie"]: record["reviews"]} for record in result]          except ServiceUnavailable as exception:              logging.error("{query} raised an error: \n {exception}".format(query=query, exception=exception))              raise |

Kết quả:





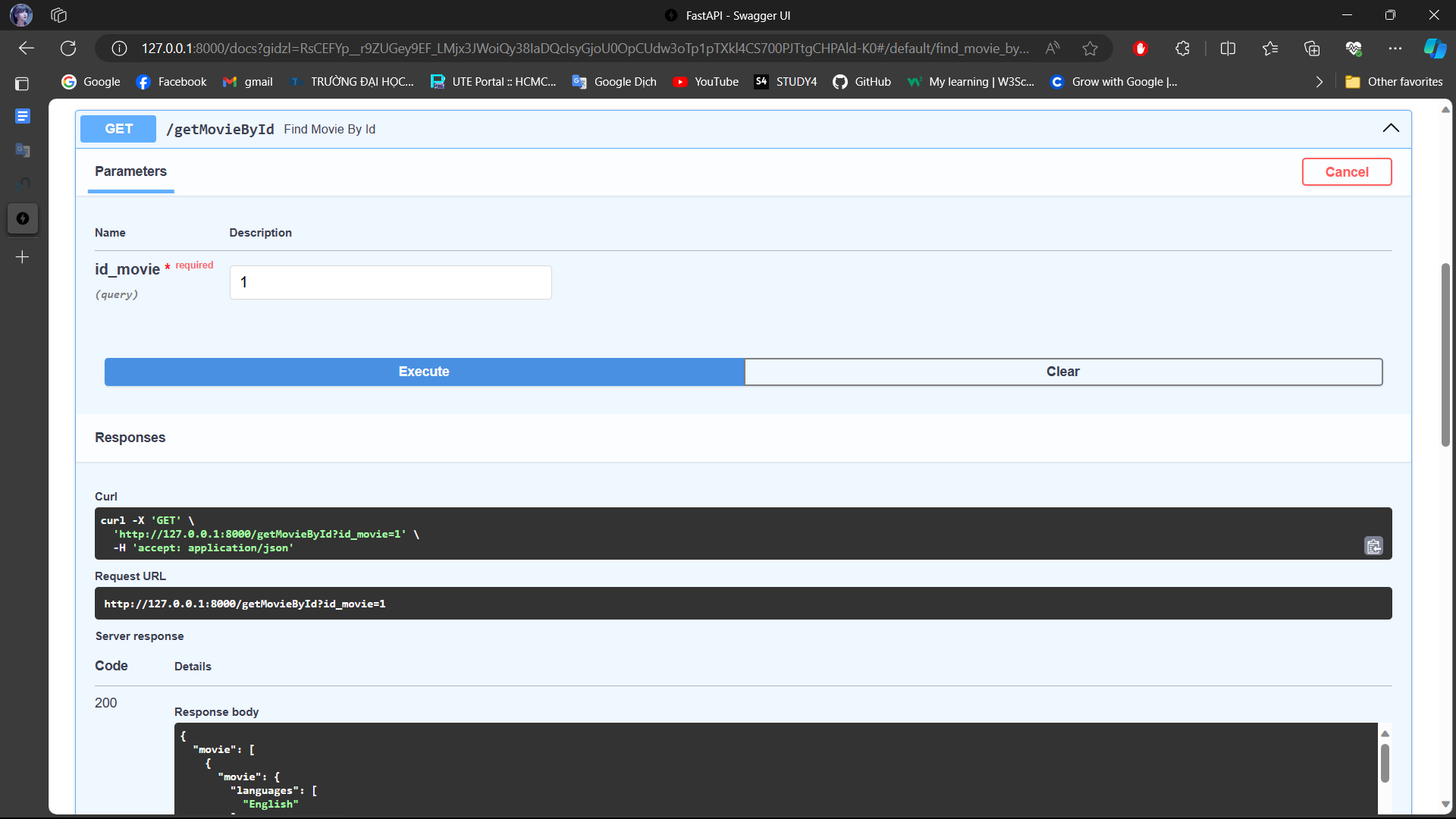
#### h. API Tìm bộ phim theo Id của phim đó

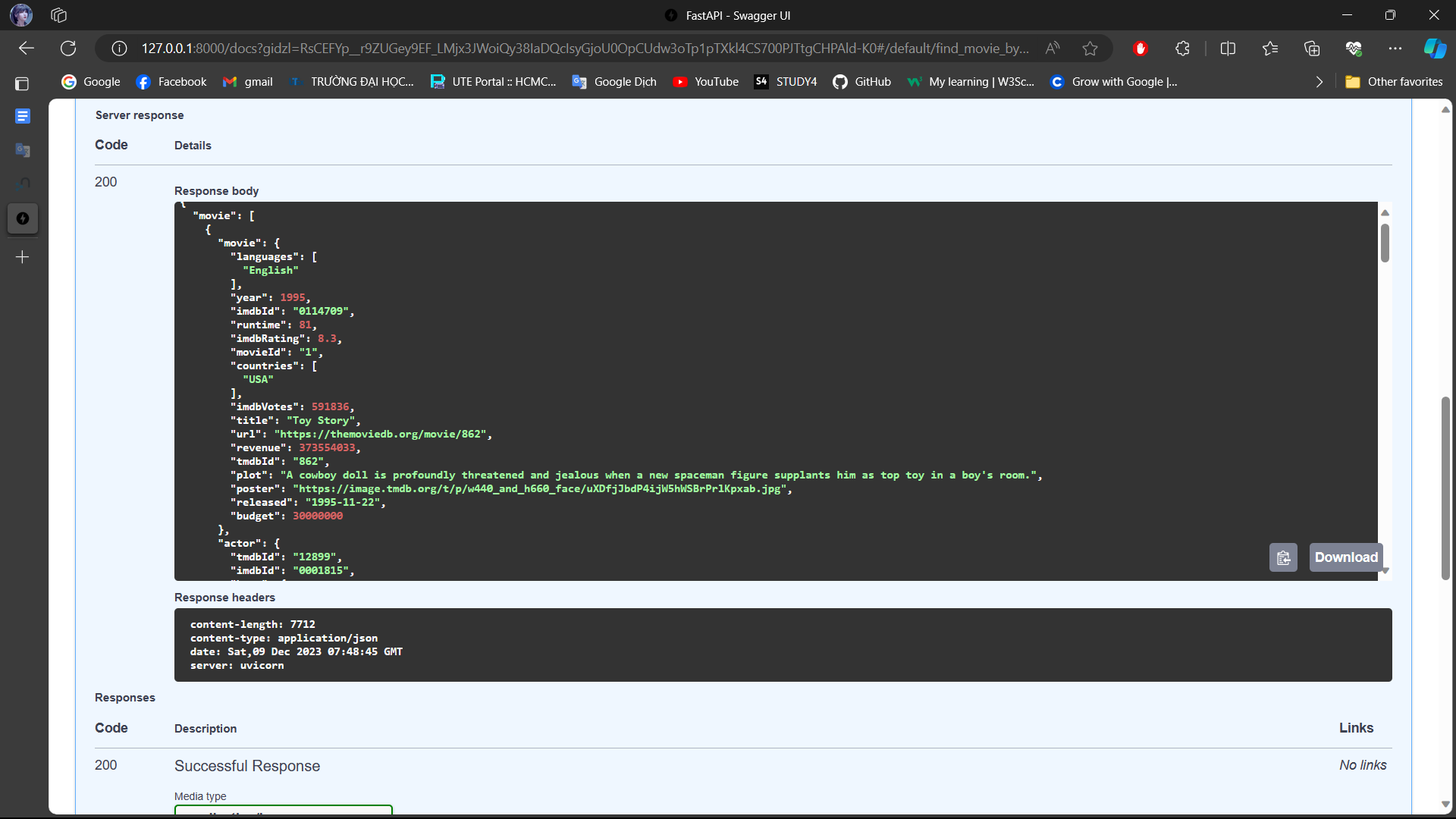
Khi người dùng nhập Id của một bộ phim nào đó, kết quả sẽ trả về đầy đủ các thuộc tính của node Movie, node Director và node Actor.

Code:

|  |
| --- |
| # Tìm phim theo Id của phim      def findMovieById(self, id\_movie):          with self.driver.session() as session:              result = session.read\_transaction(self.\_\_findIdAndReturnMovie, id\_movie)              return {"movie": result}      @staticmethod      def \_\_findIdAndReturnMovie(tx, id\_movie):          query = (              """MATCH (d:Director)-[:DIRECTED]->(n:Movie)<-[:ACTED\_IN]-(a:Actor)                  WHERE n.movieId = $id\_movie RETURN n,a,d """          )          try:              result = tx.run(query,  id\_movie=id\_movie)              return [ {"movie": record["n"], "actor": record["a"], "director": record["d"]} for record in result]          except ServiceUnavailable as exception:              logging.error("{query} raised an error: \n {exception}".format(query=query, exception=exception))              raise |

Kết quả:





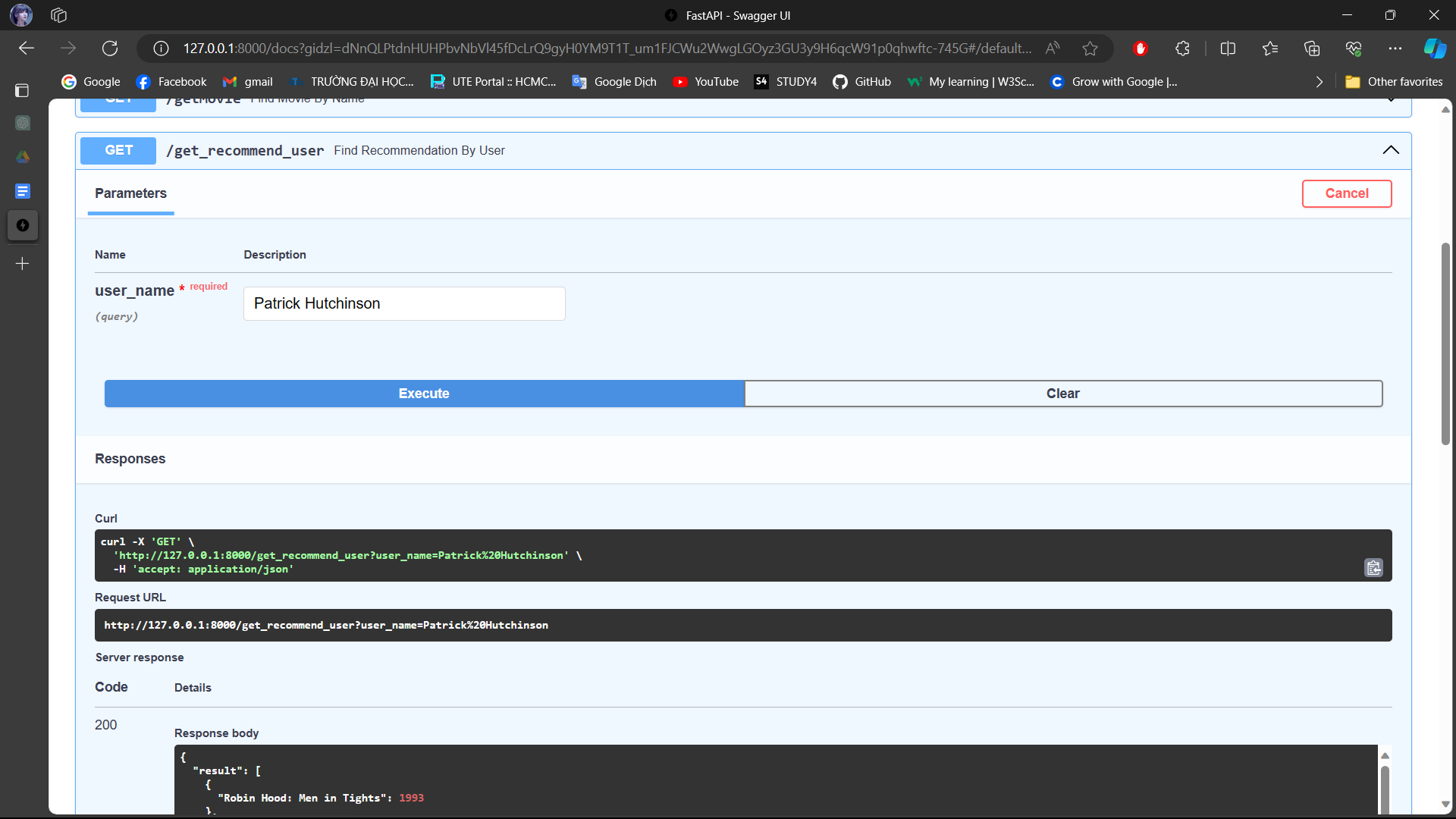
#### i. API Đưa ra đề xuất một bộ phim mà người dùng chưa xem

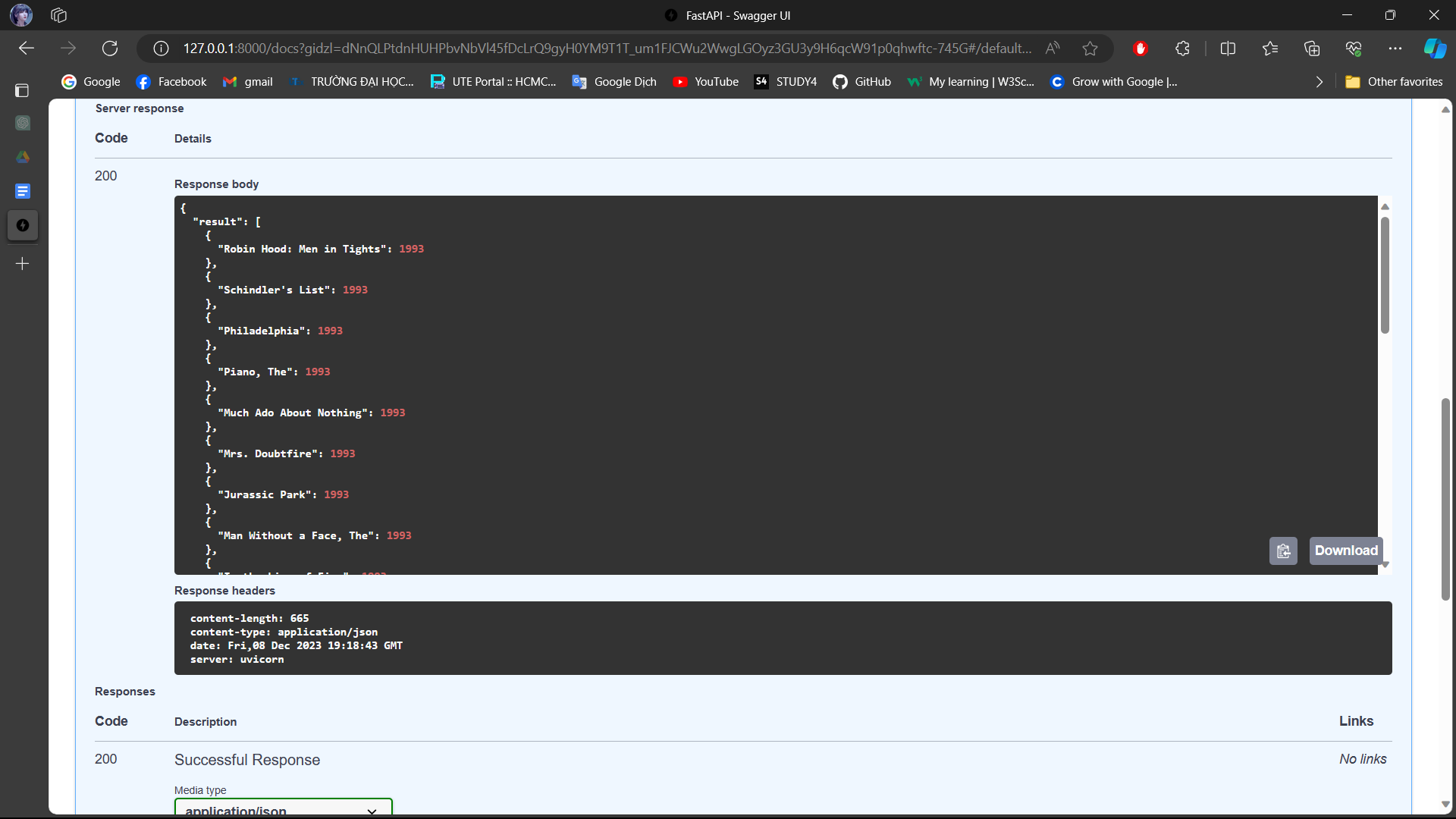
Sử dụng các đánh giá và hành động xem của những người dùng khác để tìm các mục giới thiệu. Người dùng đã xem phim này, cũng sẽ mua xem phim kia.

Code:

|  |
| --- |
| # Tìm những bộ phim mà User chưa xem      def find\_recommendation\_user(self, username):          with self.driver.session() as session:              result = session.read\_transaction(self.\_\_findByUserWatch, username)              return {"result": result}        @staticmethod      def \_\_findByUserWatch(tx, username):          query = ("""MATCH (u:User {name: $username})-[:RATED]->(:Movie)<-[:RATED]-(o:User), (o)-[:RATED]->(rec:Movie)                      WHERE NOT EXISTS( (u)-[:RATED]->(rec) )                      RETURN rec.title as title, rec.year as year                      LIMIT 25""")          try:              result = tx.run(query, username = username)              return  [ {record['title']:record['year']} for record in result]          except ServiceUnavailable as exception:              logging.error("Recommend movies by user watched: {query} raised an error: \n {exception}".format(query=query, exception=exception))              raise |

Kết quả:





# **PHẦN 3: KẾT LUẬN**

**1. KẾT QUẢ ĐẠT ĐƯỢC**

**1.1. Về kiến thức**

Nắm được các kiến thức cũng như những vấn đề liên quan về NoSQL, Neo4j. Áp dụng kiến thức để thiết kế, minh họa một tập dữ liệu cụ thể.

**1.2. Về thiết kế CSDL minh họa**

Sử dụng được các hàm, lệnh cơ bản để thiết kế CSDL minh họa về truy vấn, thêm, sửa, xóa dữ liệu trong Neo4j một cách chính xác, trực quan.

Thiết kế được giao diện để demo kết nối và các chức năng cơ bản.

**2. ƯU ĐIỂM, HẠN CHẾ CỦA ĐỀ TÀI**

**2.1. Ưu điểm**

Thiết kế CSDL minh họa trực quan, thực hiện các thao tác như thêm, sửa, xóa và truy vấn dữ liệu chính xác.

Thiết kế được giao diện để demo kết nối và các chức năng cơ bản.

**2.2. Hạn chế**

Còn tương đối hạn chế về các thao tác với CSDL trong Neo4j, chưa áp dụng được những công cụ bổ sung khác.

Giao diện chưa bắt mắt, còn khó sử dụng.

**3. HƯỚNG PHÁT TRIỂN**

Sử dụng công cụ tính toán có sẵn trong Neo4j để xây dựng hệ thống dự đoán và gợi ý phim.

Xây dựng website về mặt giao diện và tính năng.

# **TÀI LIỆU THAM KHẢO**

[1] Giới thiệu về NoSQL database. (n.d.). Viblo. Retrieved November 28, 2023, from <https://viblo.asia/p/gioi-thieu-ve-nosql-database-djeZ1a9jZWz>

[2] NoSQL là gì? | Cơ sở dữ liệu phi quan hệ, Mô hình dữ liệu sơ đồ linh hoạt. (n.d.). AWS. Retrieved November 28, 2023, from <https://aws.amazon.com/vi/nosql/>

[3] Tìm hiểu về Cơ sở dữ liệu Phi quan hệ - NoSQL. (n.d.). QuanTriMang. Retrieved November 29, 2023, from <https://quantrimang.com/cong-nghe/co-so-du-lieu-phi-quan-he-nosql-160708>

[4] Getting Started with Neo4j Desktop on Windows (pre V 1.2.3). (n.d.) Youtube. Retrieved November 30, 2023, from <https://www.youtube.com/watch?v=hIvNexwVYNw>

[5] Graph Data Modeling - Developer Guides. (n.d.) Neo4j. Retrived December 02, 2023, from <https://neo4j.com/developer/data-modeling/>

[6] Cơ sở dữ liệu đồ thị Neo4j: Khả năng và lợi nhuận cho doanh nghiệp của bạn. (n.d.). AppMaster. Retrieved December 02, 2023, from <https://appmaster.io/vi/blog/co-so-du-lieu-do-thi-neo4j>

[7] Tìm hiểu về ngôn ngữ truy vấn Cypher. (n.d.) Viblo. Retrieved November 23, 2023, from <https://viblo.asia/p/tim-hieu-ve-ngon-ngu-truy-van-cypher-gDVK2BmAKLj>

[8] Neo4j Tutorial. (n.d.) tutorialspoint. Retrieved November 23, 2023, from <https://www.tutorialspoint.com/neo4j/index.htm>

[9] Cypher Cheat Sheet - AuraDB Enterprise. (n.d.) Neo4j. Retrieved December 03, 2023, from <https://neo4j.com/docs/cypher-cheat-sheet/5/auradb-enterprise>

[10] Neo4j Python Driver 5.15. (n.d.) Neo4j. Retrieved December 03, 2023, from <https://neo4j.com/docs/api/python-driver/current/>

[11] Tạo API siêu nhanh với FastAPI. (n.d.) HIENLTH Channel. Retrieved December 03, 2023, from <https://www.youtube.com/watch?v=mSoa-oraC6w>

[12] FastAPI (Phần 1): Giới thiệu và setup môi trường. (n.d.) Viblo. Retrieved November 04, 2023, from <https://viblo.asia/p/fastapi-phan-1-gioi-thieu-va-setup-moi-truong-WR5JRxjQVGv>

[13] Hướng dẫn cơ bản framework FastAPI từ A -> Z (Phần 1). (n.d.) Viblo. Retrieved November 04, 2023, from <https://viblo.asia/p/huong-dan-co-ban-framework-fastapi-tu-a-z-phan-1-V3m5W0oyKO7>