**Tài liệu báo cáo tìm hiểu và tích hợp gRPC**

1. **gRPC là gì?**
2. **Khái niệm:**

* gRPC là viết tắt của Google Remote Procedure Call.
* gRPC là một framework RPC mã nguồn mở, hiện đại và hiệu năng cao mà có thể chạy trên bất kỳ môi trường nào.
* Framework này được Google khởi công phát triển vào năm 2015, đến 08/2016 thì được phát hành chính thức
* Đây được cho là một thế hệ tiếp theo của RPC (Remote Procedure Calls) đặc biệt là trong mô hình Microservices.
* Nó mang lợi ích về hiệu suất và các tính năng hiện đại cho các ứng dụng client-server

1. **Vì sao cần gRPC?**

* Khi chúng ta có nhiều services và đang tải rất cao. Ví dụ vài trăm service và tải đâu đó ở ngưỡng 100k CCU – Concurrent Users (số lượng user đang hoạt động cùng một thời điểm).
* Khi đó nếu một request cần phải tổng hợp dữ liệu qua nhiều services. Ở mỗi đầu service khi nhận các request trung gian này, chúng phải encode và decode liên tục (VD: JSON/XML). Việc này có thể gây quá tải cho các CPU. Lẽ ra CPU nên giành cho việc khác quan trọng hơn là chỉ vì en/de code dữ liệu trung gian.
* Ý tưởng về việc làm thế nào để các service giao tiếp với nhau với tốc độ cao nhất, giảm tải en/de code data chính là lý do thúc đẩy gRPC ra đời.

1. **Lưu ý trong gRPC**

* gRPC nên dùng để giao tiếp backend-to-backend. CPU không gánh nhiều cost cho encode/decode mỗi đầu ra. Tuy nhiên đặc tính mỗi đầu cần import file model chung (gen từ protobuf file) nên nếu update thì phải update đủ. Việc này vô tình tạo dependency cho các bên sử dụng, có thể bạn sẽ không thích điều này.
* gRPC thường được đấu nối vào service mesh (hoặc sidecar trong Microservices), để có thể xử lý connection HTTP/2 cũng như monitoring nó tốt hơn.
* gRPC support streaming 2 đầu nên rất được lòng các fan streaming system, envent sourcing (stream event).
  + VD: gRPC được sử dụng trong: vitess, neo4J vì lý do trên
* gRPC nếu dùng cho frontend-backend thì thật sự rất cân nhắc. Connection statefull tạo nhiều sự khó chịu trong scale tải hoặc bạn có thể bị Head of line blocking
* gRPC vẫn có thư viện gRPC Gateway chính chủ của Google. Tức là các bạn vẫn có thể chạy 1 port http/1 cho REST và port HTTP/2 của gRPC đồng thời. Như vậy không phải là không có cách để quay về REST thân quen, nhưng đương nhiên là đi qua proxy service nên phức tạp hơn.

1. **Tóm lại:**

* Tóm lại gRPC là một kỹ thuật rất ưu việt để scale tải hệ thống, đặc biệt trong hệ thống phân tán, nhiều services hoặc Microservices. Việc sử dụng tốt gRPC vẫn phụ thuộc phần lớn vào kỹ thuật xây dựng service và khả năng deploy và vận hành (DevOps).

1. **REST là gì?**
2. **Khái niệm?**

* REST API là một giao diện lập trình ứng dụng (API) tuân thủ các ràng buộc và quy ước kiến trúc REST được sử dụng trong việc giao tiếp giữa client và server.
* REST là viết tắt của Representational State Transfer, nó được tạo ra bởi nhà khoa học máy tính Roy Fielding.
* REST API thường vẫn sử dụng giao thức HTTP/1 kèm theo các định nghĩa trước đó mà cả client và server cần tuân thủ.
* Hiện tại, các REST API dùng JSON rất phổ biến. Một ít có thể vẫn dùng XML

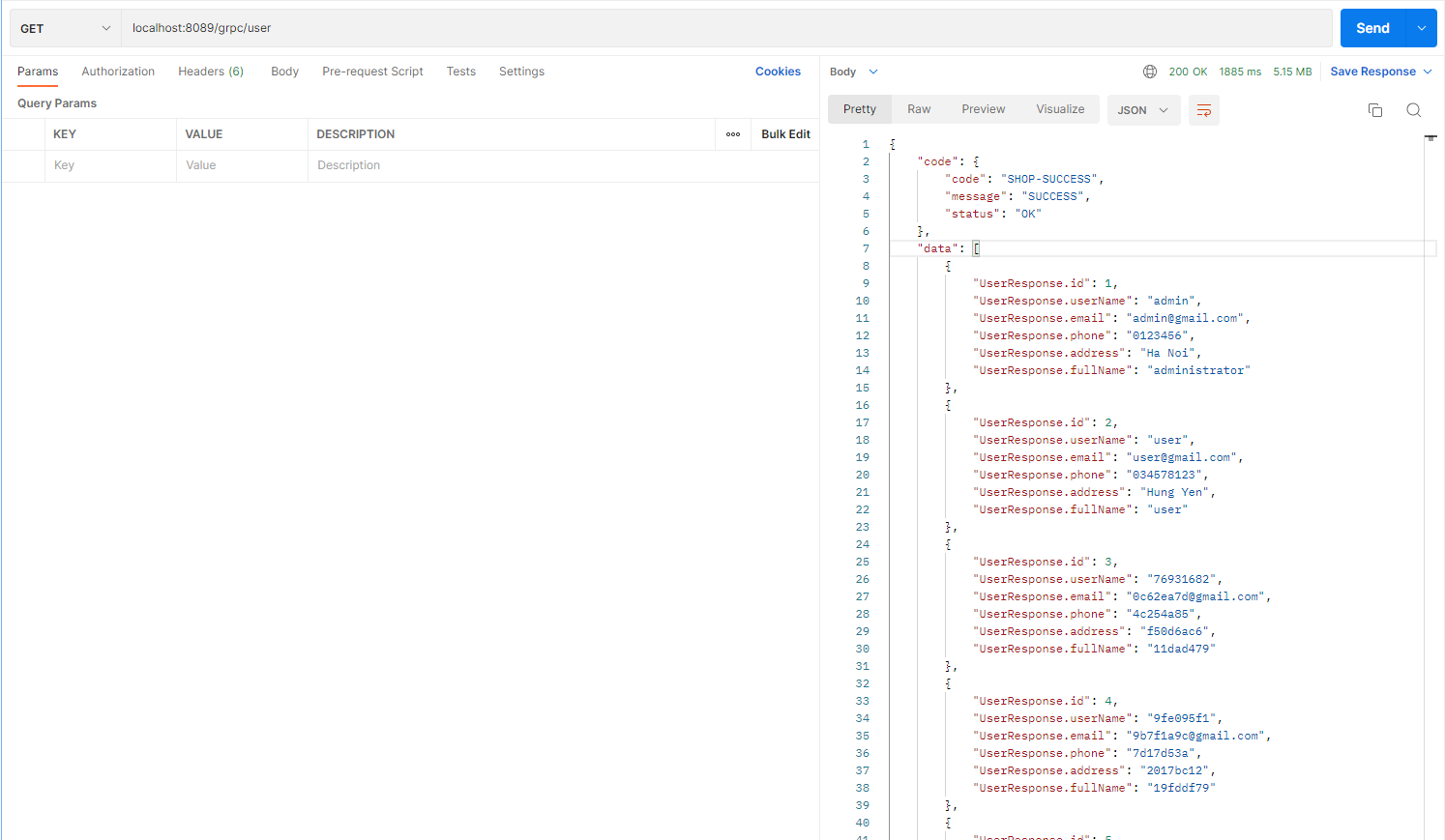
1. **Hai thành phần trong REST API**

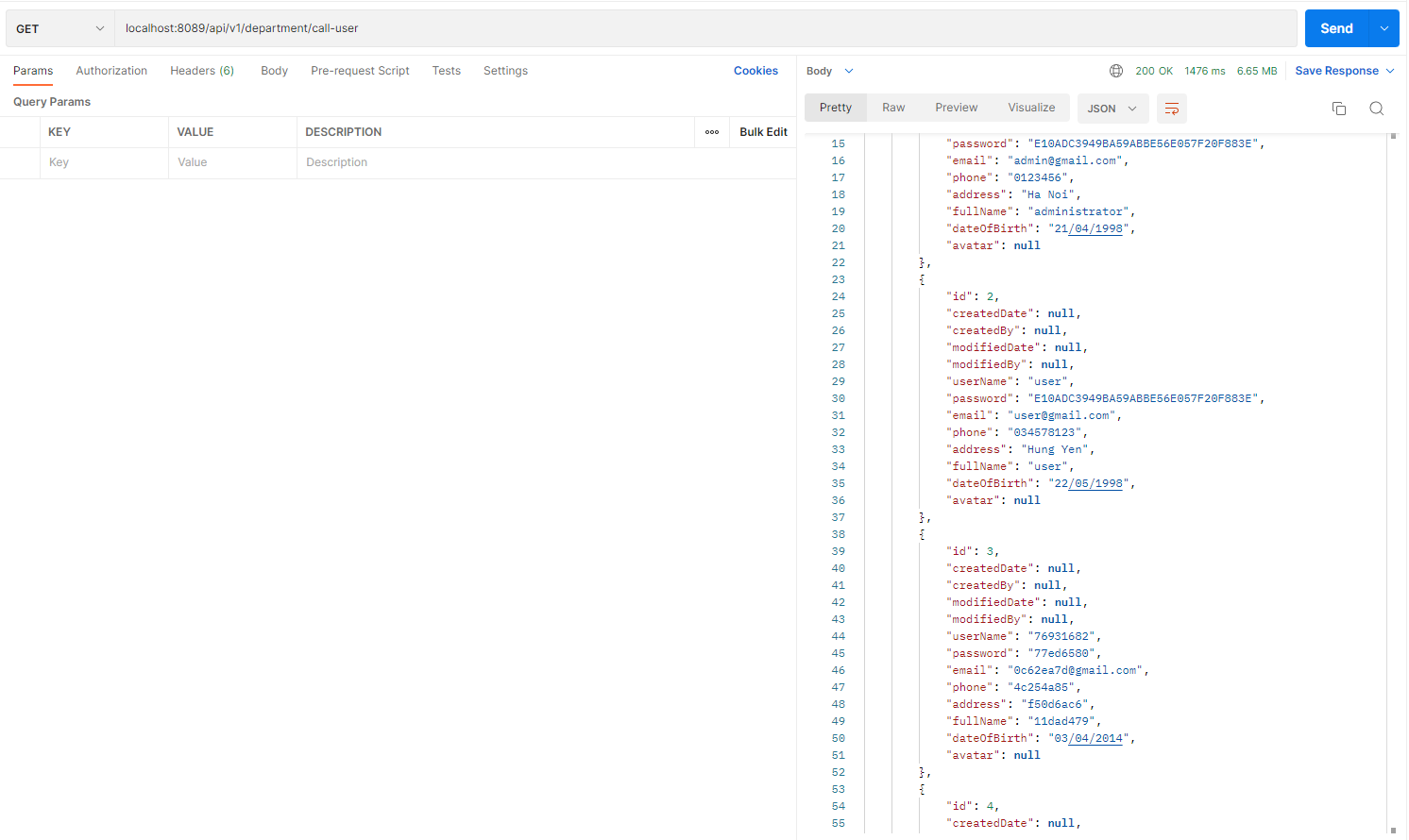
* Rest là một đại diện cho sự chuyển đổi dữ liệu. Trong kiến trúc này client và server hoàn toàn độc lập, chúng không biết gì về nhau. Mỗi một request REST API đều không mang theo trạng thái trước đó. Như vậy để đôi bên trao đổi state, chúng sẽ phải thông qua các resources. Các resource này chính là phần đại diện cho sự thay đổi dữ liệu.
* API là giao diện lập trình ứng dụng. Giao diện này không giành cho người dùng cuối mà dành cho các nhà phát triển. Nó là cái bề mặt, chỉ thấy được phần khai báo (tên, tham số, kiểu trả về), body thì không biết.

1. **So sánh gRPC và REST trong dự án mock**
2. **Mô tả dự án:**

* Project microservice có các service:
  + Common
  + Department-service (grpc client)
  + Proto
  + User-service (grpc server)
* Trong dự án department-service đóng vai trò là grpc client sẽ gọi tới user-service là grpc server để lấy dữ liệu về.
* Trong database user-service tạo 20k dữ liệu để test
* Department-service sẽ lấy tất cả dữ liệu từ user-service về và trả về cho client, sử dụng:
  + gRPC
  + REST API
* Dưới đây là kết quả so sánh trên dự án mock:

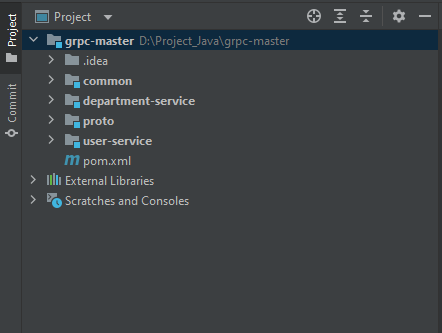
|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **gRPC** | **REST API** |
| Thời gian trả về | 2.56s | 2.08s |
| Kích cỡ data trả về | 5.15MB | 6.65 MB |
|  |  |  |
|  |  |  |





1. **Tích hợp gRPC trong dự án Microservice**

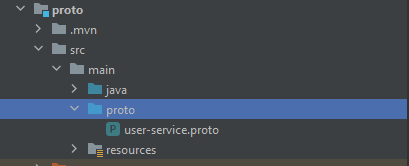
* Tại dự án tạo các service sau:



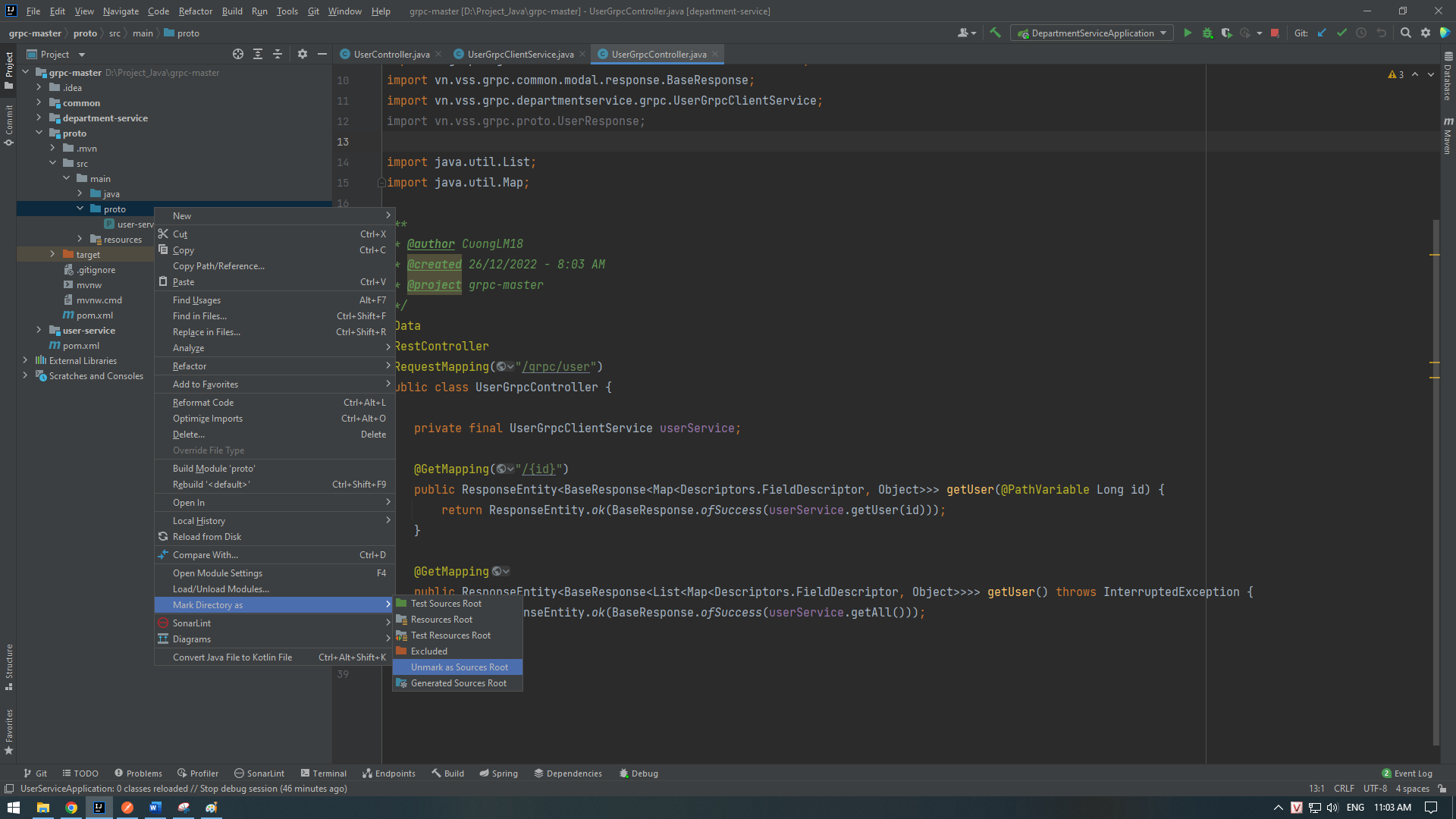
* Mô tả dự án:
  + Common: là service dùng chung
  + Proto: là service chịu trách nhiệm lưu trữ file .proto và generate class sinh ra từ file .proto
  + Department-service: đóng vai trò là gRPC-client
  + User-service: đóng vai trò là gRPC-server

1. **Khởi tạo service proto**

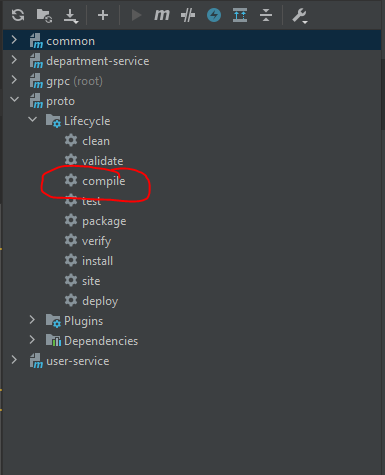
* Tại dự án proto ta tạo folder proto và thêm file user-service.proto



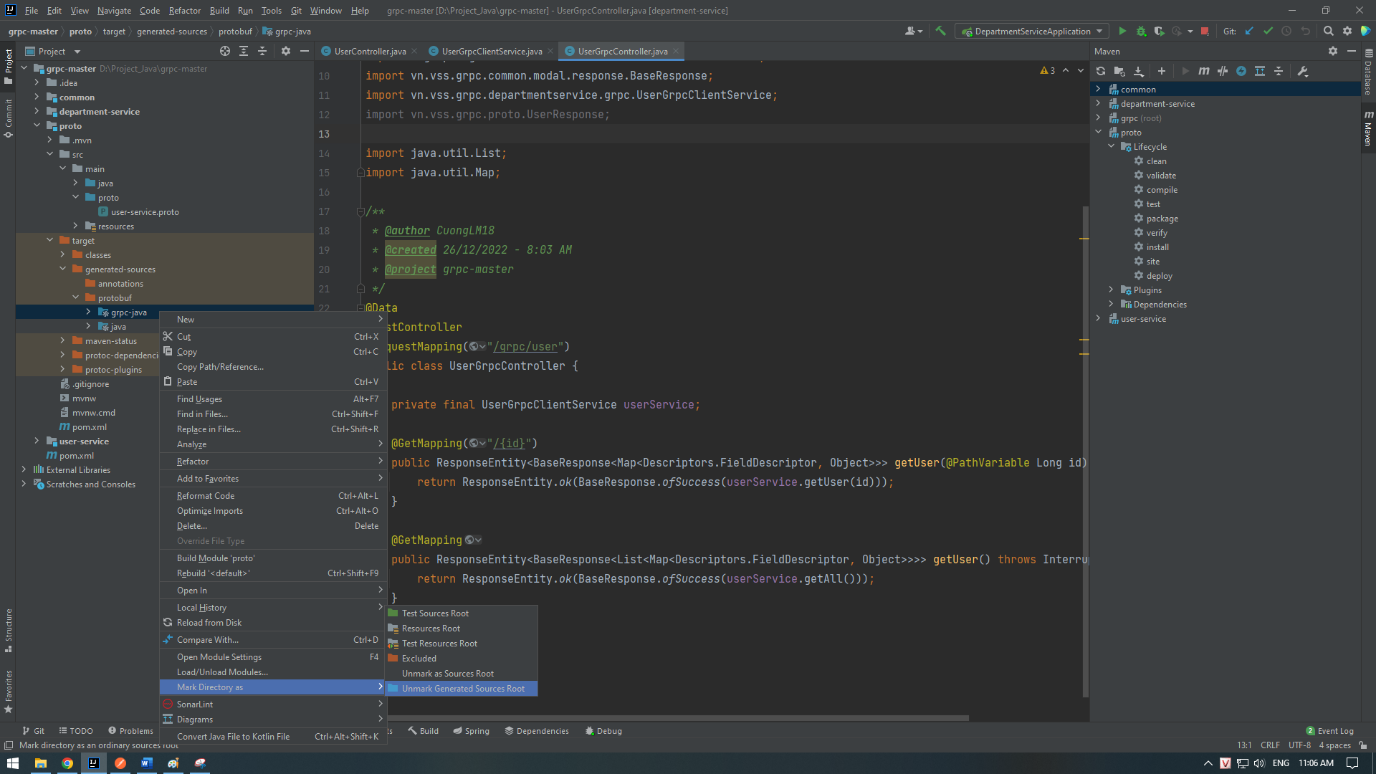
* Sau đó chọn như sau



* Chọn maven và chọn compile service proto để generate ra class được sinh ra từ file proto



* Trong file target -> generated-sourced của dự án proto ta sét như sau:

****

* Ta thêm đoạn config vào trong file pom.xml

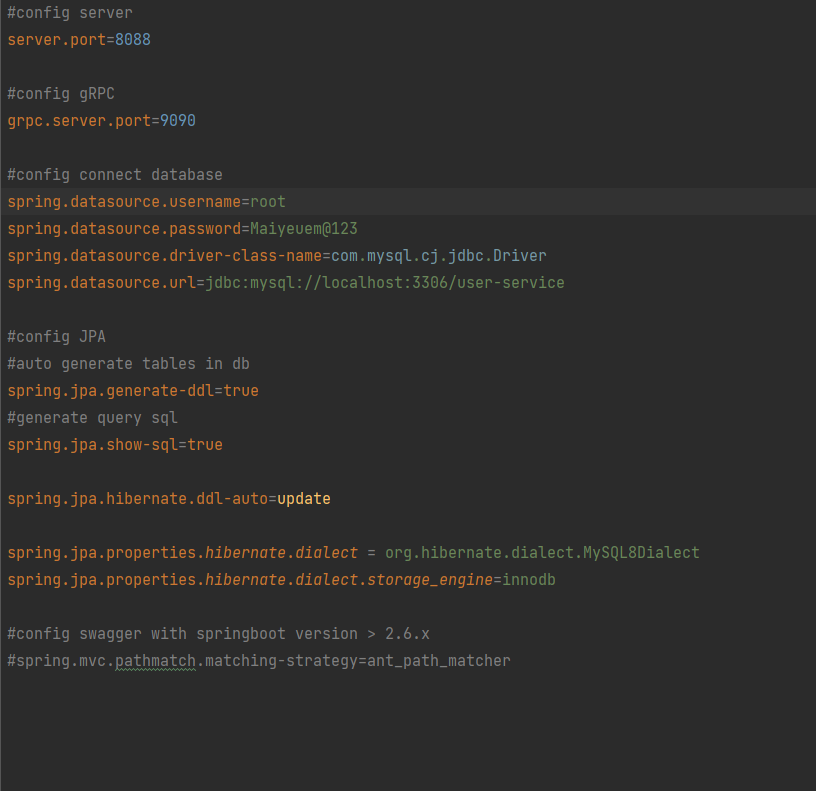


1. **Khởi tạo project user-service (gRPC – server)**

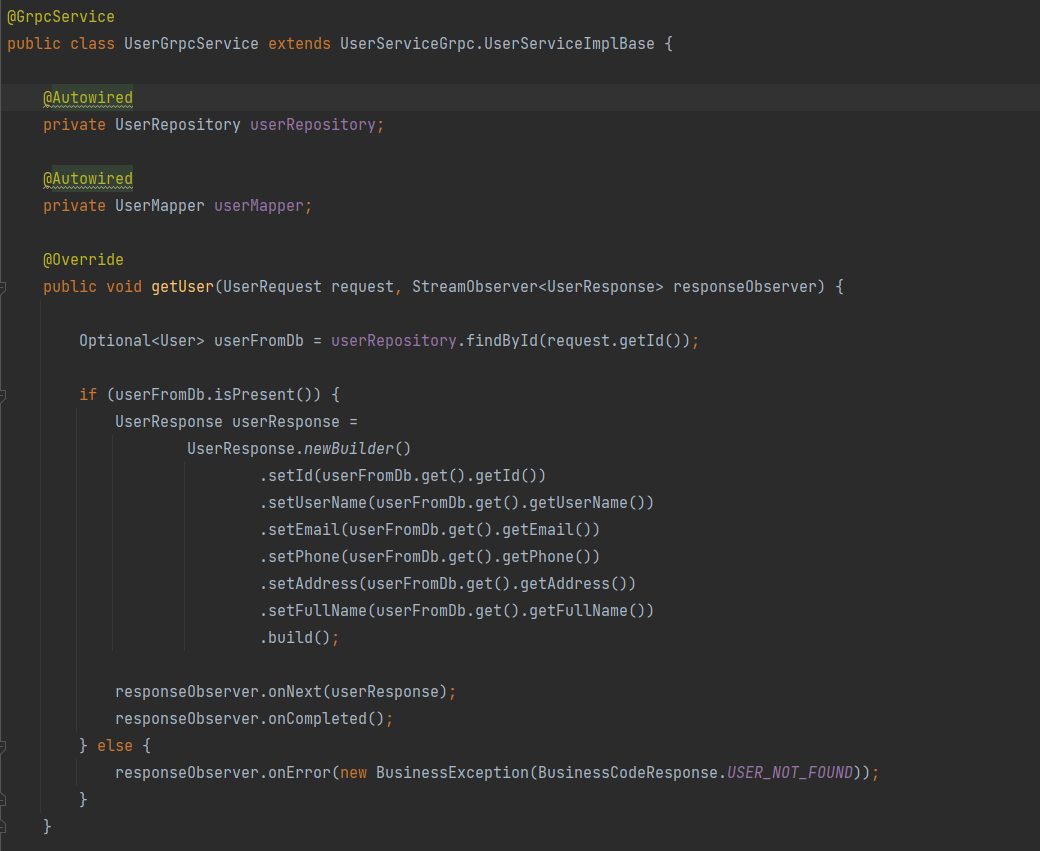
* Ta thêm các thư viện sau vào file pom.xml



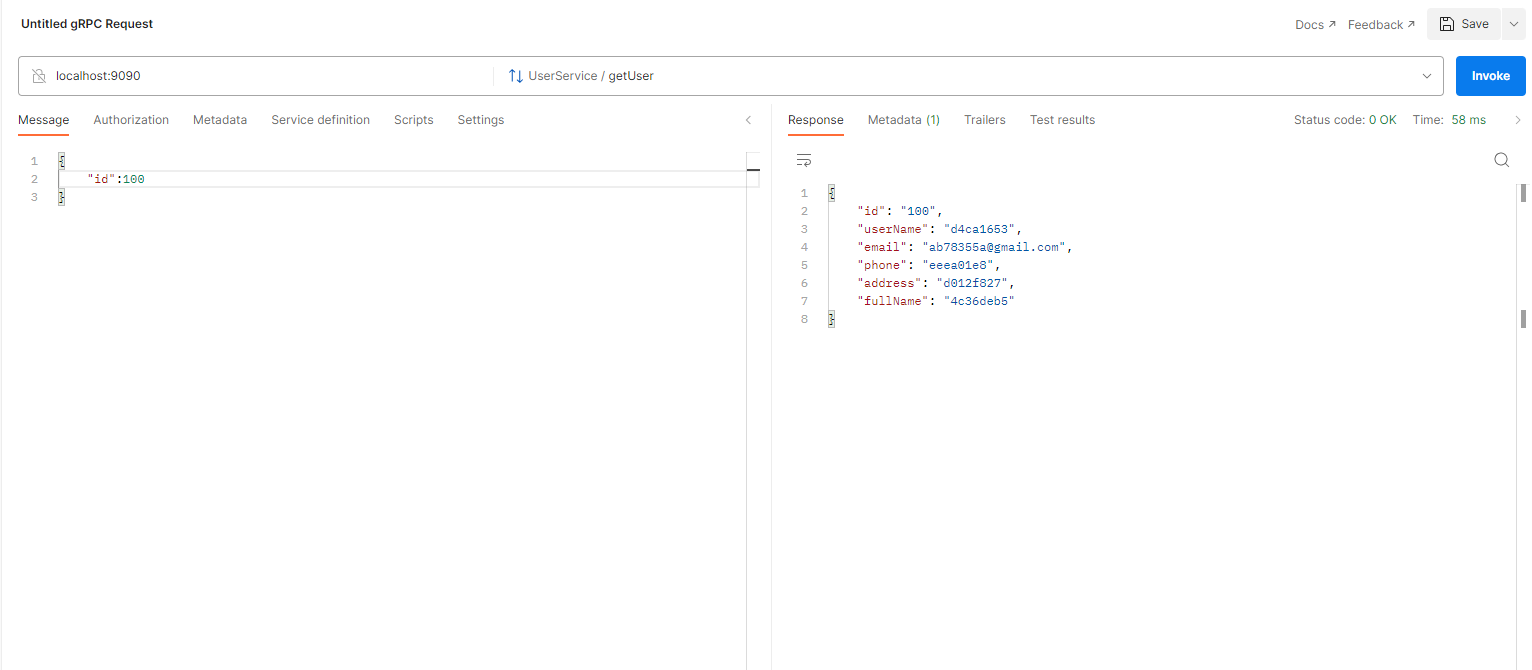
* Trong file config ta config như sau:



* Trong package grpc ta tạo class UserGrpcService extends class UserServiceImplBase được sinh ra từ file user-service.proto
* Ta override lại các method và triển khai

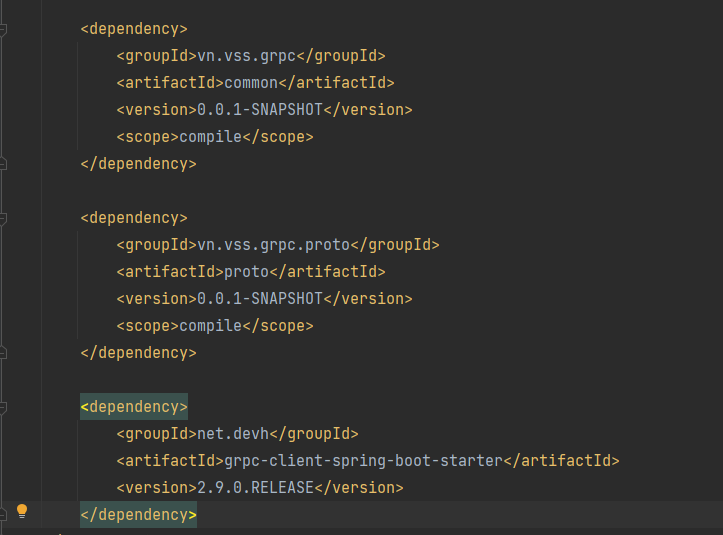


* Sau khi config ta run project và dùng postman test ta nhận được kết quả như sau:

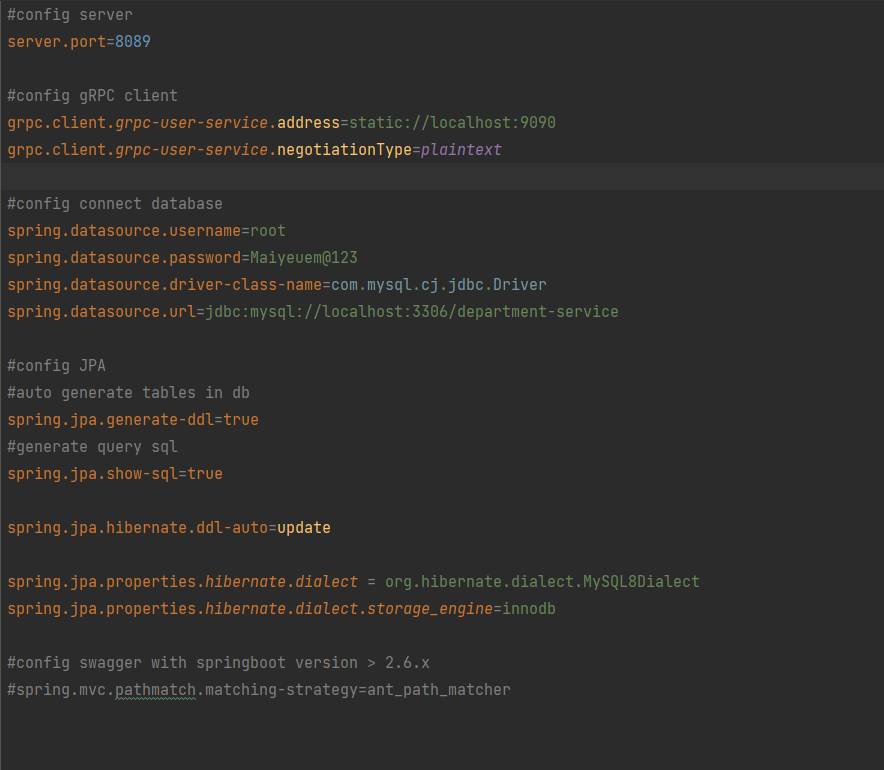


1. **Khởi tạo project department-service (gRPC-client)**

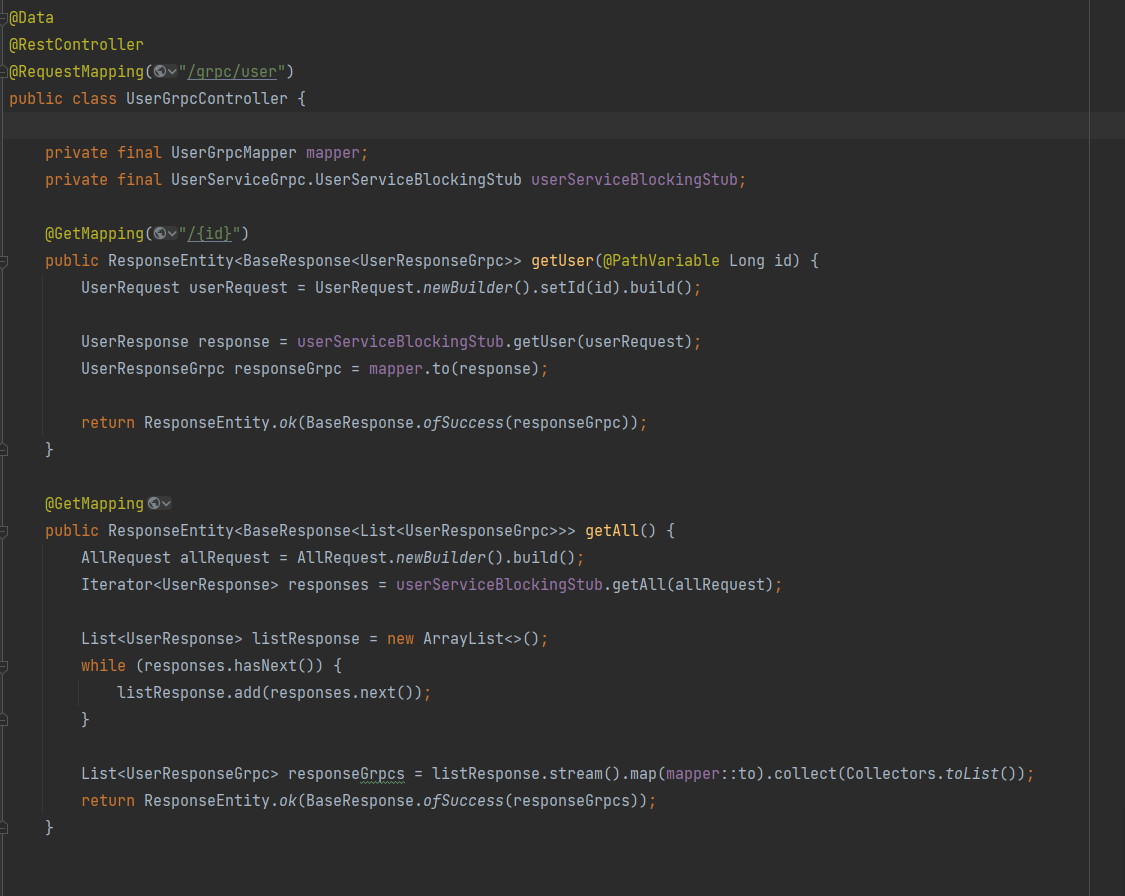
* Ta thêm các dependencies sau vào file pom.xml



* Trong file application.properties ta config như sau:



Tạo class UserGrpcController:



* Tạo class GrpcClientConfig:

