There're 3 microservices(MS) stubs implemented for you - books, authors and frontend.

Authors and books microservices contain few pre-created records.

Frontend MS should make a calls for both books and authors MS and aggregate their responses into 1 JSON.

**Request/Response Communication**

- Authors MS should have GET endpoint api/v1/authors to return all pre-created records in JSON like:

1. [
2. {
3. "id": 1,
4. "firstName": "Loreth Anne",
5. "lastName": "White - v2"
6. },
7. ...
8. ]

- Books MS should have GET endpoint api/v1/books to return all pre-created records in JSON like:

1. [
2. {
3. "id": 1,
4. "title": "Semiosis: A Novel - v2",
5. "pages": 326,
6. "authorId": 1
7. },
8. ...
9. ]

- Frontend MS should make a call to both authors and books MS using GET endpoints created from above.

Frontend MS have GET endpoint api/v1/dashboard and return aggregated results for authors and books MS in JSON like:

1. {
2. "authors": [
3. {
4. "id": 1,
5. "firstName": "Loreth Anne",
6. "lastName": "White - v2"
7. },
8. ...
9. ],
10. "books": [
11. {
12. "id": 1,
13. "title": "Semiosis: A Novel - v2",
14. "pages": 326,
15. "authorId": 1
16. },
17. ...
18. ]
19. }

**gRPC communication:**

- Implement gRPC version of authors and books MS.

- Frontend MS should be able to populate aggregated data from authors and books MS

**Message-passing communication:**

- Implement new REST PUT endpoint for **Books** MS to add a new book to the local storage (check **BookService** class). To implement this task you may need to implement new class like

**BookAndAuthor** with followingfields

private int id;  
private String title;  
private int pages;  
private int authorId;  
private String firstName;  
private String lastName;

- Once a new book is added new event with **BookAndAuthor** info should be posted to the RabbitMQ queue.

- **Authors** MS should consume event, that contains **BookAndAuthor** object, from the RabbitMQ and check if the author exists in the local storage (check **AuthorService** class). If the author is absent – it should be added to the local storage.

Useful information to start with:

* All microservices are ready to be run as Docker images, you can find **Dockerfile** for each microservice and **docker-compose.yml** to start all microservices with ports binding.
* You need to build and **package** microservice before running it in docker, check **pom.xml** file
* After microservices **.jar** files are built you can run microservices with docker-compose command.
* Useful docker-compose commands:
  1. **docker-compose up --build -** builds, (re)creates, starts, and attaches to containers for a service.
  2. **docker-compose down** -Stops containers and removes containers, networks, volumes, and images created by **up**.
* Existing **docker-compose.yml** file contains required microservices port bindings and dependencies:
  1. **Authors** MS is available by port **8091**.
  2. **Books** MS is available by port **8092**.
  3. **Frontend** MS is available by port **8093** and depends on Authors and Books
  4. **RabbitMQ** is available by ports **5672** and management port **15672.** You can check management console with **guest/guest** as user/password.
* Protobuf files (**.proto**) should be placed in **/src/main/proto** folder – this is default maven plugin behaviour.
* Default gRPC server port is **9090** anditshould beset in the **Dockerfile**’s and **docker-compose.yml**

For further reference, please consider the following sections:

[Official Apache Maven documentation](https://maven.apache.org/guides/index.html)

[Spring Boot Maven Plugin Reference Guide](https://docs.spring.io/spring-boot/docs/2.1.8.RELEASE/maven-plugin/)

**Guides**

The following guides illustrate how to use some features concretely:

[Building a RESTful Web Service](https://spring.io/guides/gs/rest-service/)

[Serving Web Content with Spring MVC](https://spring.io/guides/gs/serving-web-content/)

[Building REST services with Spring](https://spring.io/guides/tutorials/bookmarks/)

**GRPC**

[Introduction to gRPC](https://www.baeldung.com/grpc-introduction)

[gRPC for Spring Boot Microservices](https://medium.com/@sajeerzeji44/grpc-for-spring-boot-microservices-bd9b79569772)

[Language Guide (proto3)](https://developers.google.com/protocol-buffers/docs/proto3#simple)

**RabbitMQ**

[Messaging with Spring AMQP](https://www.baeldung.com/spring-amqp)

[RabbitMQ with Spring Boot](https://medium.com/@marcosstefani/rabbitmq-with-spring-boot-d05197fce05e)