Practicum 13

CSCI-P 465/565 Software Engineering I

Indiana University Bloomington - Spring 2024

- Today:
  - Communication between microservices
    - Direct messaging with HTTP
    - Indirect messaging with RabbitMQ

RabbitMQ modes

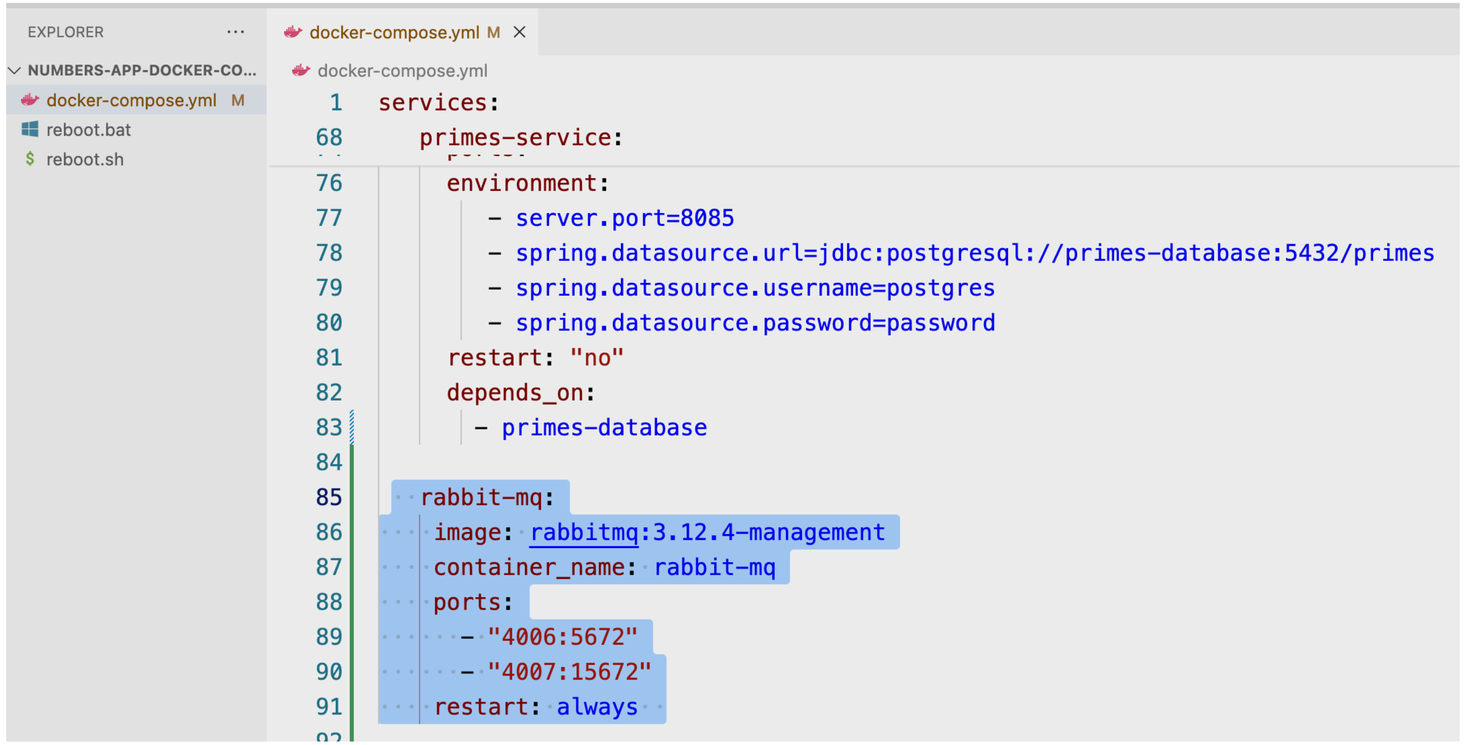
- Direct messaging
- Work queue: distribute tasks among worker nodes.
- Publish/subscribe
- Routing: receive messages selectively
- Topics: receive messages based on a pattern
- Publisher confirm: get a confirmation that the message was processed.

Communication between microservices - RabbitMQ

Add a RabbitMQ container

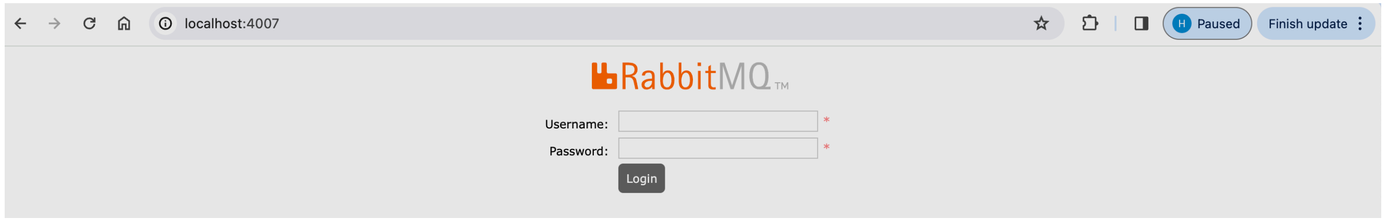
Steps

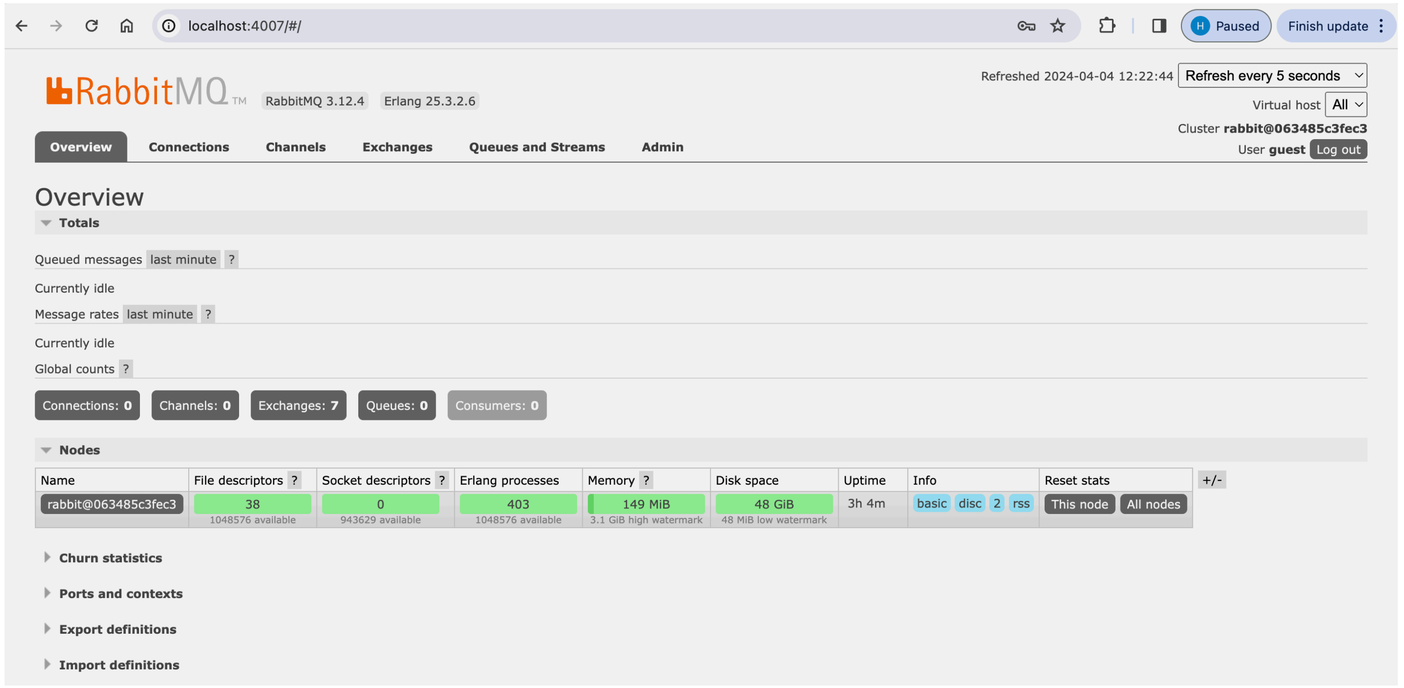
- Add a rabbitmq container:



Steps

- Reboot the docker compose and verify that you can connect to RabbitMQ management tool at http://localhost:4007
- The default username is: guest
- The default password is: guest

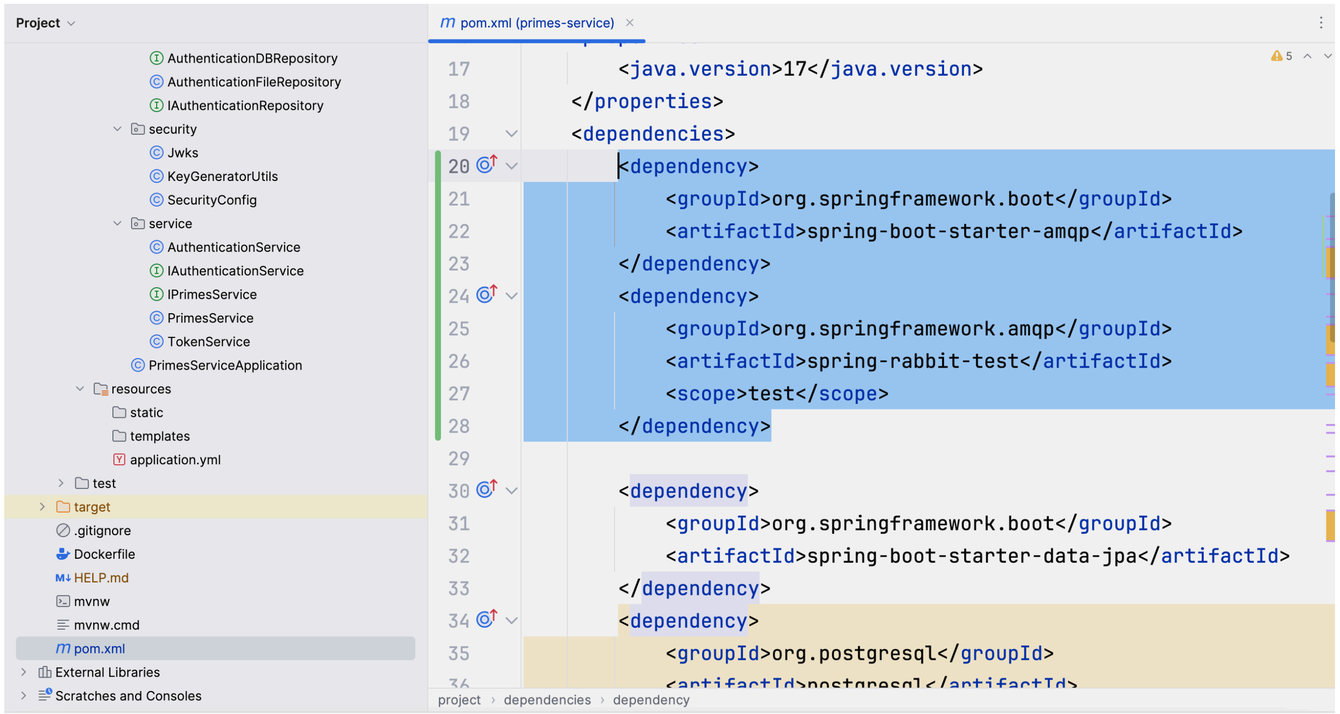




Modify the primes-service to send messages to RabbitMQ

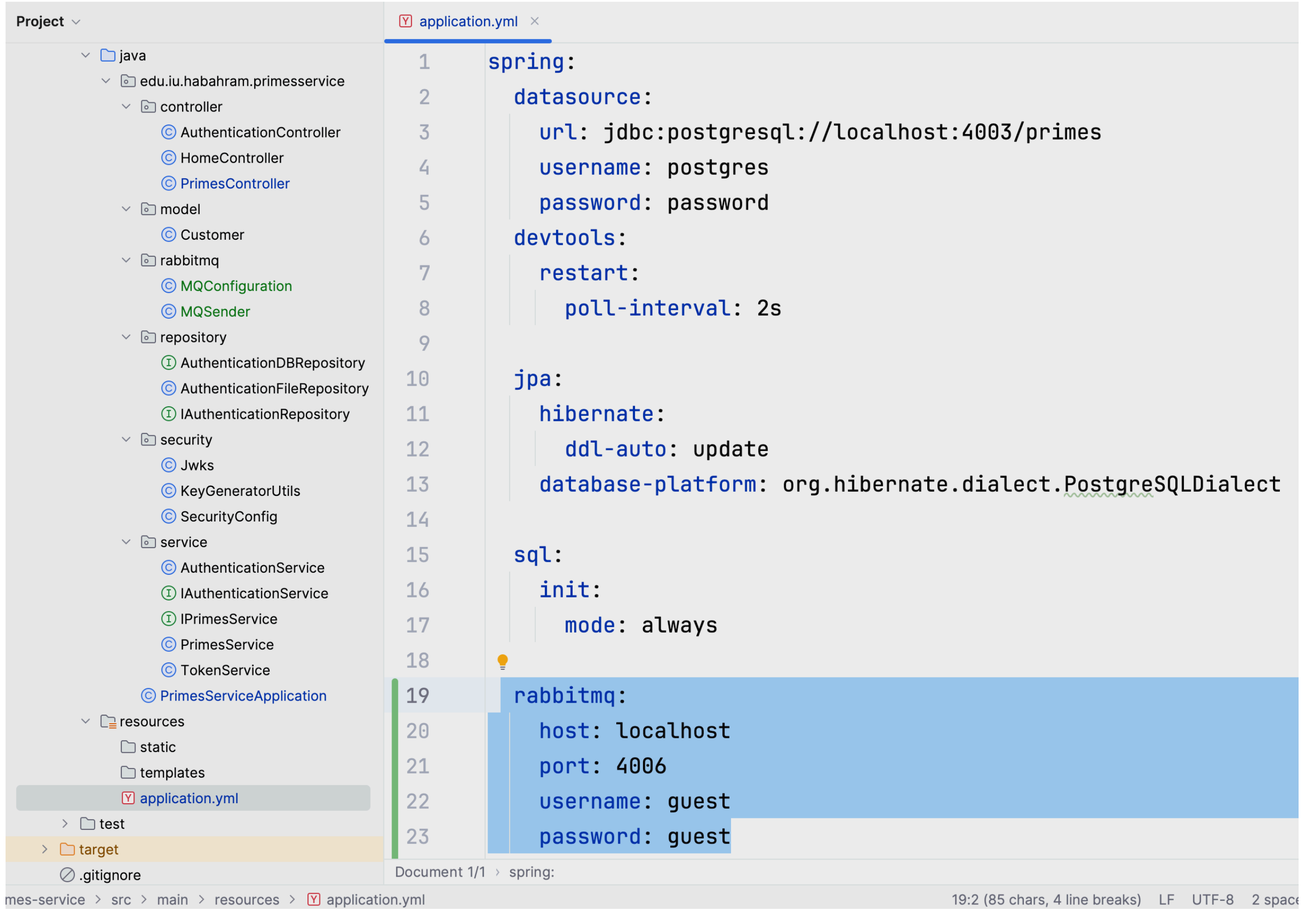
Steps

- In primes-service backend add the dependencies:



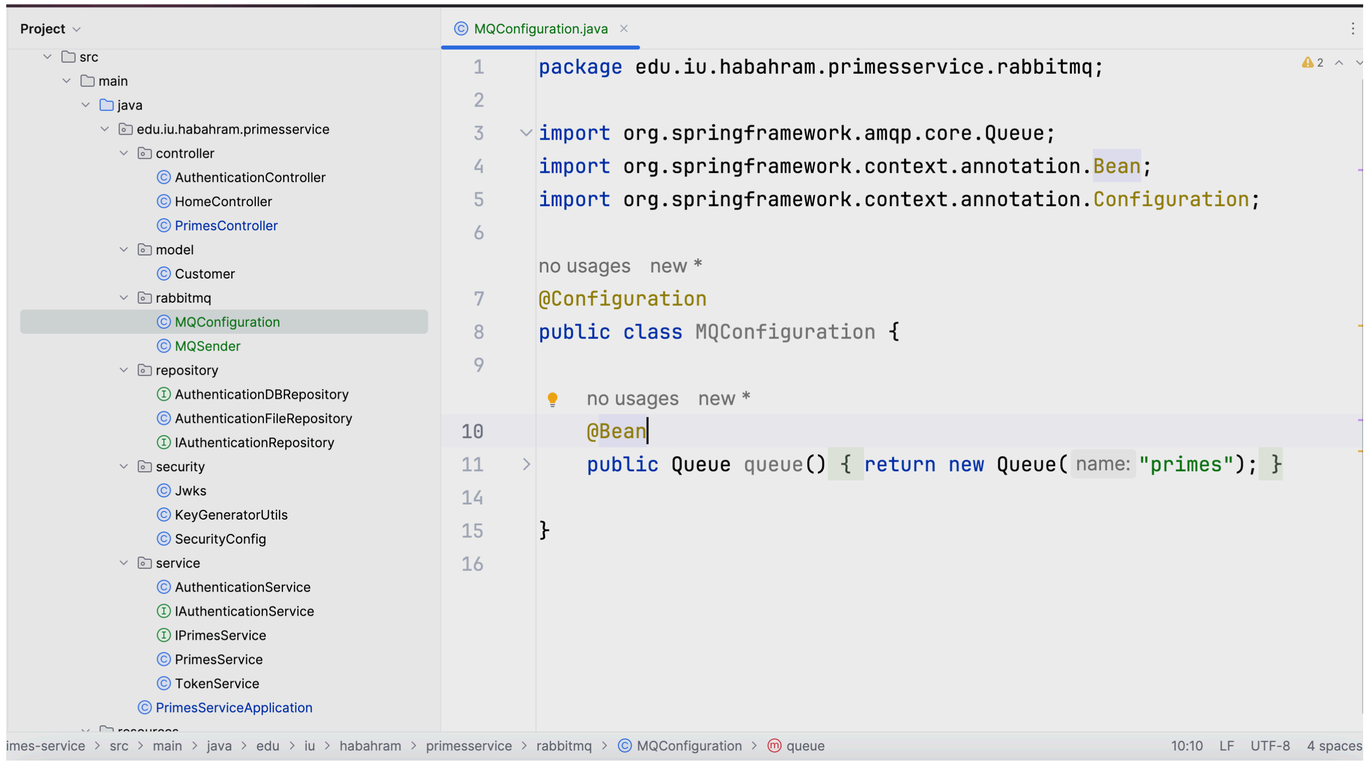
Steps

- In primes-service backend add the rabbitmq configuration settings to your application.yml file:



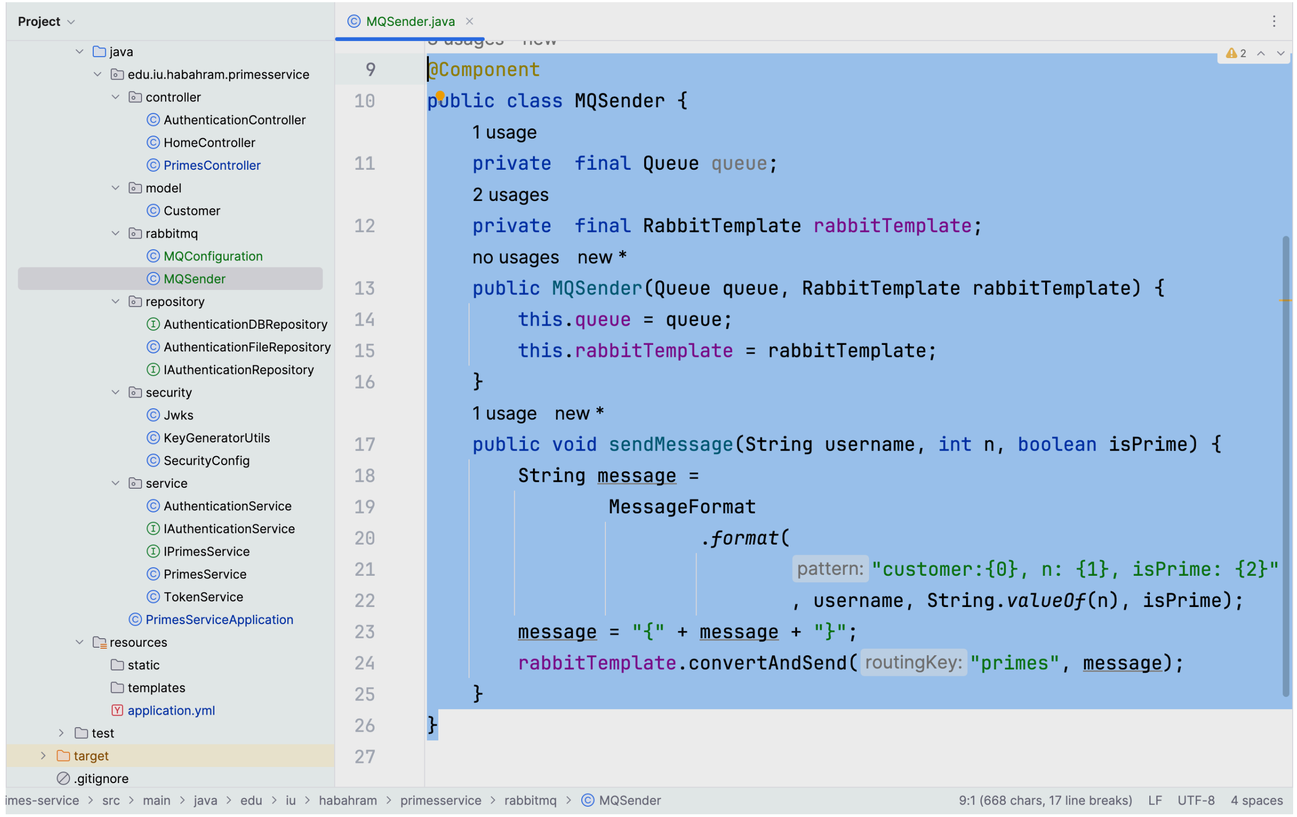
Steps

- In primes-service backend add a package, rabbitmq, and add two classes to it: MQConfiguration and MQSender.
- In the MQConfiguration add a bean that returns a reference to the primes queue:



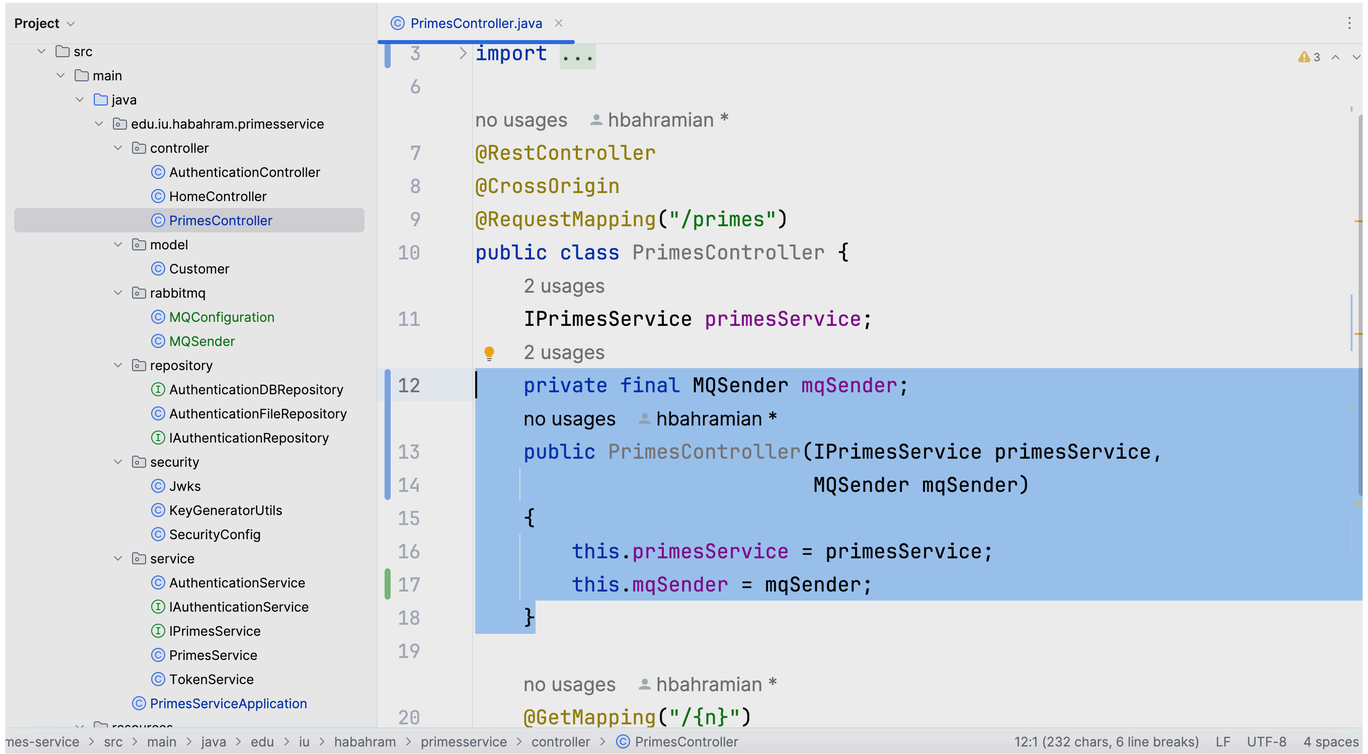
Steps

- In the MQSender add a function that sends a message (the number and a boolean indicating wether it is prime or not) to the primes queue:



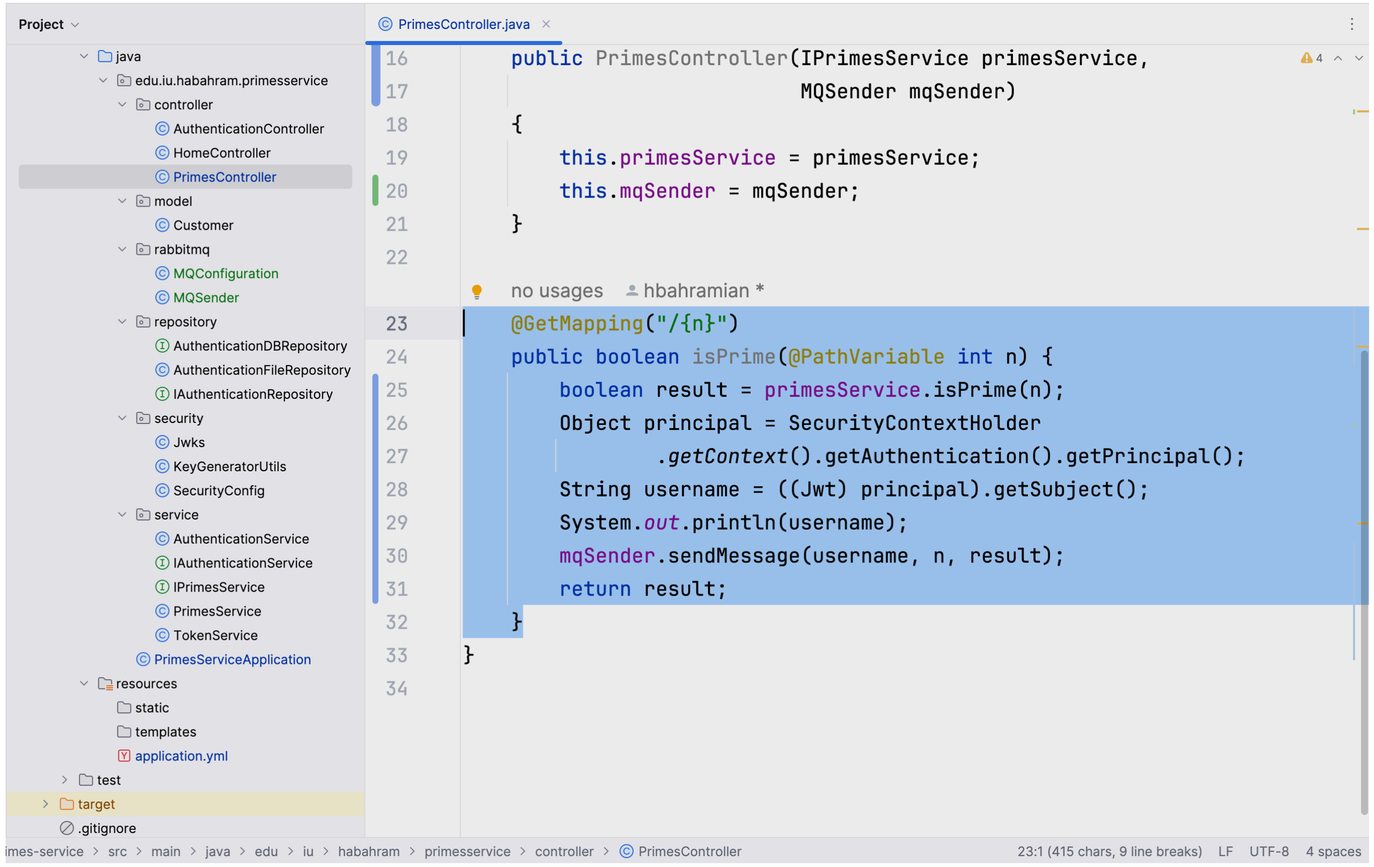
Steps

- In the PrimesController add a variable of type MQSender:



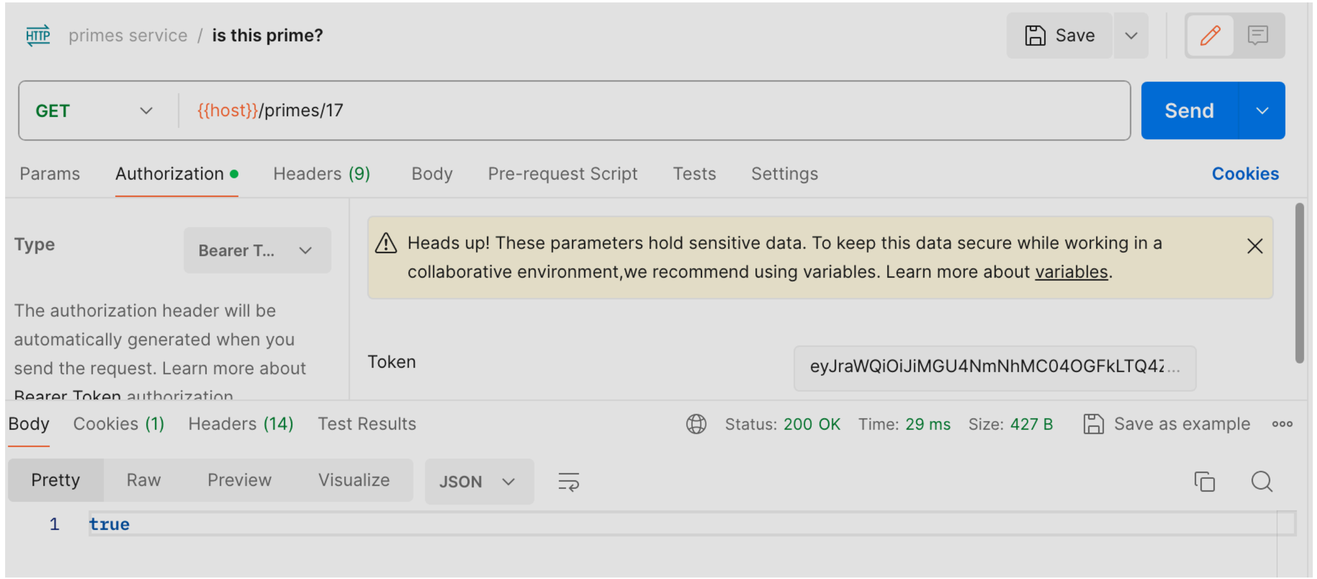
Steps

- In the PrimesController modify the endpoint to send a message containing the number and its primality status to the queue:



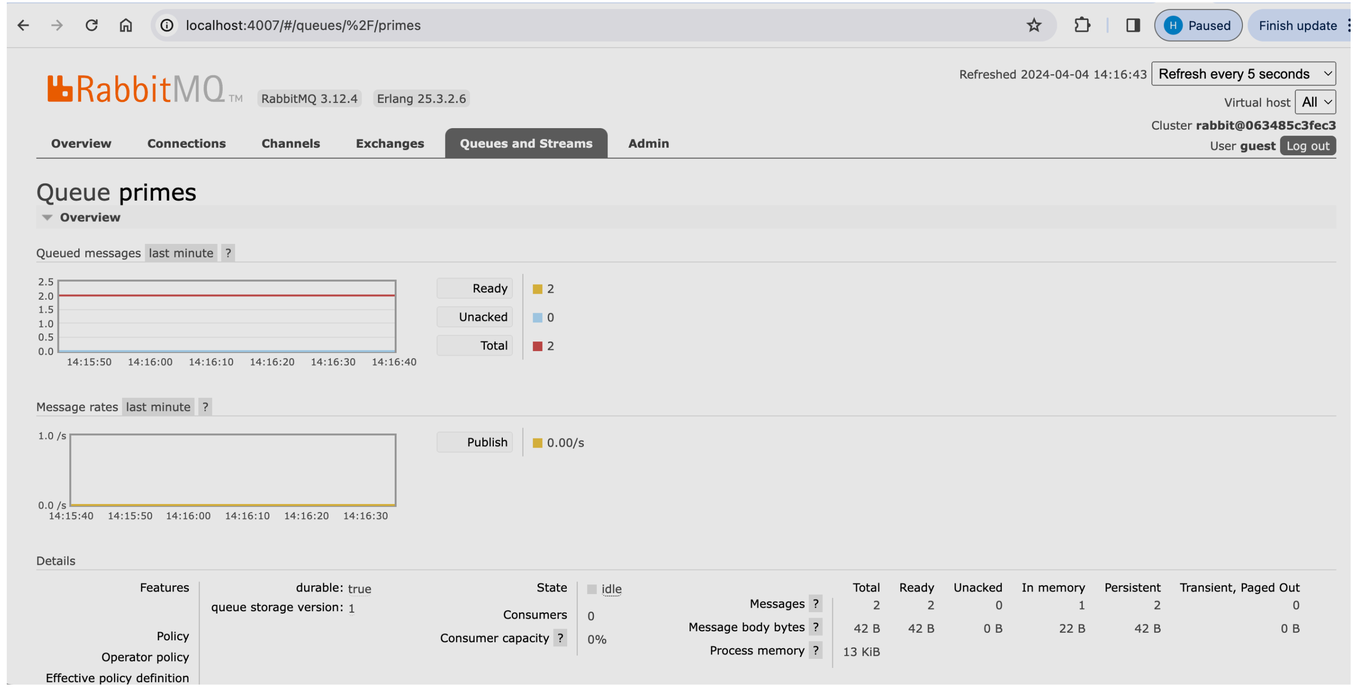
Steps

- Rerun your primes service application and using Postman call the prime check endpoint:



Steps

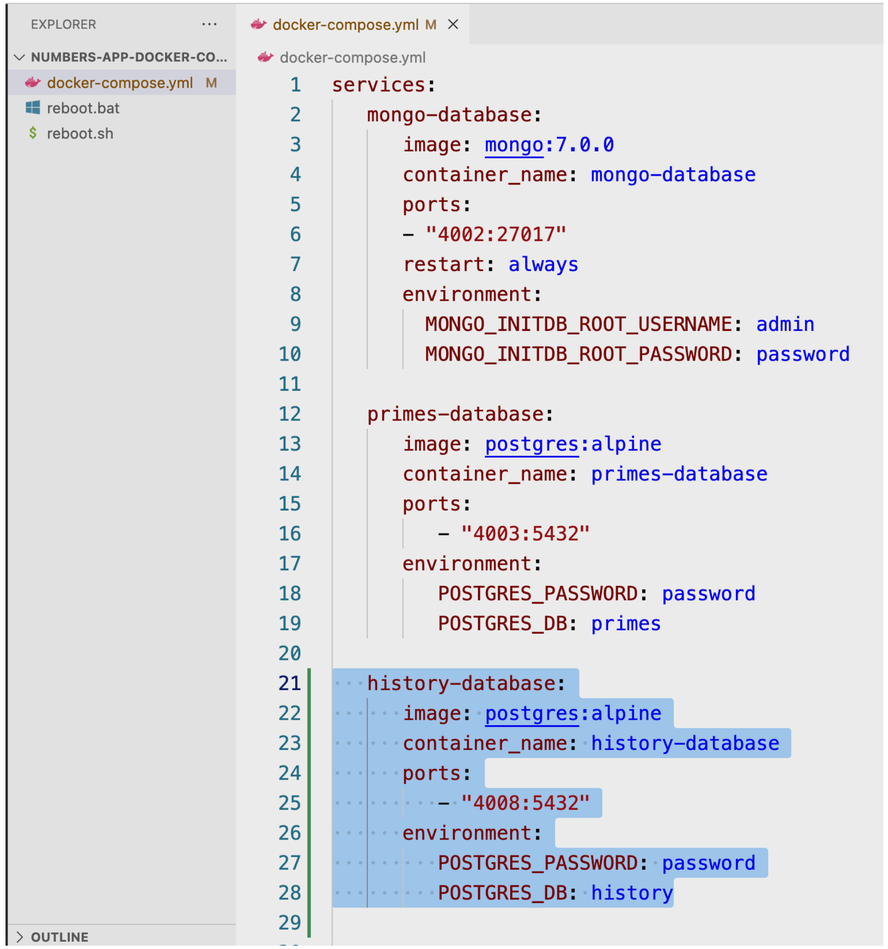
- Go to the RabbitMQ dashboard and verify that the message is received by the queue:



History Service

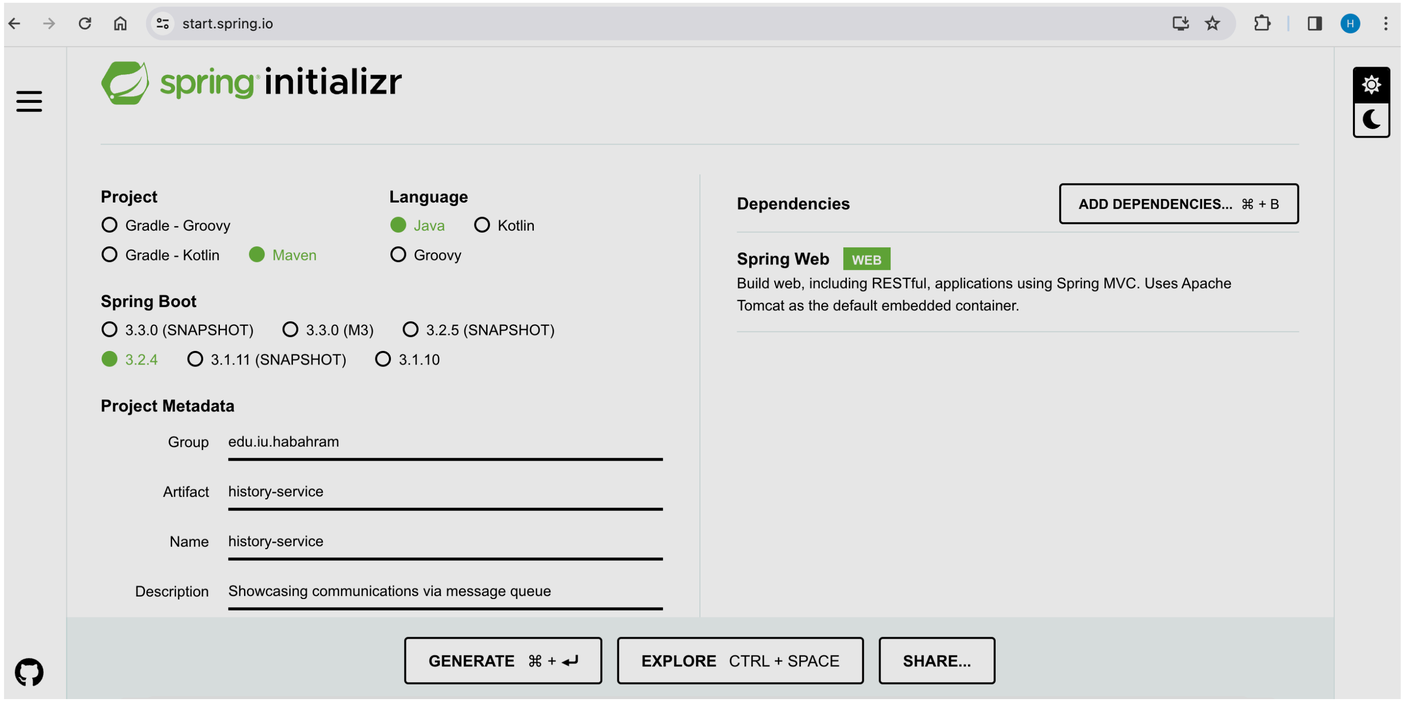
Steps

- First add a new database server to your docker-compose:



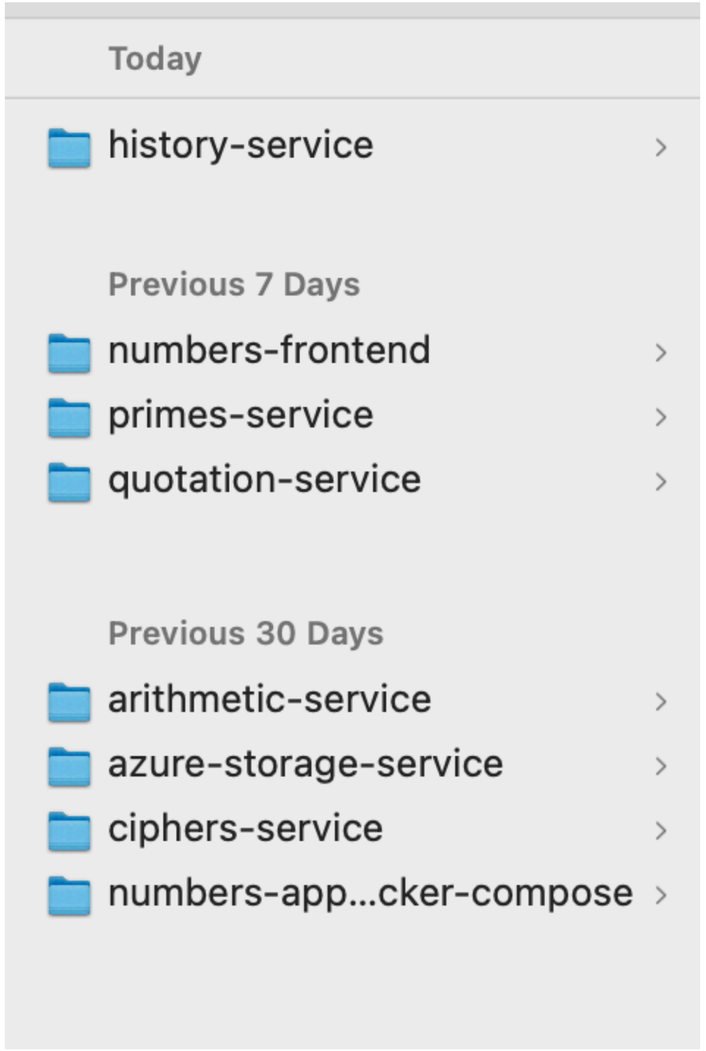
Steps

- Then create the service using spring initializer:
- Go to https://start.spring.io/ and select Java as the language, Maven as the build tool, and add Spring Web dependency.
- Generate the project and open it using IntelliJ IDEA.



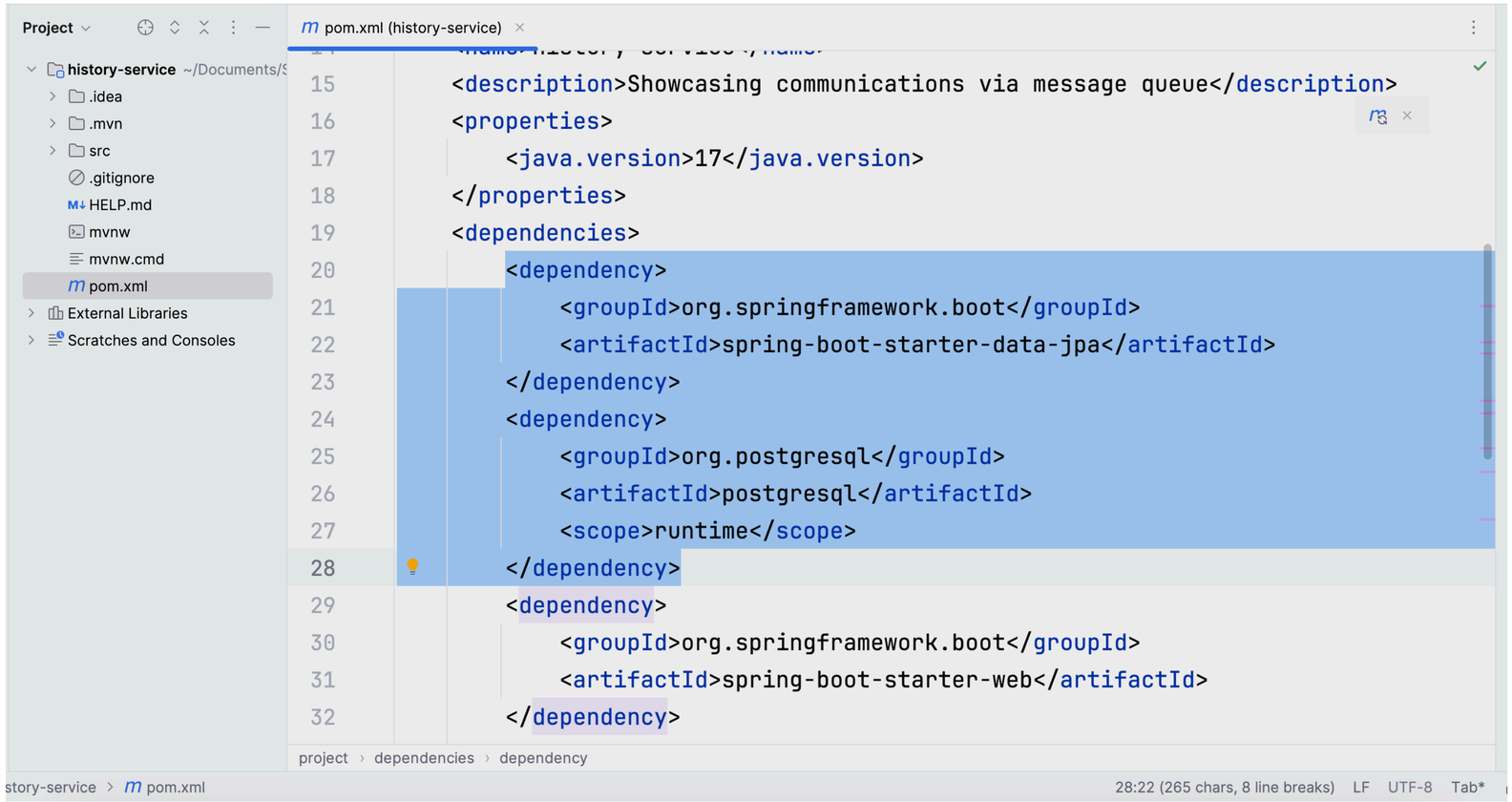
Steps

- Put the history service folder in the same folder where all the other services are located:



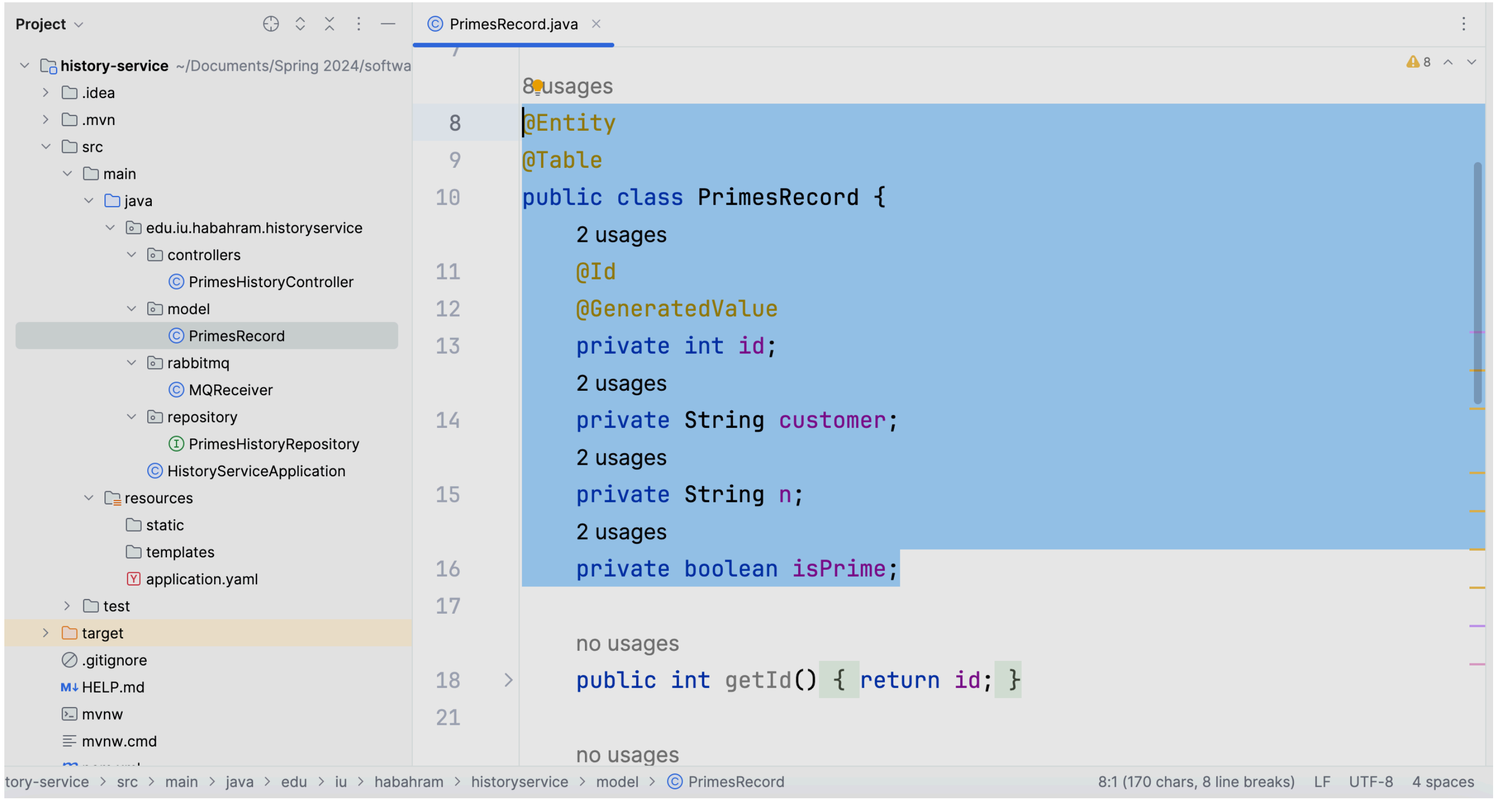
Steps

- Add JPA and Postgres dependencies to the project:



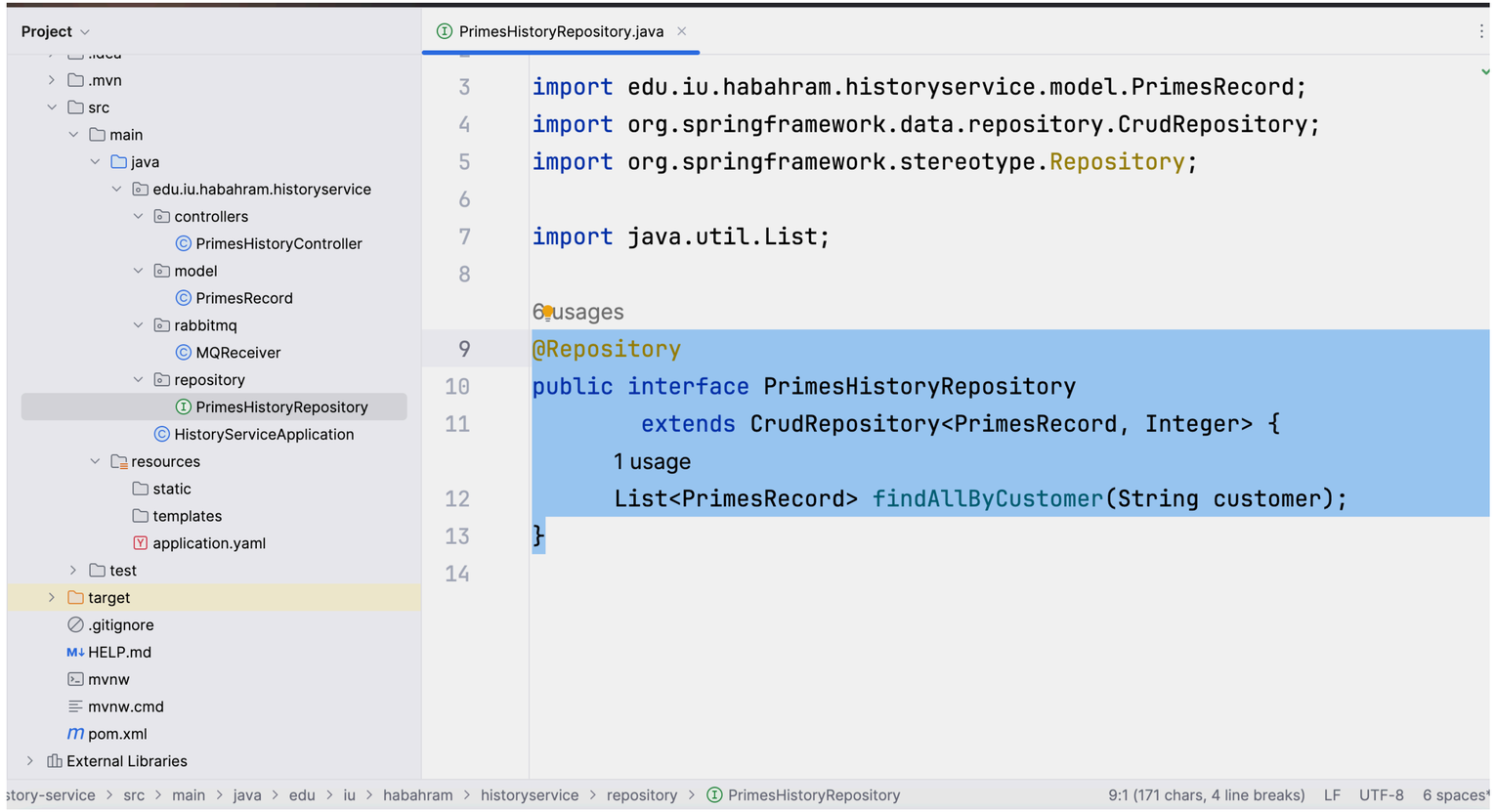
Steps

- Add a model, PrimesRecord, to store the history of primality checks:



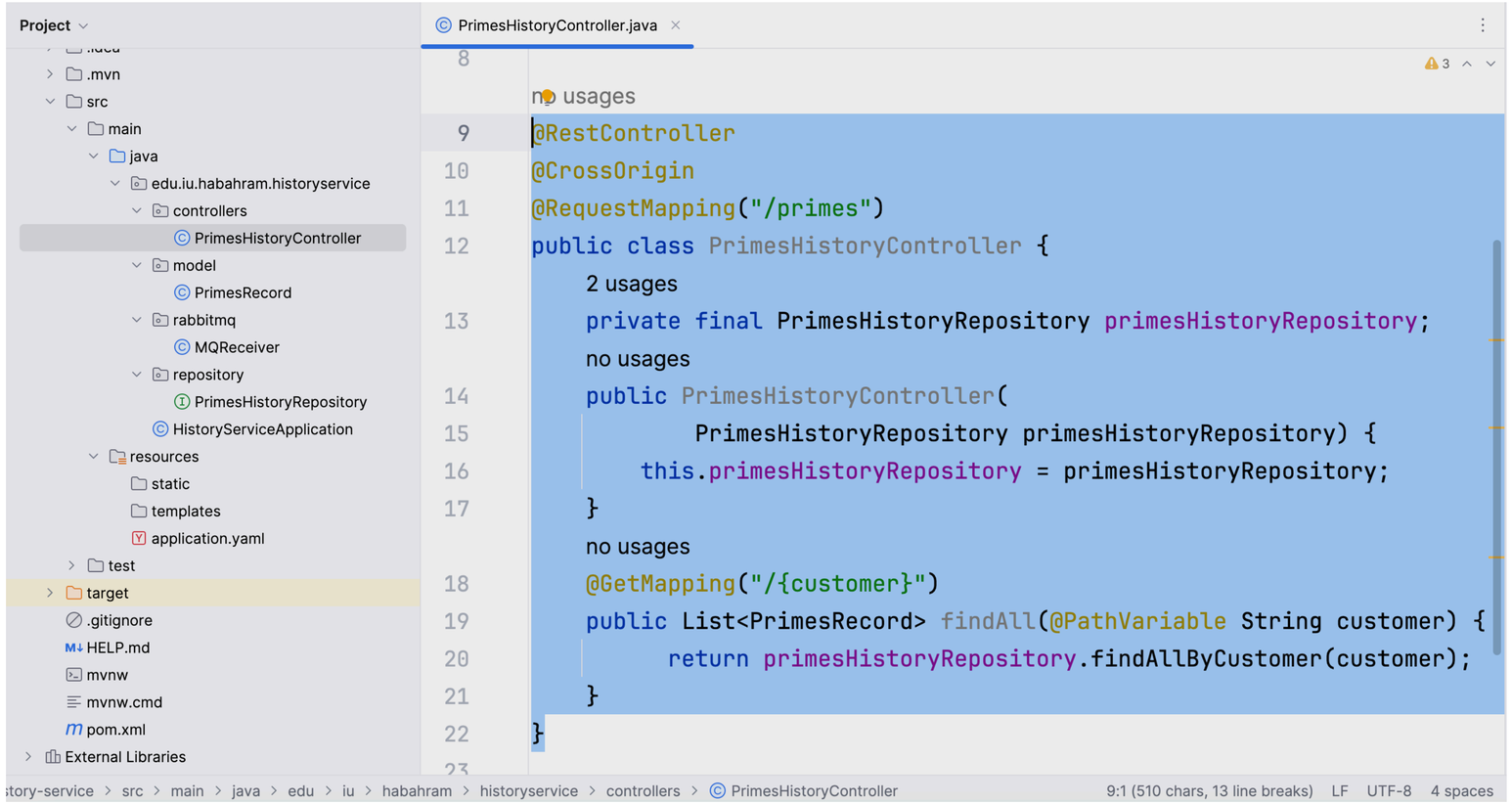
Steps

- Add the repository:



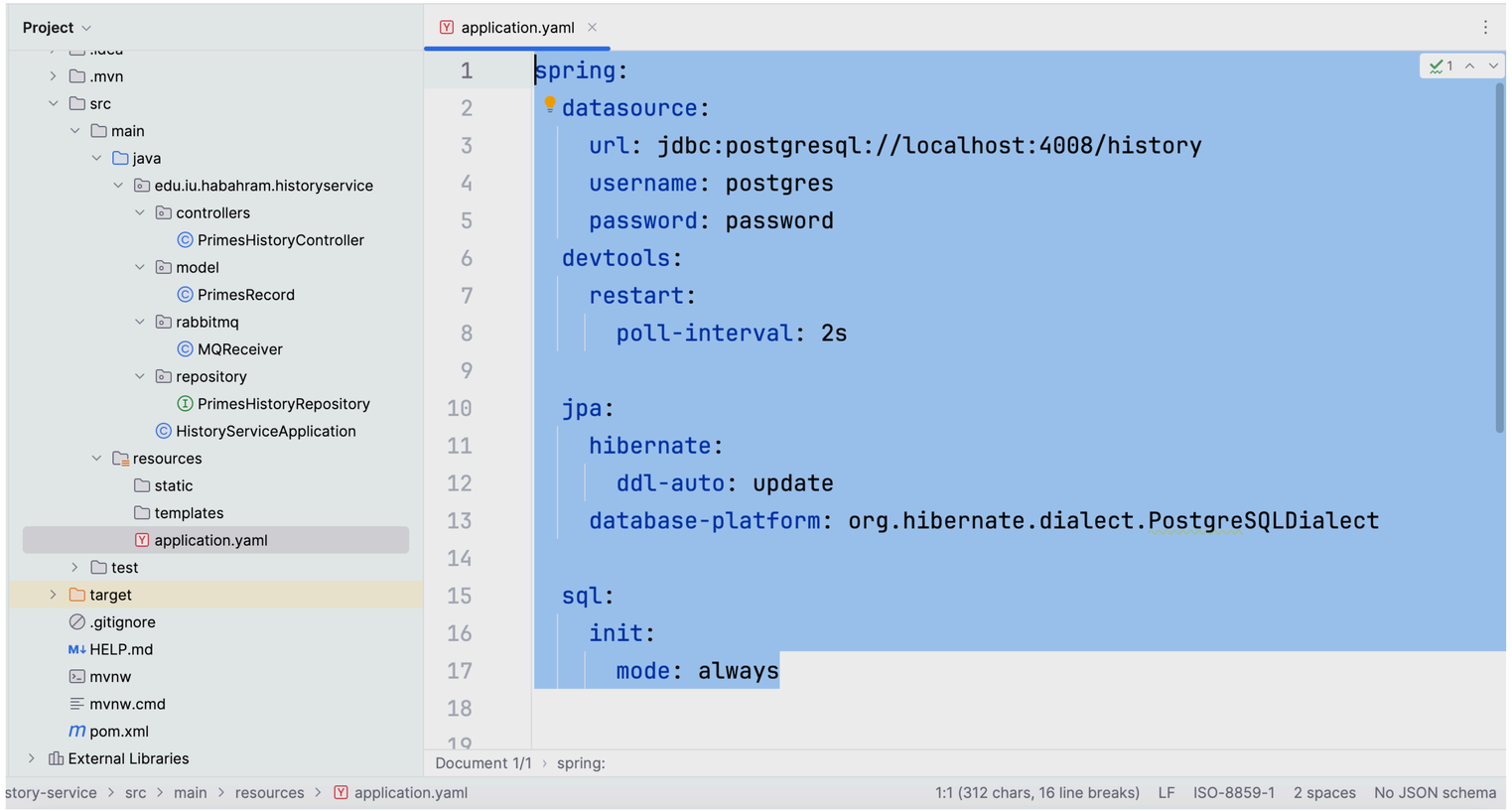
Steps

- Add the controller:



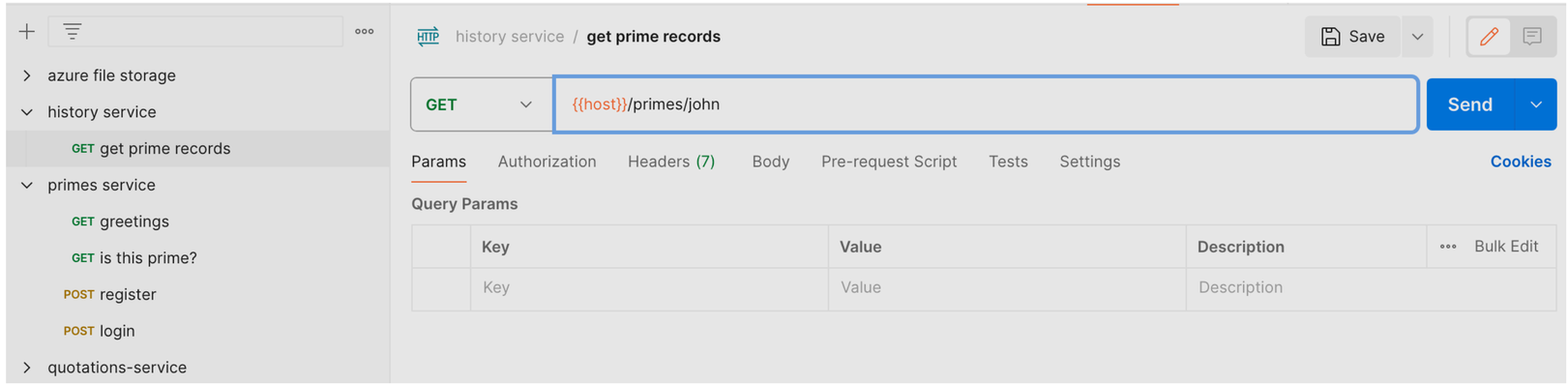
Steps

- Add the configuration properties:



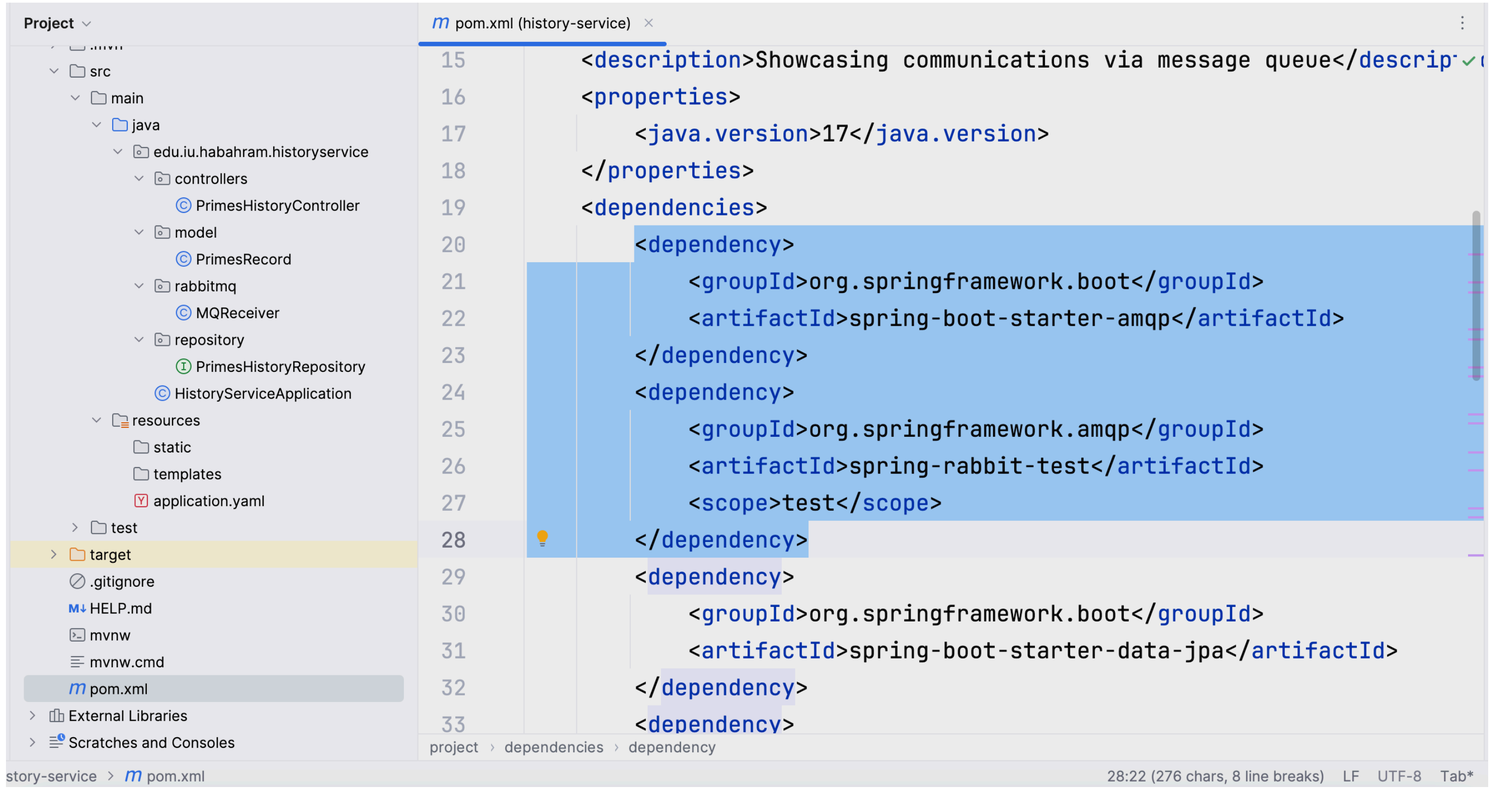
Steps

- Using postman verify that the history service is working. For now it should return an empty list.



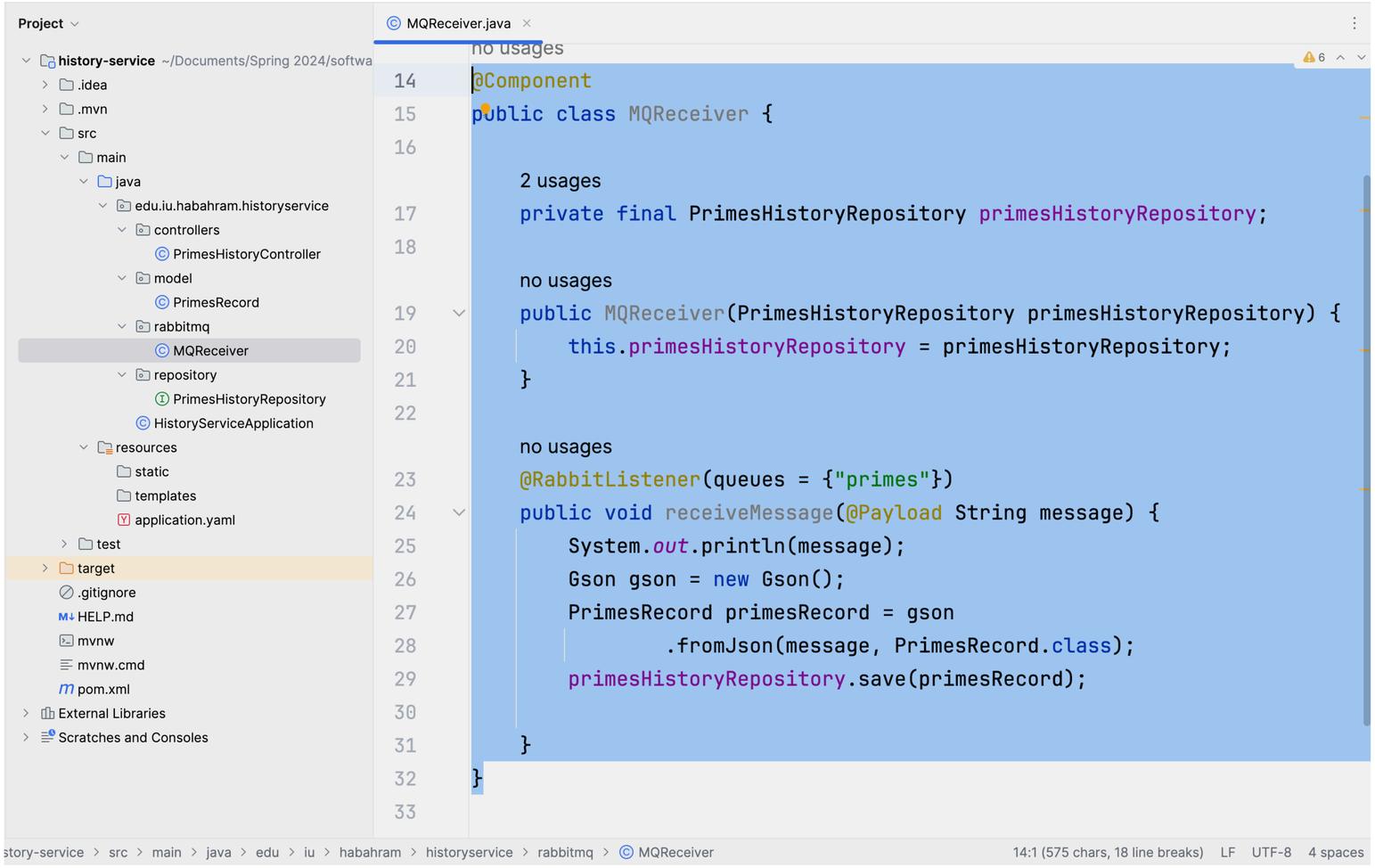
Steps

- Add the rabbitmq dependencies to the pom.xm:



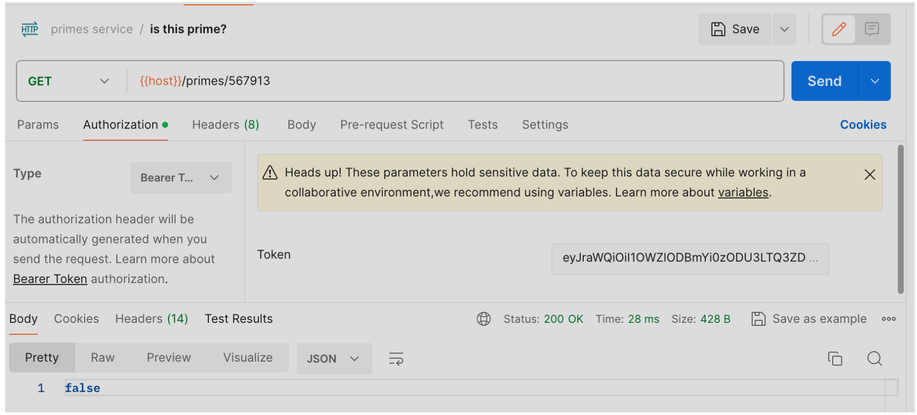
Steps

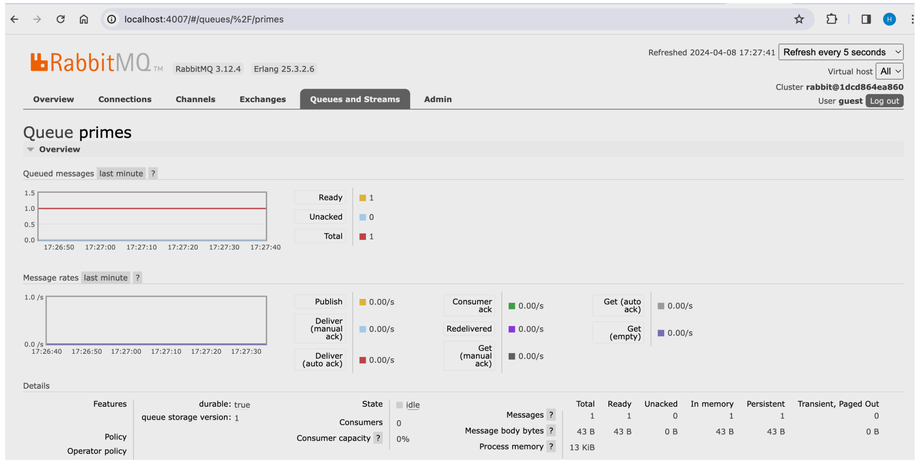
- Add a rabbitmq receiver that receives the messages from the queue and stores them in the Postgres database:

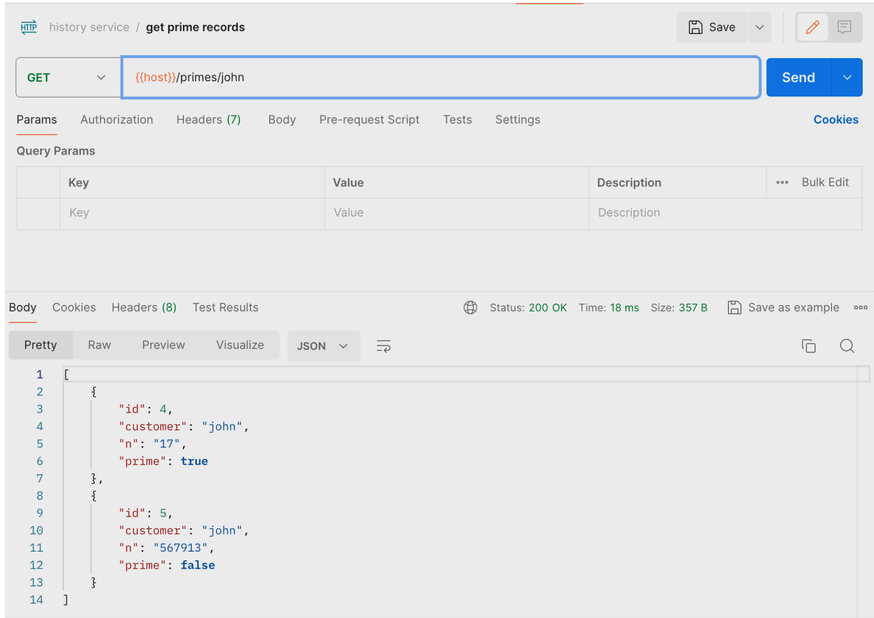


Steps

- Test the sender/ receiver:
- Using your primes services send a prime check request.
- Using your history service verify that the prime check request is stored in the database.







The END

- That was it for today! Congratulations!
- Submit the url of your primes service repository.
- Submit the url of your history service repository.