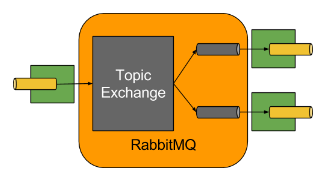
**RabbitMQ**

**Why messaging system ?**

In the micro services,most of the times we use rest or graph QA which are Asynchronous according to its architecture(We do not wait until it complete.).But to implement Rest or GraphQL we use HTTP protocol.It is Synchronous protocol.Because of that Rest or graphQl called become synchronou s.To avoid that the best solution is messaging system.



***RabbitMQ is a ordinary queue management protocol(First come first out) which use AMQP(Advanced messaging queue protocol) library.***

***Kafka used to process steams of messages.***

***Broker handle routing mechanisum(Content base routing header based routing) in rabbitMQ.Kafka has different architecture to achive that same thing.***

Fair dispatch with rabbitMQ(Prefetch 2, Consumer can hold max 2 messages)

RabbitMQ is a message broker/queue manager(message broker software) that implements advanced message queuing protocol(AMOP).But It also support several other protocols such as STOMP,MQTT and HTTP.

Messaging increase loose coupling and scalability.

AMOP standards messaging using producers, Brokers and consumers.

01.Producer - produce the message

02.Broker-Hold the message

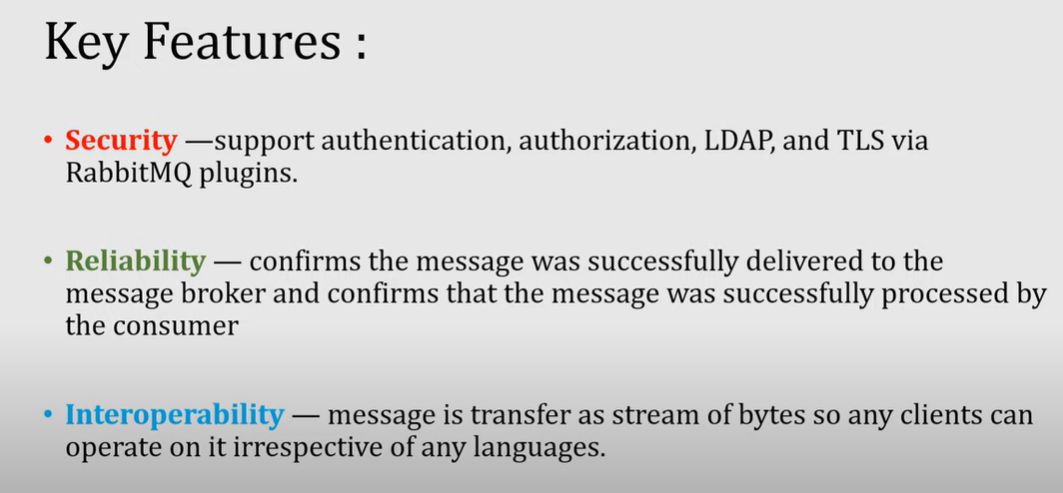
03.Consumer-Consume the message

AMOP standards was designed with the following main characteristics.

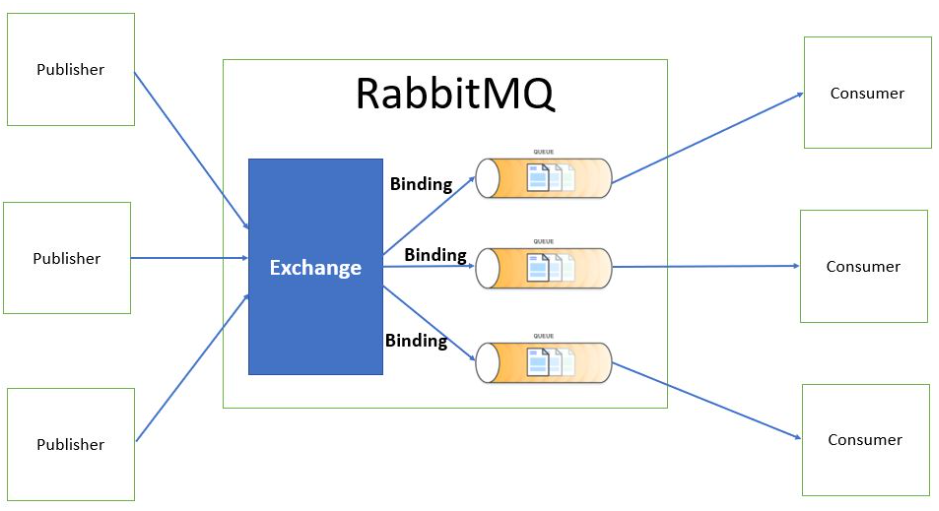
01.Security

02.Reliability

03.Interoperability.



How AMQP work



Producer emits messages to exchanges.

Consumer recives messages from queue.

Binding connect an exchange with a queue using binary key(Link between a queue and an exchanged).

Routing Key - is a key that exchange looks like at to decide how to route the message to queue.

Exchange compares routing key with binding key(Sends the messages to one or more queue by following a specific set of rules).

Message exchange depends on exchange types(Fanout ,Direct ,Topic,Header and Default(nameless) exchange )Information that sent from producer to a consumer.

RabbitMQ have the following types of Exchanges to sends message from Exchanger to queues.

01.Fanout - Sending same message to all queue

02.Direct - Sending messages to queues which match with [routing key = binding key]

03.Topic - exchange message for partially match of key.

04.Default(nameless) exchange - Sending messages to queue by considering queue name instead of routing key(Routing Key = queue name).

05.Header - Sending messages to queue by considering message header instead of routing key.

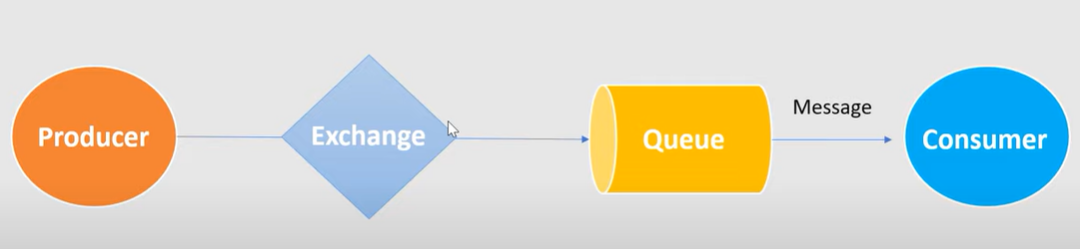
***NOTE: These types (Fanout ,Direct ,Topic ,Default,Header ) have been be deeply described in a next post.***

**Traditional queue**

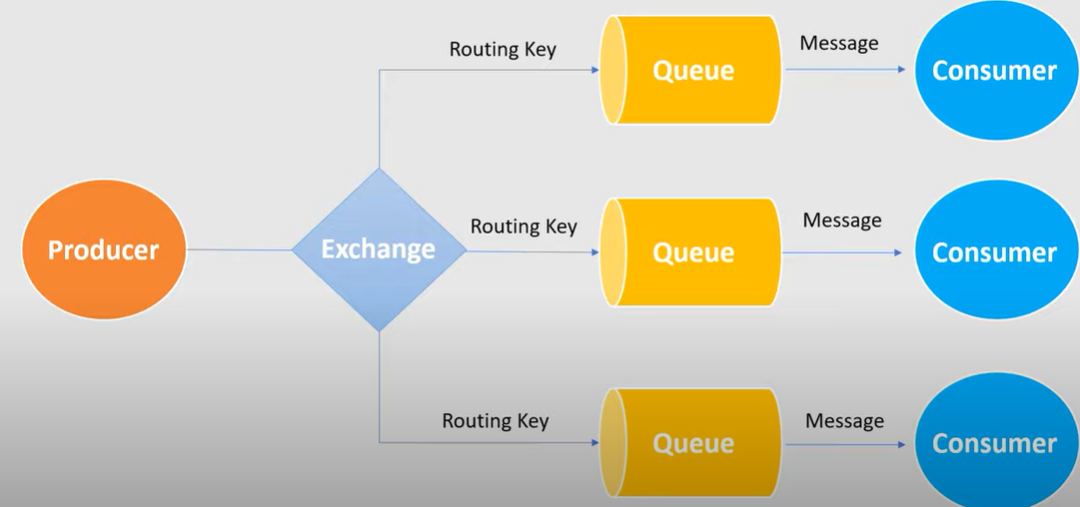


**In RabbtiMQ**

It is not a normal queue system.It is a queue management system and to achieve it, RabbitMQ uses a technique call EXCHANGE.



**RabbtiMQ in advance**



# **Exchange Types**

The ways which exchanger sends message to queues.

01.Fanout - Sending same message to all queue

02.Direct - Sending messages to queues which match with [routing key = binding key]

03.Topic - exchange message for partially match of key.

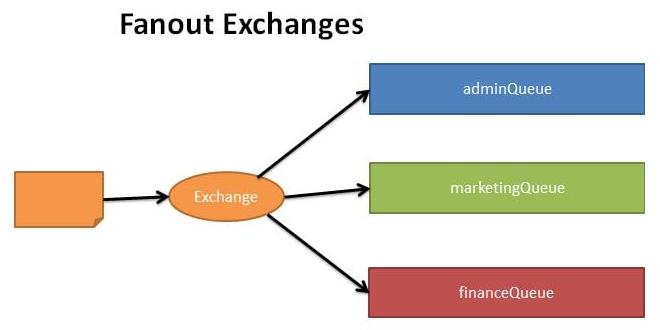
04.Default(nameless) exchange - Sending messages to queue by considering queue name instead of routing key(Routing Key = queue name).

05.Header - Sending messages to queue by considering message header instead of routing key.

Support Tutorial : <https://www.javainuse.com/messaging/rabbitmq/exchange>

01.Fanout

The message is routed to all the available bounded queues. The routing key if provided is completely ignored. So this is a kind of publish-subscribe design pattern.



Normal way

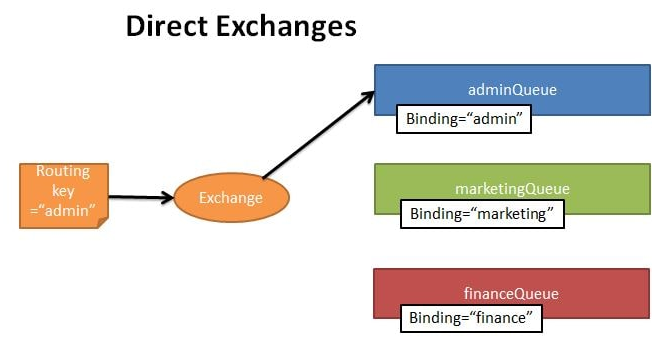
rabbitTemplate.convertAndSend(name of the exchange, “routingKey”,messageDataDto);

Set routing key as empty to achieve Fanout exchange.

rabbitTemplate.convertAndSend(name of the exchange, “”, messageDataDto)

02.Direct

Based on the routing key a message is sent to the queue having the same routing key specified in the binding rule. The routing key of exchange and the binding queue have to be an exact match. A message is sent to exactly one queue.

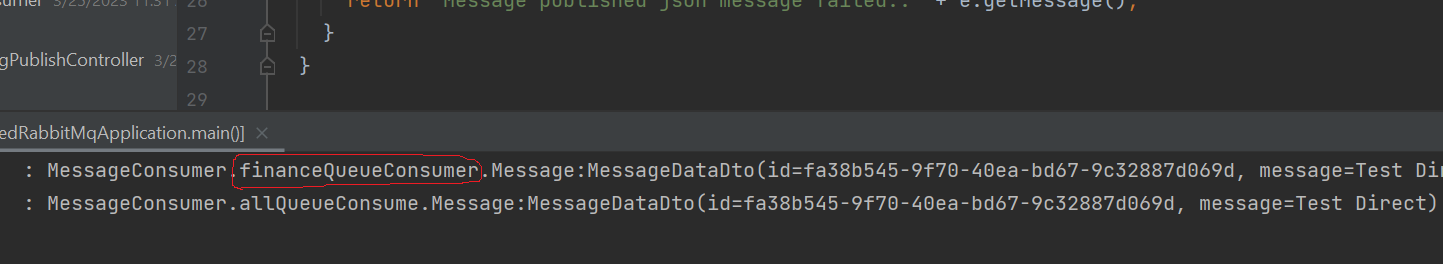


Need to provide the routing key.Then given routing key is considered with binding keys by the Exchanger.

@Bean  
Binding allBinding(Queue allQueue, TopicExchange normalTopicExchange) {  
 return BindingBuilder.*bind*(allQueue).to(normalTopicExchange).with(*“*routingKey*”*);  
}

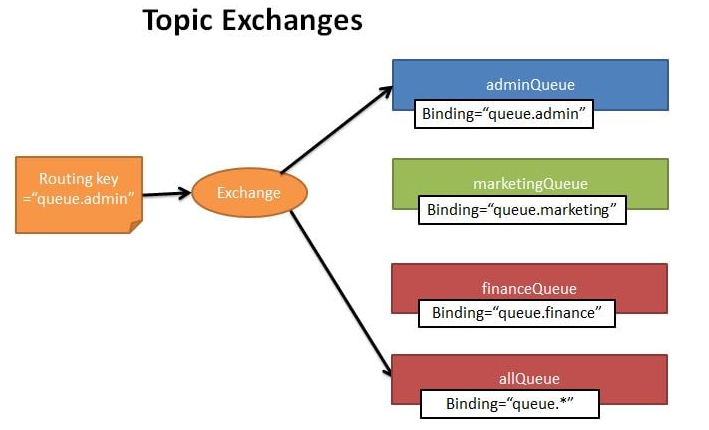
rabbitTemplate.convertAndSend(name of the exchange, “routingKey”, messageDataDto);

<http://localhost:8080/publish-message/normal-exchange/%20queue.finance-key/Test-Dirrect>



03.Topic

Here again the routing key is made use of. But unlike in direct exchange type, here the routing key of the exchange and the bound queues should not necessarily be an exact match. Using regular expressions like wildcard we can send the exchange to multiple bound queues.

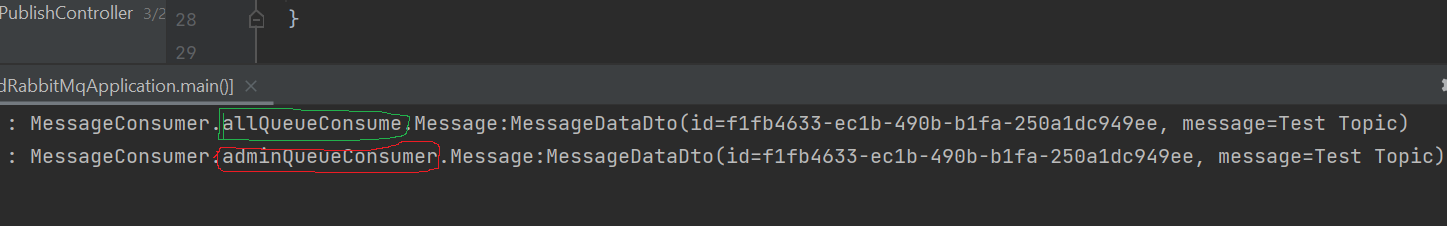


Routing key with “queue.any-value” will be send to the queue which have the binding key of “queue.any-value” and send bellow binding queue as well.

@Bean  
Binding allBinding(Queue allQueue, TopicExchange normalTopicExchange) {  
 return BindingBuilder.*bind*(allQueue).to(normalTopicExchange).with(*“*queue.\**”*);  
}

rabbitTemplate.convertAndSend(name of the exchange, “queue.abc”, messageDataDto);

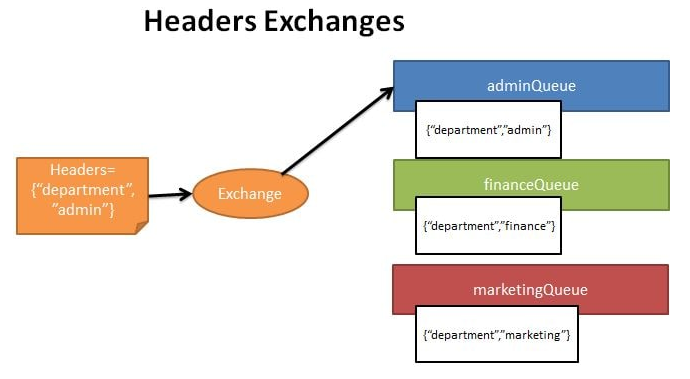
<http://localhost:8080/publish-message/normal-exchange/%20queue.admin-key/Test-Topic>



04.Default(nameless) exchange

05.Header

In this type of exchange the routing queue is selected based on the criteria specified in the headers instead of the routing key.



// Header Exchange testing...  
@Bean  
HeadersExchange headerTopicExchange() {  
 return new HeadersExchange("HEADER\_EXCHANGE");  
}  
// Binding Queues with Header Topic Exchange…

@Bean  
Binding marketingBindingForHeaderExchange(  
 Queue marketingQueue, HeadersExchange headerTopicExchange) {  
 return BindingBuilder.*bind*(marketingQueue)  
 .to(headerTopicExchange)  
 .where("DEPARTMENT")  
 .matches("ADMIN");  
}

Produce a message

MessageProperties messageProperties = new MessageProperties();  
messageProperties.setHeader(“*DEPARTMENT”*, "ADMIN");  
MessageConverter messageConverter = new SimpleMessageConverter();  
Message message = messageConverter.toMessage(messageDataDto, messageProperties);  
rabbitTemplate.send(exchange, "", message);

<http://localhost:8080/publish-message/by-header/header-exchange/finance%20/Header-exchange-test>

**How to delete messages from the queue**

Queue sends message to the consumer, After that consumer will send acknowledgement to the queue to delete message from the queue(After consumer completes the message related process, this will be send because message is not needed to hold in queue after it proceed)

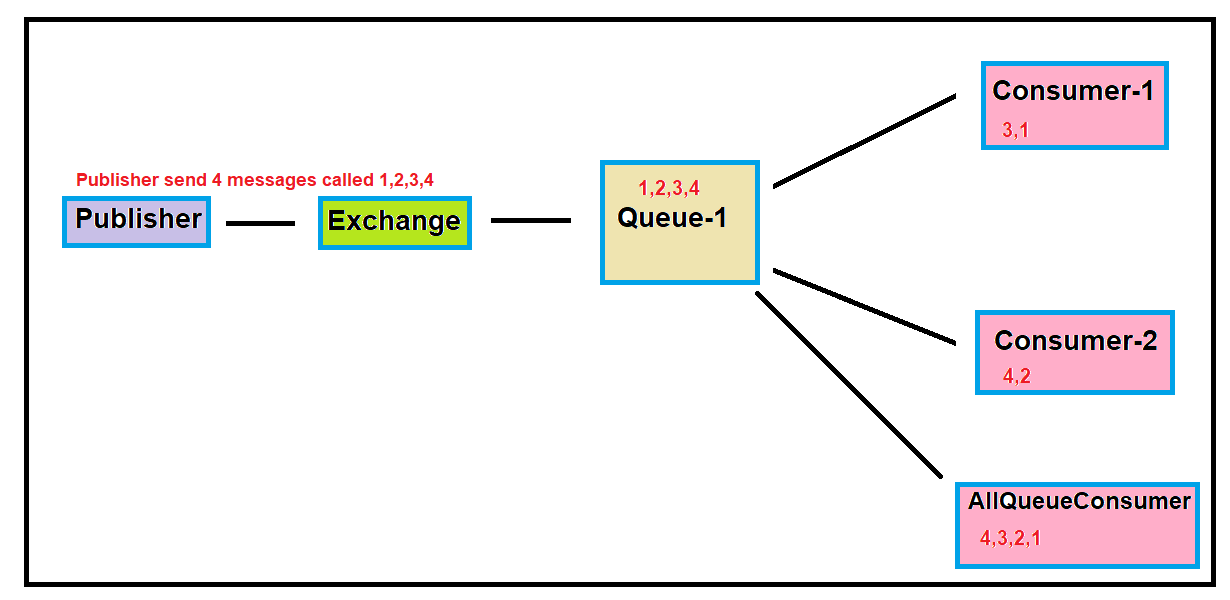
**Type of sending messages to consumers by a Queue.**

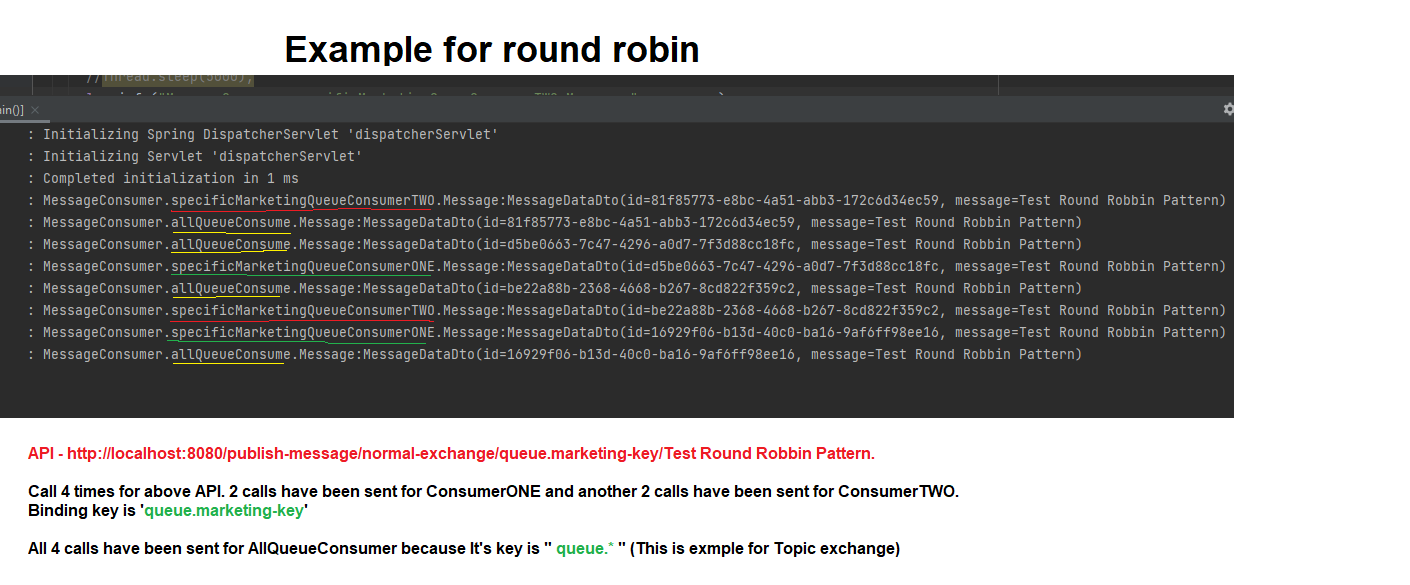
01.Round-robin

02.Fair dispatch with rabbitMQ(Ex.Prefetch 2, Consumer can hold max 2 uncompleted messages)

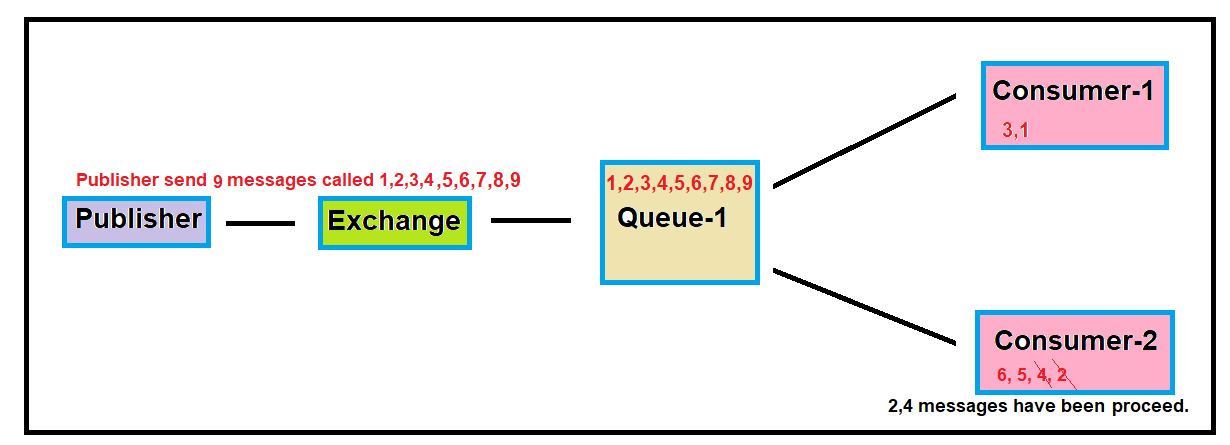
**01.Round-Robin**

**<http://localhost:8080/publish-message/normal-exchange/queue.marketing-key/Test-Round-Robbin-Pattern>**

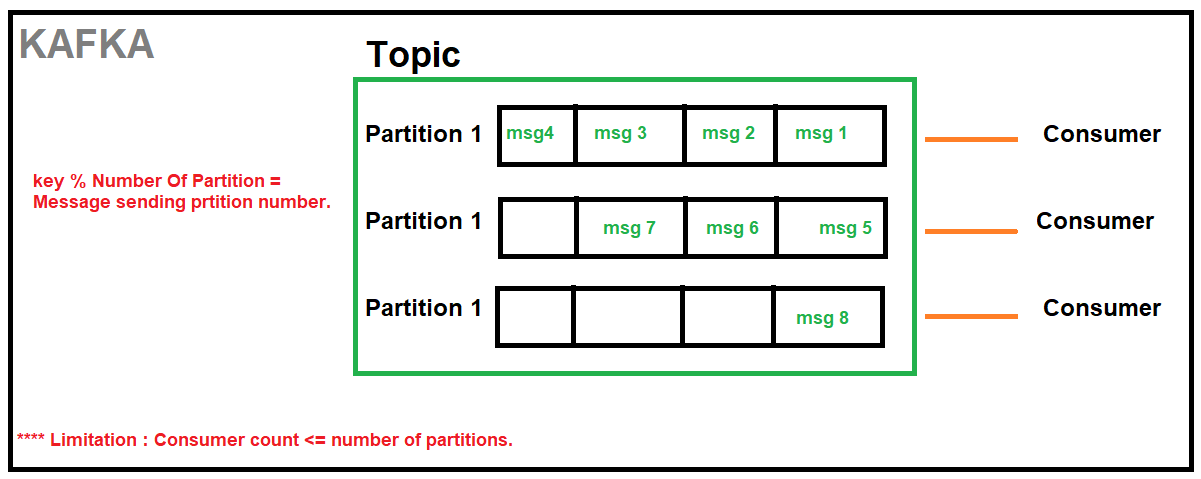
****

****

**02.Fair dispatch with rabbitMQ(Ex.Prefetch 2, Consumer can hold max 2 uncompleted messages)**

****

**In the Kafka this will be achieved by topic partitions.**

****

**Installing RabiitMQ to Windows**

Guide:

<https://www.youtube.com/watch?v=V9DWKbalbWQ&ab_channel=TechnicalBabaji>

NOTE: ERlang is need to run RbbitMQ and ERlang version should be compatible with rabbitMQ version.

<https://www.rabbitmq.com/which-erlang.html>

01.Download and install ERlang

<https://www.erlang.org/downloads>

02.Download and install RabbitMQ

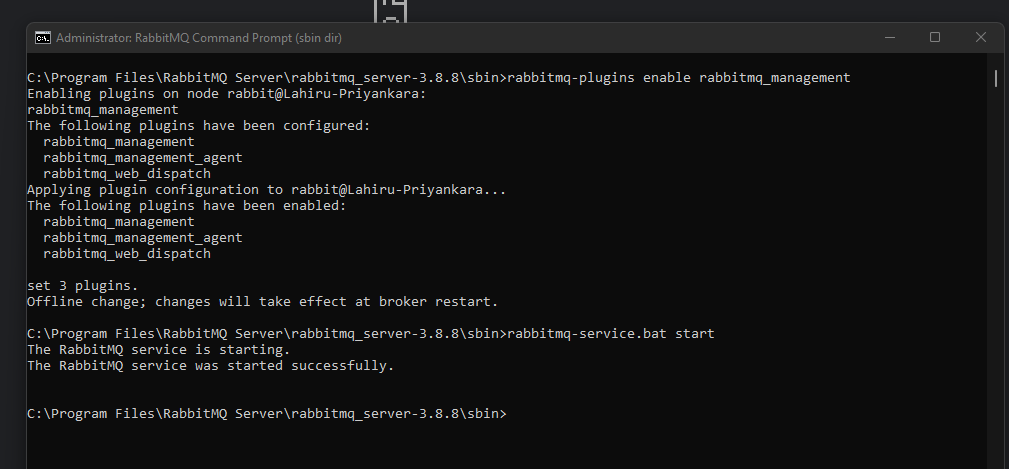
<https://www.rabbitmq.com/install-windows.html>

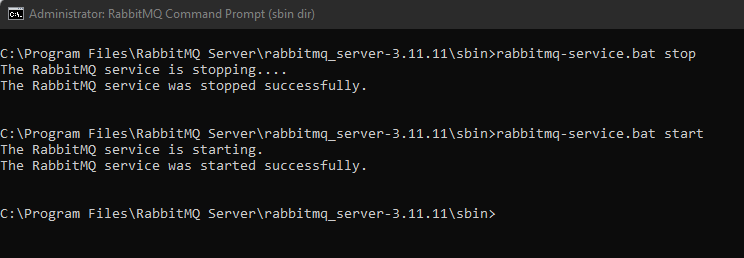
[ RabbitMQ Server install Directory - C:\Program Files\RabbitMQ Server\rabbitmq\_server-3.8.3\sbin ]

03.Search rabitMQ command prompt and open it.

04.Run command - “rabbitmq-plugins enable rabbitmq\_management”

05.Run command - “rabbitmq-service.bat start”



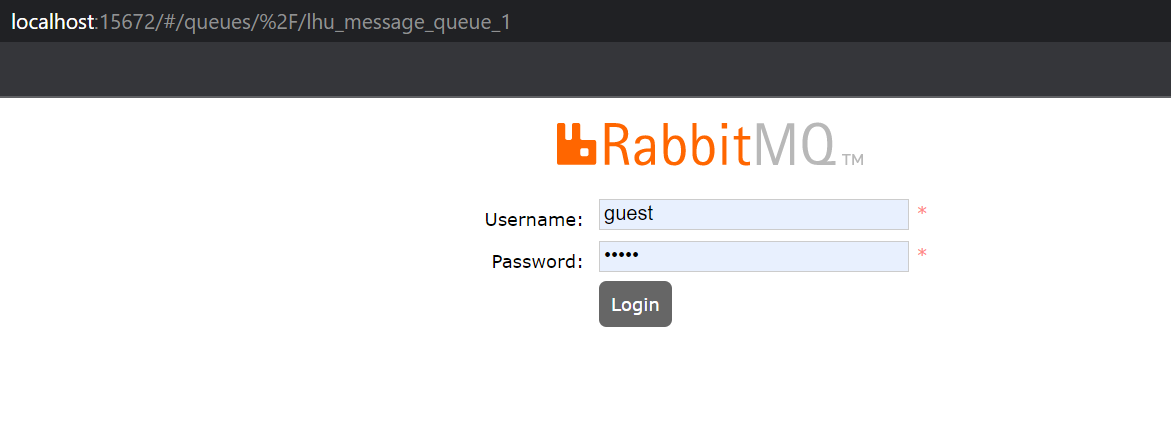


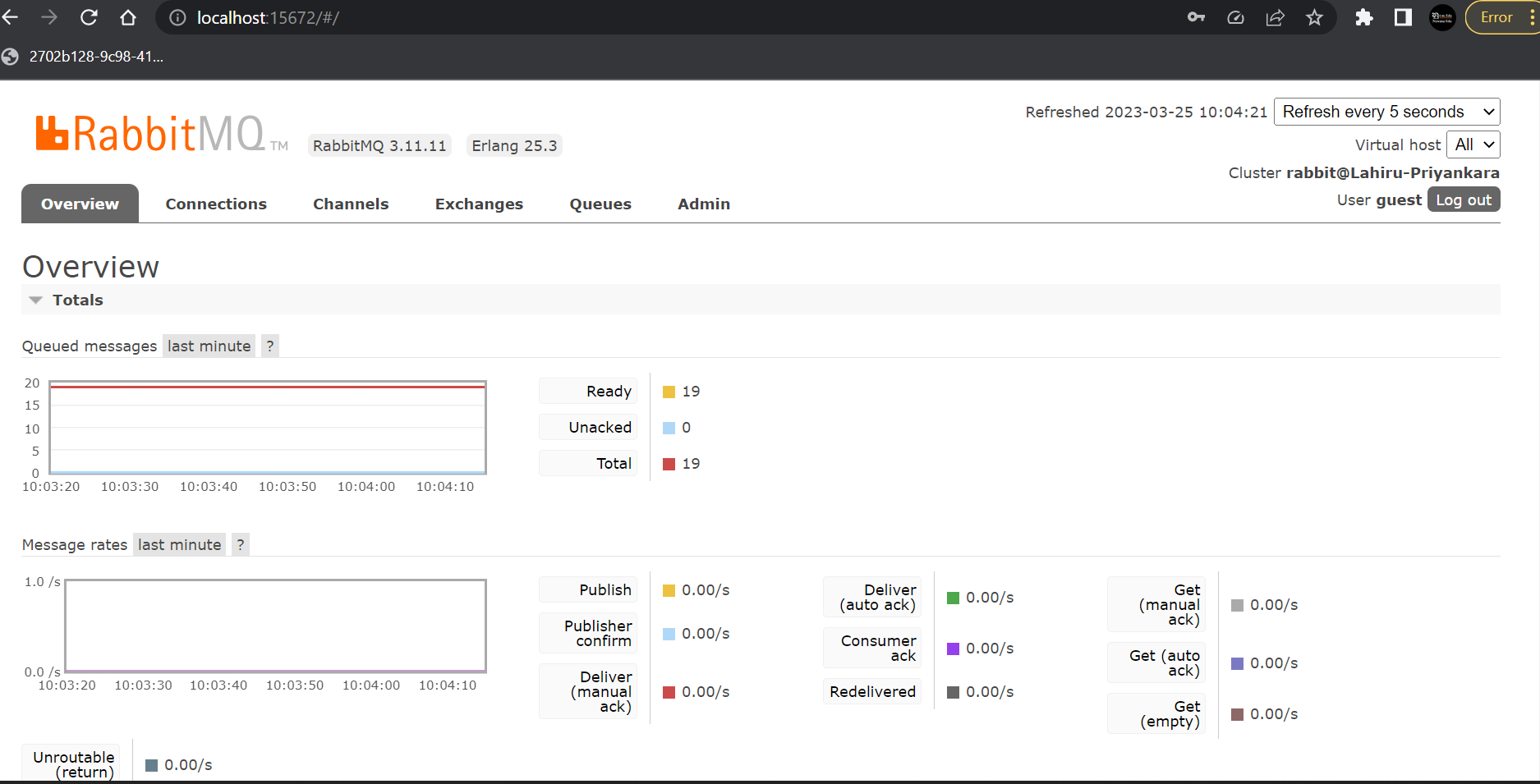
06.Open browser and enter bellow to redirect to RabbitMQ Dashboard

<http://localhost:15672/> or  [http://127.0.0.1:15672](http://127.0.0.1:15672))

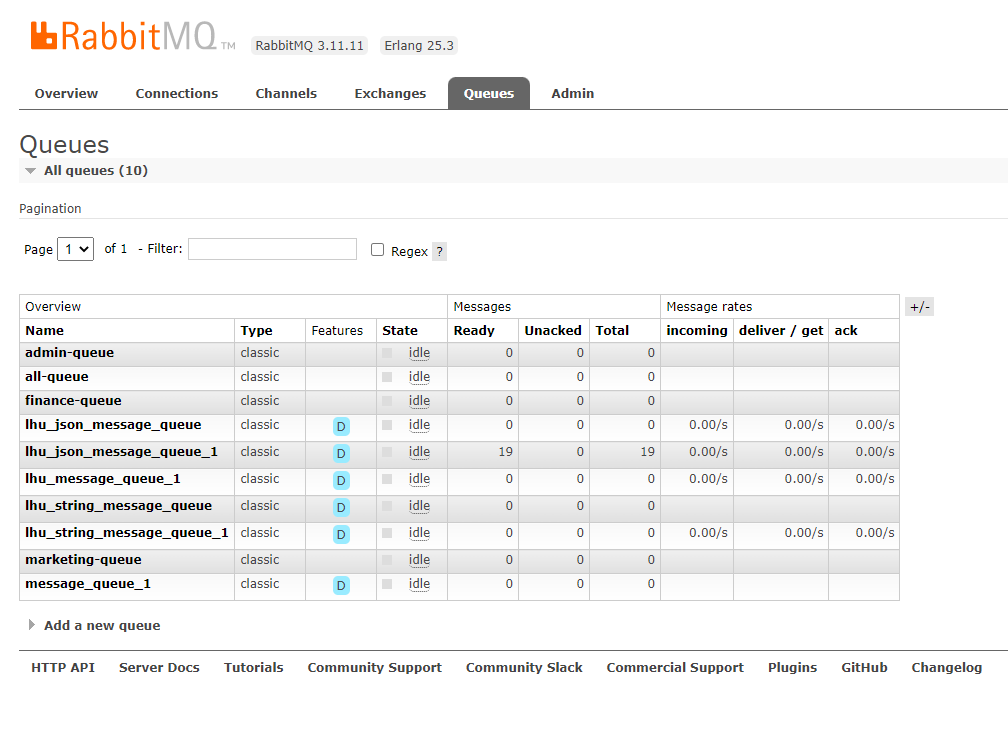
07.Login page default username = guest and default password = guest

08.After successfully login you should see RabbitMQ Home page

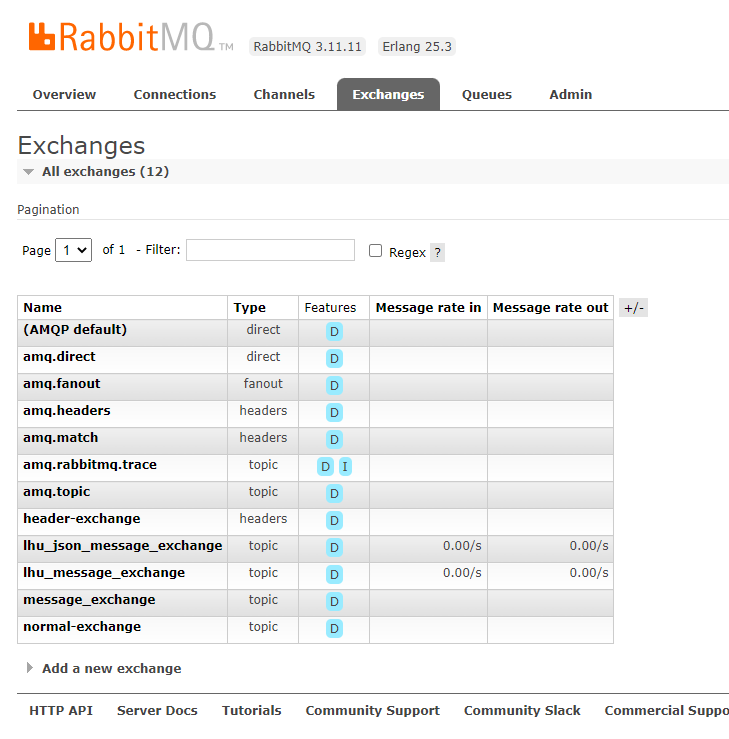




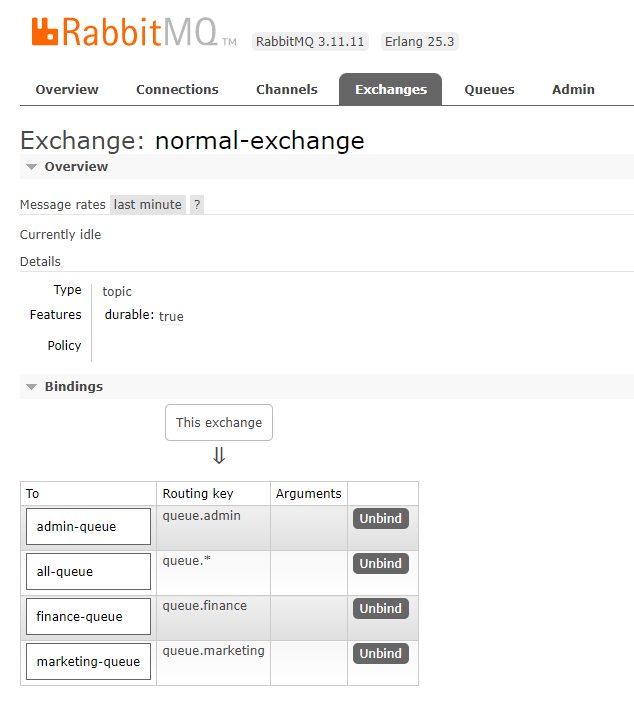
**Created Queues**



**Craeted Exchanges.**

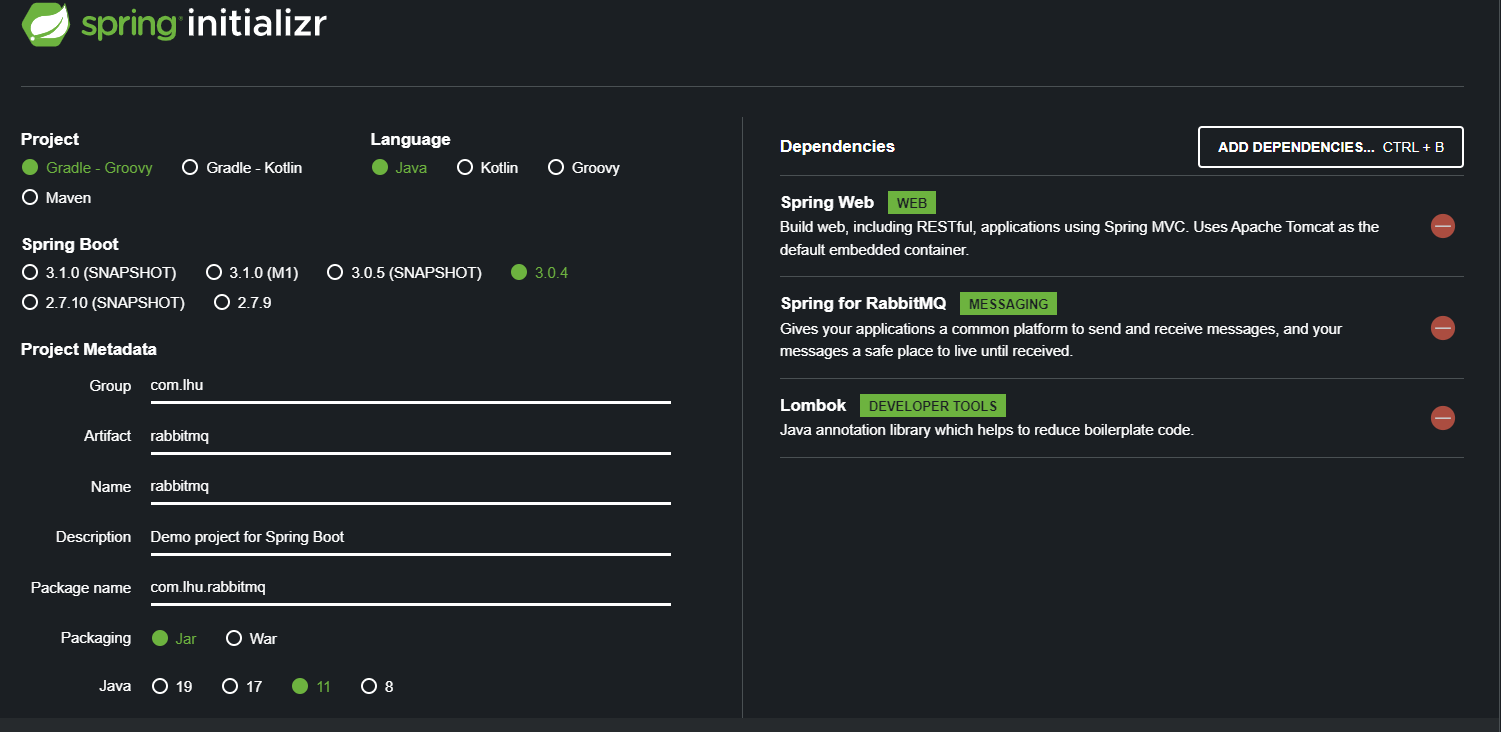


**Bind Queue with normal-exchange.**



**RabbitMQ Implementation with Spring Boot.**

There two spring boot projects have been develop by adding bellow dependencies to describe about the RabbitMQ behavior.One project contain basic working flow and other project implements rabbitMQ features in advanced. Both projects can be downloaded from bellow GitHub link. *Not:advancedRabbitMQ project is still implementing(Still not fully covered all the things.)*



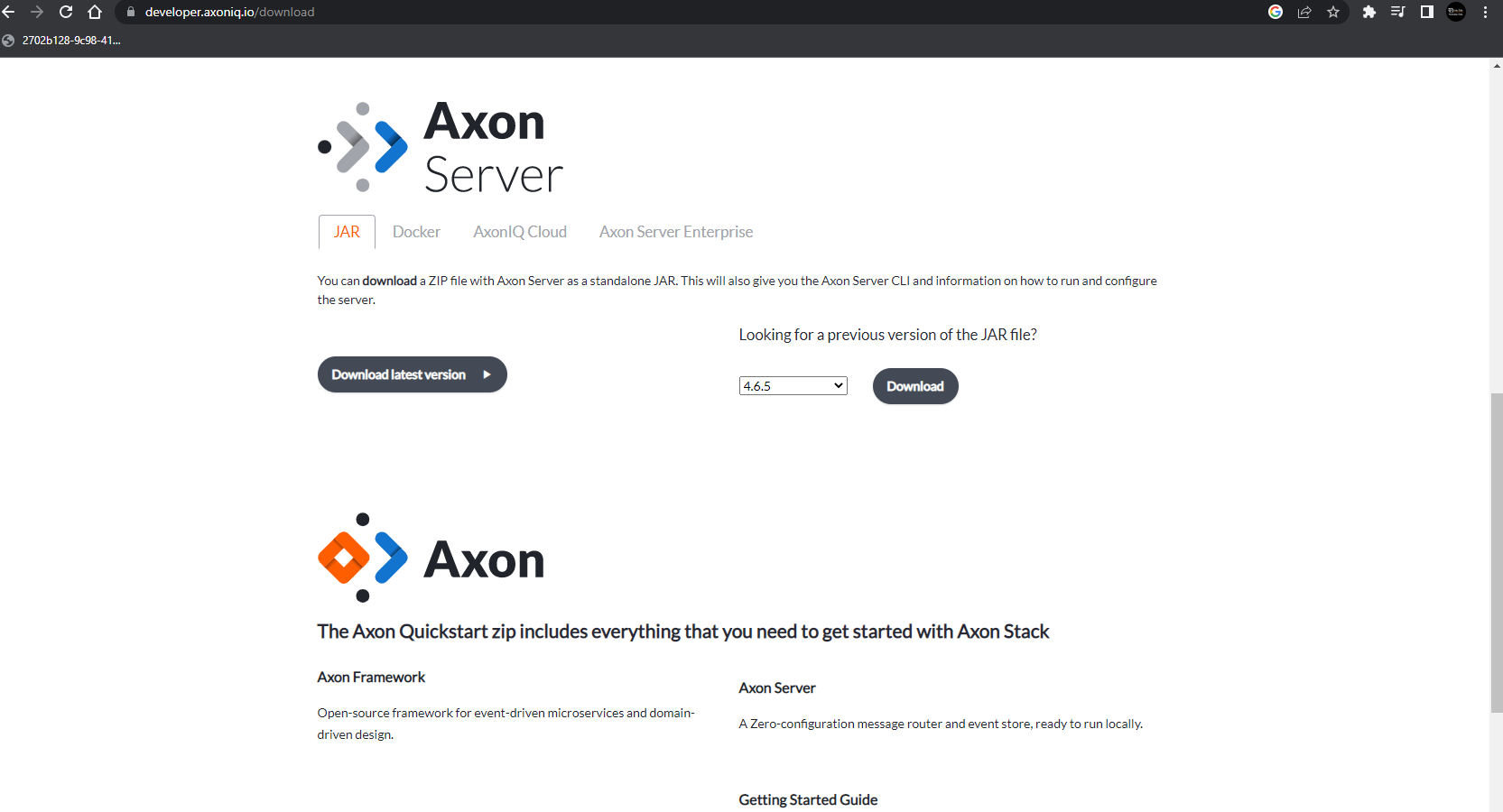
..

SAGA - OCHASTRATION

Instal axon server..

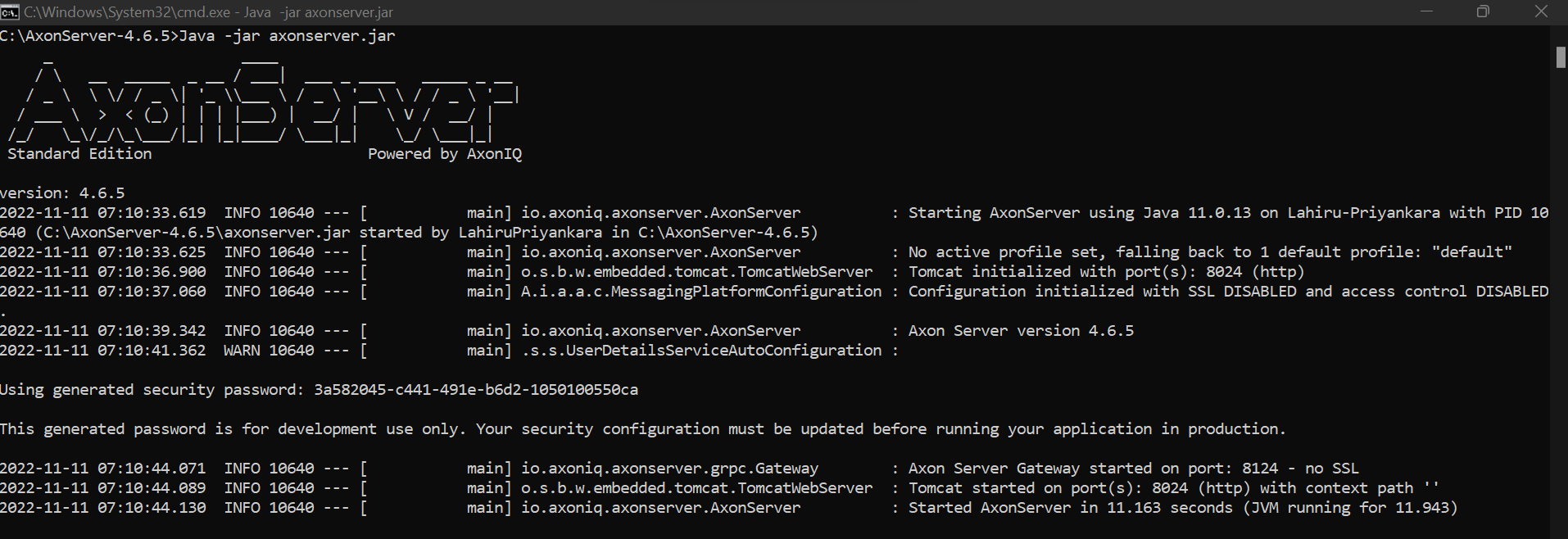
Got to

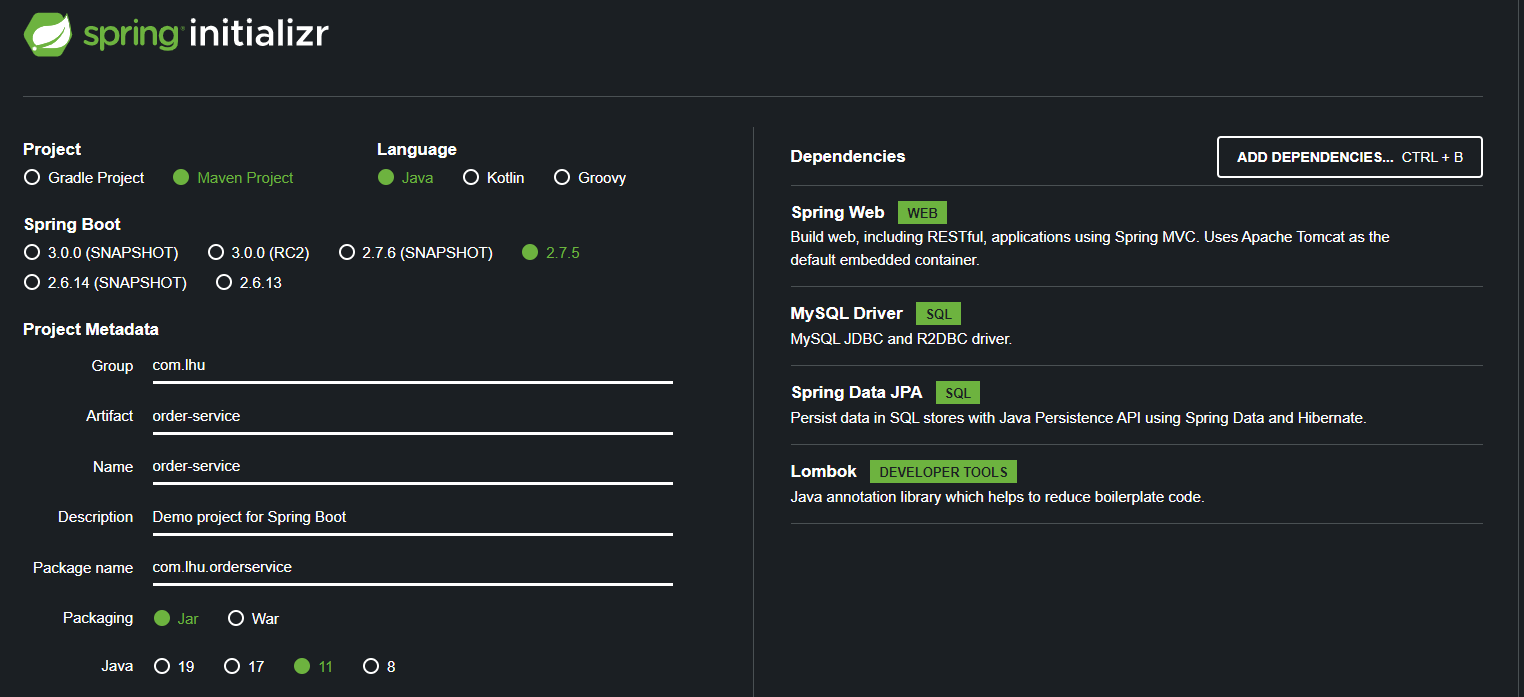
<https://developer.axoniq.io/download>



UnZip and go in to folder and run bellow command and Server will start on port:8124

Java -jar axonserver.jar





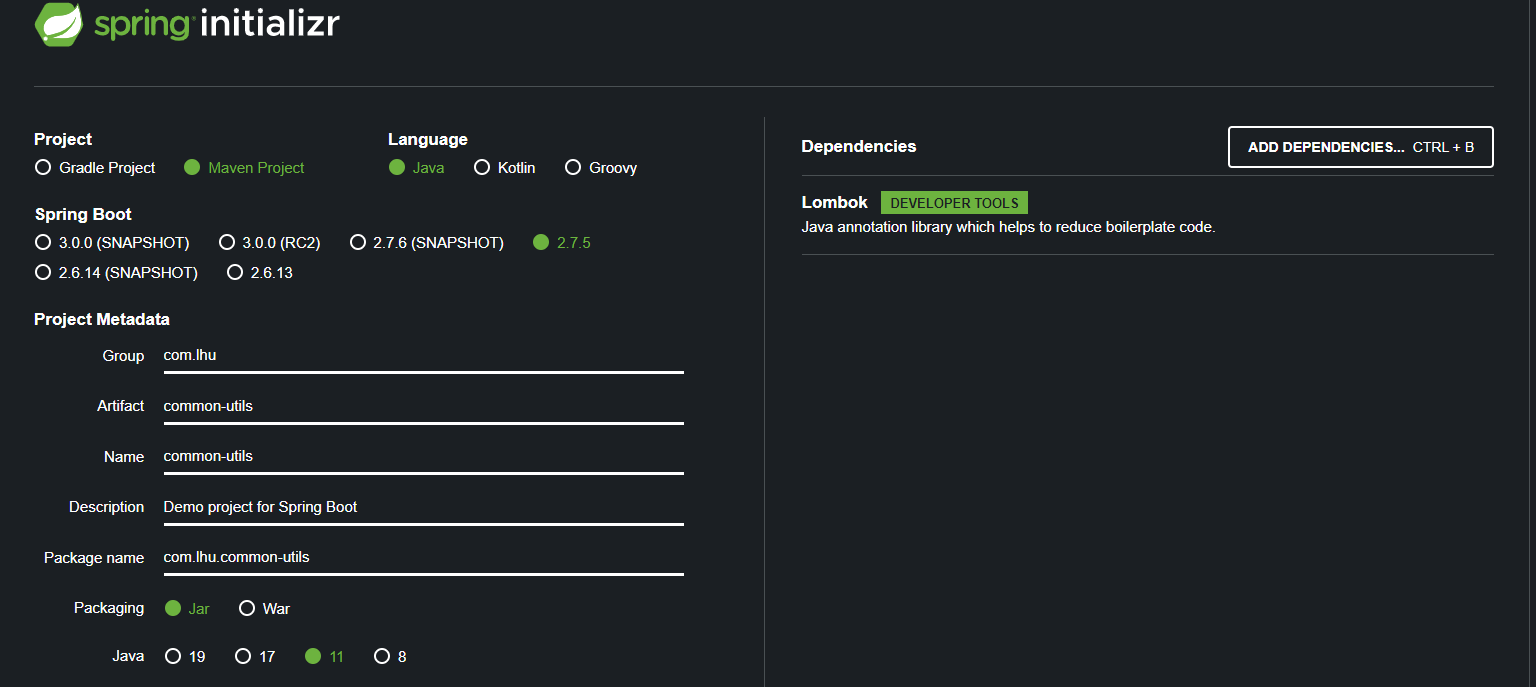
Instead of ablove dependencies…bellow dependencies also needed to work with axon

<!-- https://mvnrepository.com/artifact/org.axonframework/axon-spring-boot-starter -->  
<dependency>  
 <groupId>org.axonframework</groupId>  
 <artifactId>axon-spring-boot-starter</artifactId>  
 <version>4.5.3</version>  
</dependency>  
<!-- https://mvnrepository.com/artifact/com.google.guava/guava -->  
<dependency>  
 <groupId>com.google.guava</groupId>  
 <artifactId>guava</artifactId>  
 <version>31.0.1-jre</version>  
</dependency>

And Comman util can be added like

<dependency>  
 <groupId>com.dailycodebuffer</groupId>  
 <artifactId>CommonService</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
</dependency>

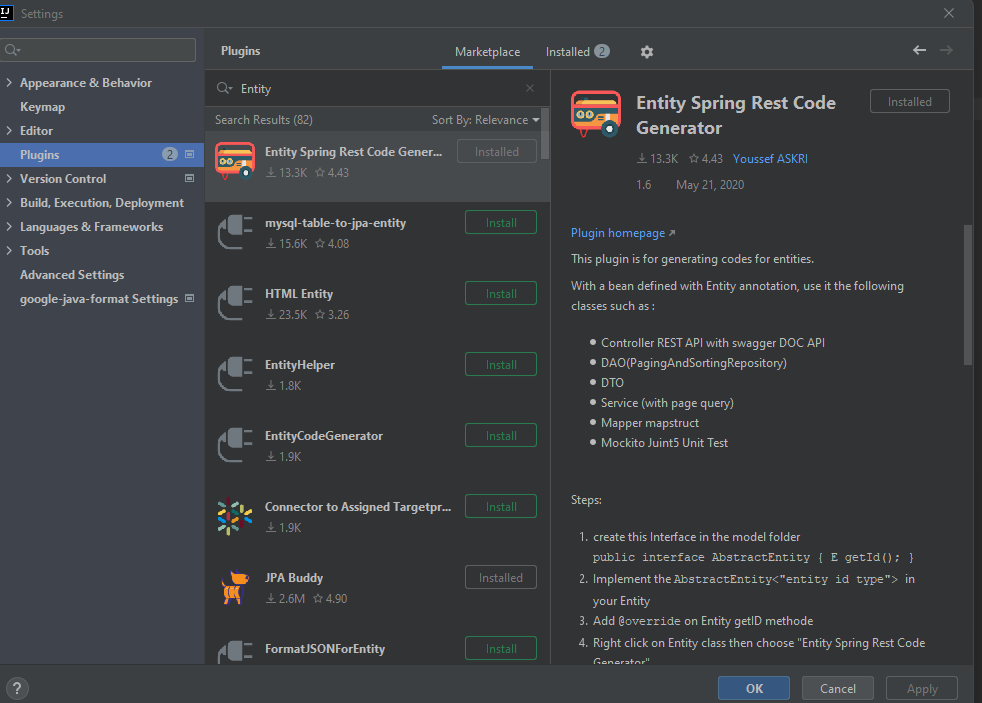
Common-utils

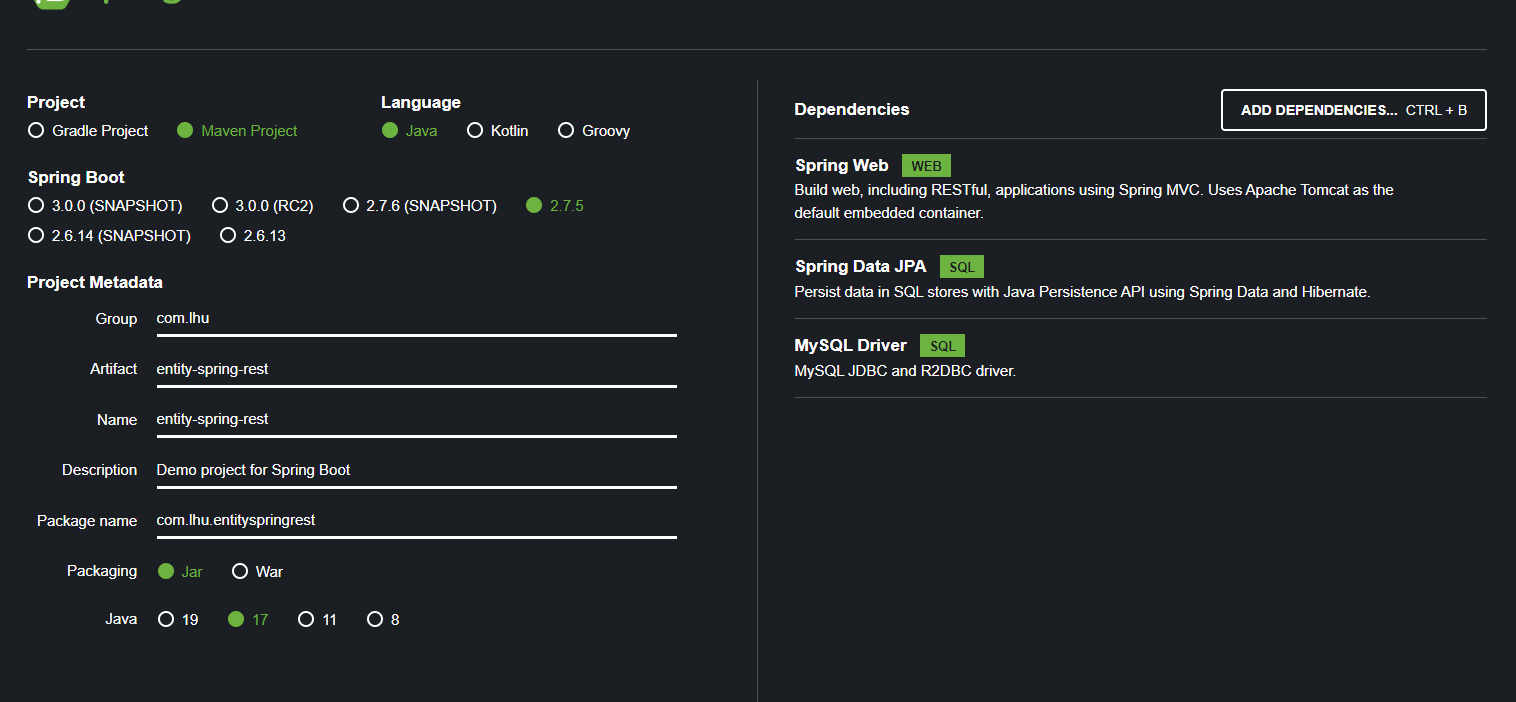


This also needed..

<!-- https://mvnrepository.com/artifact/org.axonframework/axon-spring-boot-starter -->  
<dependency>  
 <groupId>org.axonframework</groupId>  
 <artifactId>axon-spring-boot-starter</artifactId>  
 <version>4.5.3</version>  
</dependency>

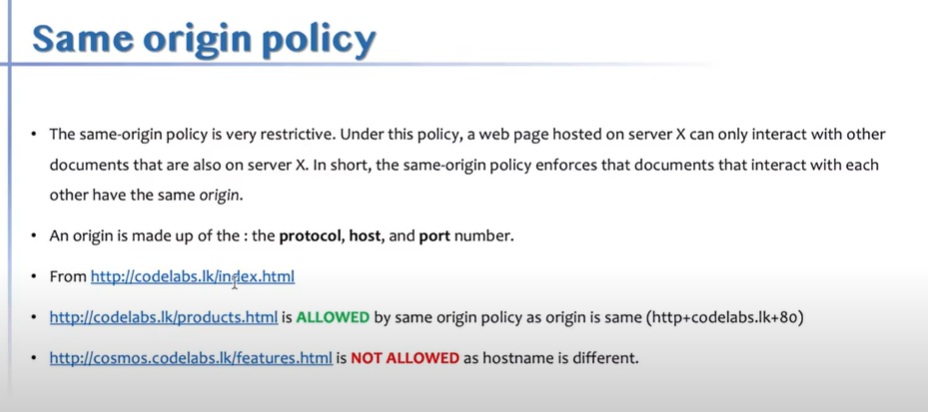
….………….

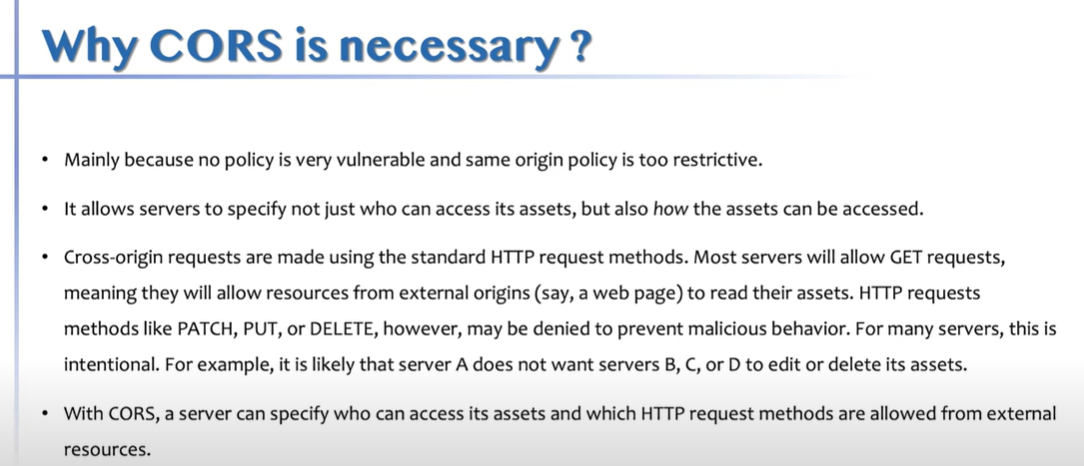


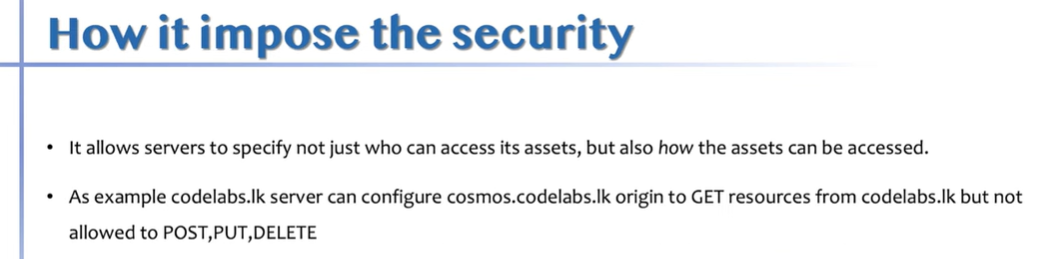


CORS origin problem..Corss origin resource sharing.









Cors validation happen in 3 different ways.

1. coming from real origin which we allowed.
2. Whether you are carrying allowed headers(Authorization header + content type header).
3. Method - GET,POST,PUT…

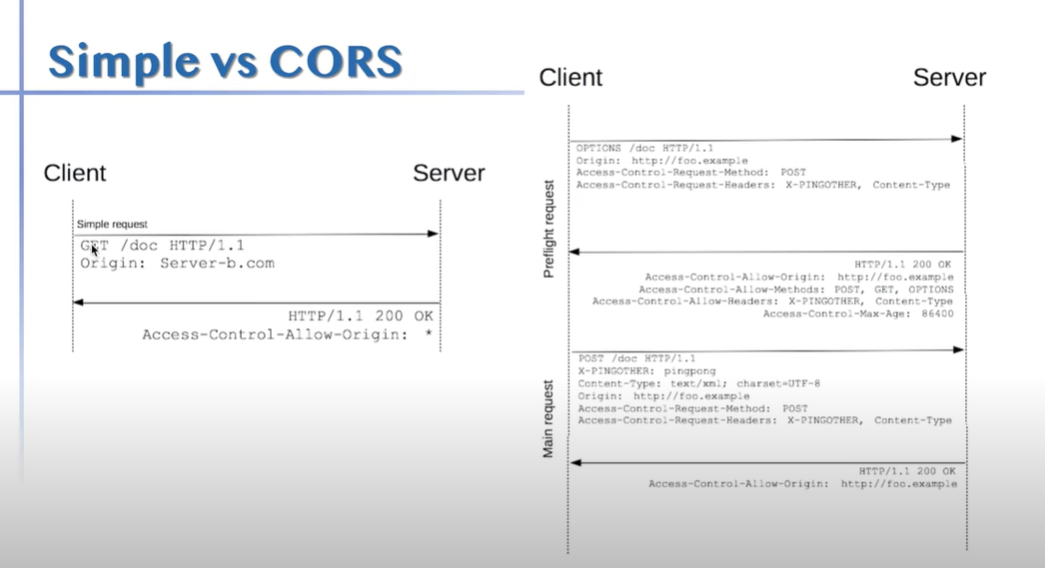
I am coming from this origin,I am carrying this much of messages and can I fly - Pre flight request.

Postman - insominia.

Cors only experience in browser.

Sent pre-flight request.If server is cors enagled, server replies back with I am allowwiing these origin,these headers ,these methods..If server is not cors enagled,then server not respond to pre flight(Optional call) request.then borwser says an error says cors fail.

Cors fail - You are not allowed cors at all OR You are allowed cors at all and send from different domain or different method or allow 2 headers you are trying to send 3 headers.



Browser interceptors to bypass this.

**GraphQL**

**Tutorial -**

**Channel :**

**<https://www.youtube.com/@DanVega>**

**Video:**

**<https://www.youtube.com/watch?v=TVk2fMEezO4&list=RDCMUCc98QQw1D-y38wg6mO3w4MQ&index=3&ab_channel=DanVega>**

New API standard that was invented and open source by Facebook.

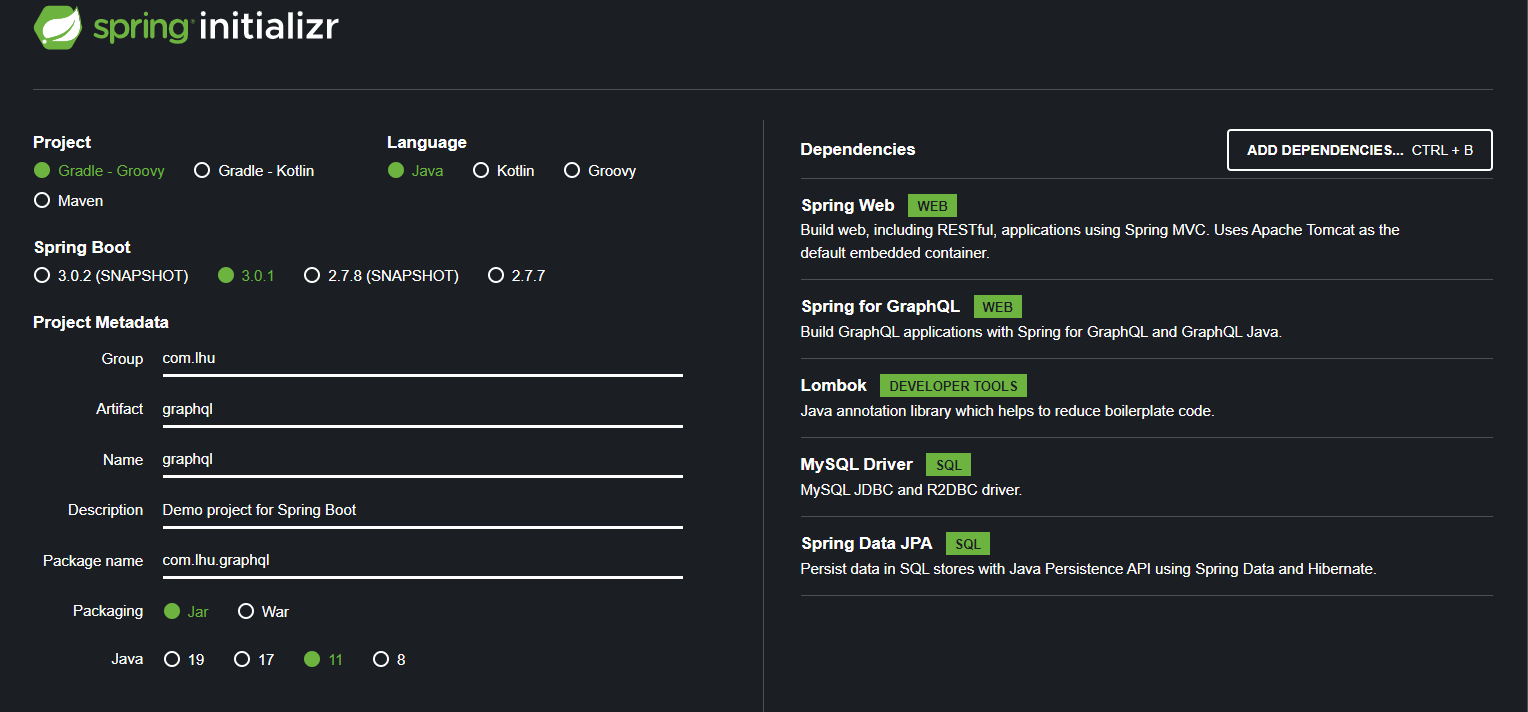
Enable declarative data fetching.

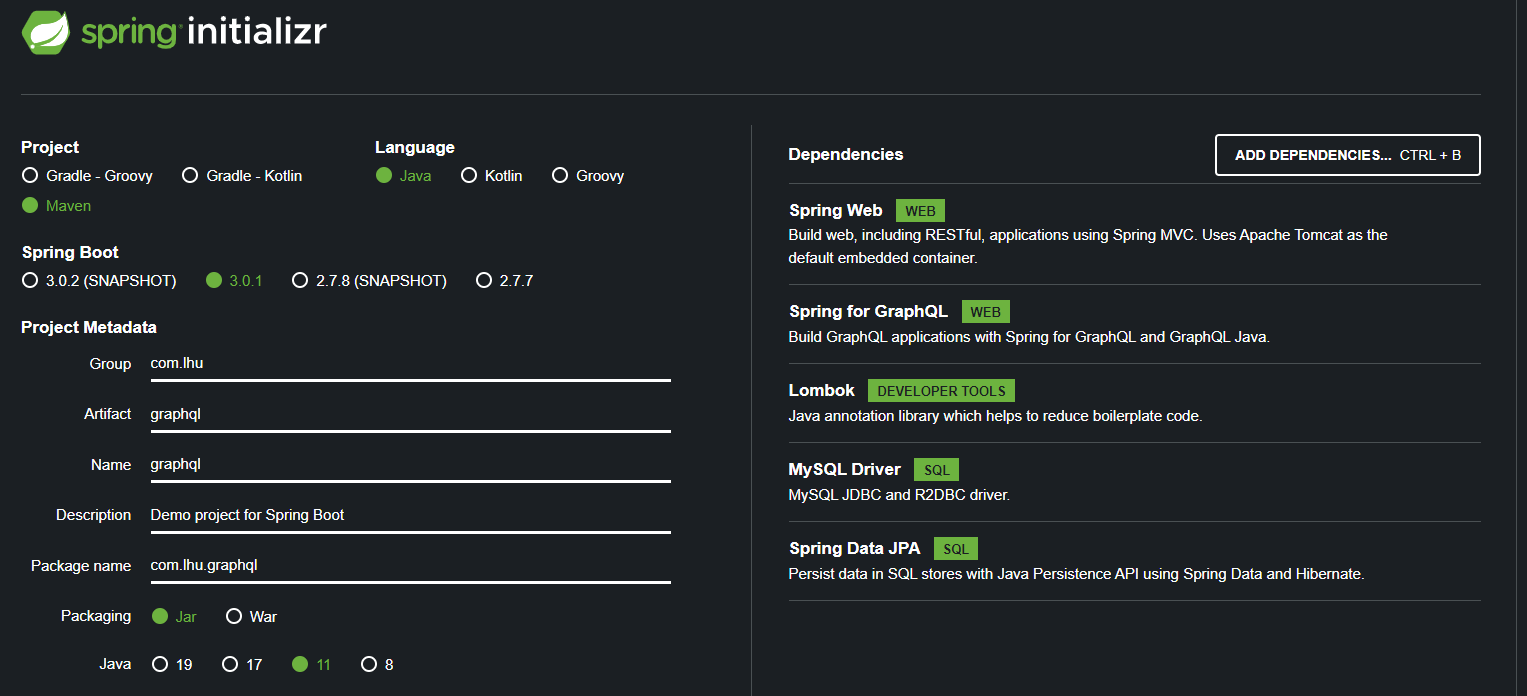
CKient ta puluwan decide kranna mona wage datada API eken fetch karaganan one kiyla..namuth rest walanm API eken dena tika apita gana wenwa.Namuth mekedi client one data tika witarak gannwa.

GraphQL sever exposes single endpoint and responds to queries.

Multiplae EndPoint hadala fixed data structure ekak tiyna data privide karnawa wada eka end point ekak hadala එක multiple consumers ලට ඔහුන්ට අවශ්‍ය data පමණක් access kara ගන්න පුළුවන් විදිහට provide කරන technology එකක් වේ.

https://www.howtographql.com/basics/0-introduction/





Query - to retrieve data.

Mutation - to changed the data (Create,Update,delete)

Subscription - Which allows allows you to create a connection read data and keep that connection for when data changes(15.00).

<http://localhost:2020/graphiql?path=/graphql>

# Welcome to GraphiQL

#

# GraphiQL is an in-browser tool for writing, validating, and

# testing GraphQL queries.

#

# Type queries into this side of the screen, and you will see intelligent

# typeaheads aware of the current GraphQL type schema and live syntax and

# validation errors highlighted within the text.

#

# GraphQL queries typically start with a "{" character. Lines that start

# with a # are ignored.

#

# An example GraphQL query might look like:

#

#     {

#       field(arg: "value") {

#         subField

#       }

#     }

#

# Keyboard shortcuts:

#

#   Prettify query:  Shift-Ctrl-P (or press the prettify button)

#

#  Merge fragments:  Shift-Ctrl-M (or press the merge button)

#

#        Run Query:  Ctrl-Enter (or press the play button)

#

#    Auto Complete:  Ctrl-Space (or just start typing)

#

query{

  allStudent{

    id

    firstName

    lastName

    age

    studentClass{

      id

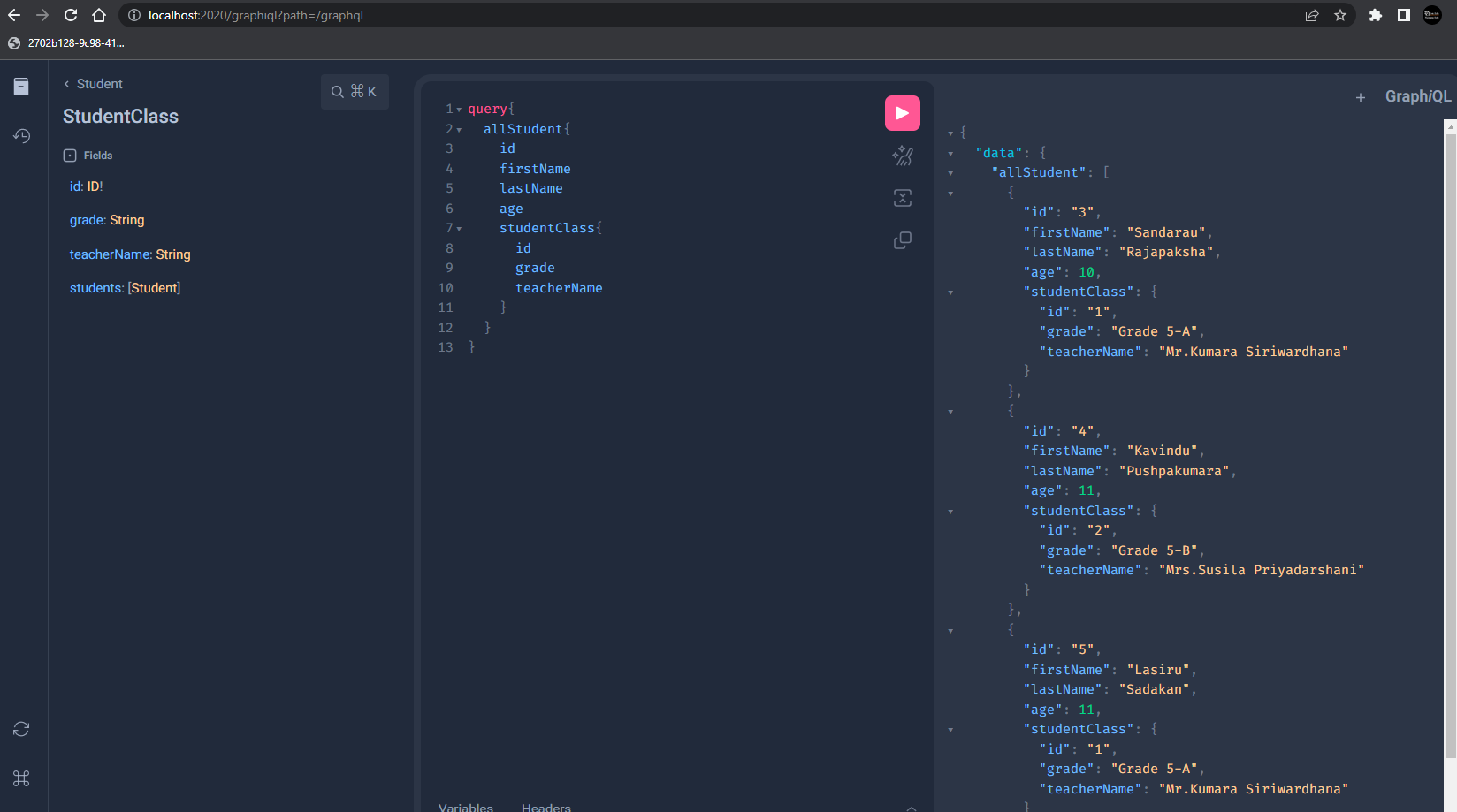
      grade

      teacherName

    }

  }

}



query{

  findStudent(id:4){

    id

    firstName

    lastName

    age

    studentClass{

      id

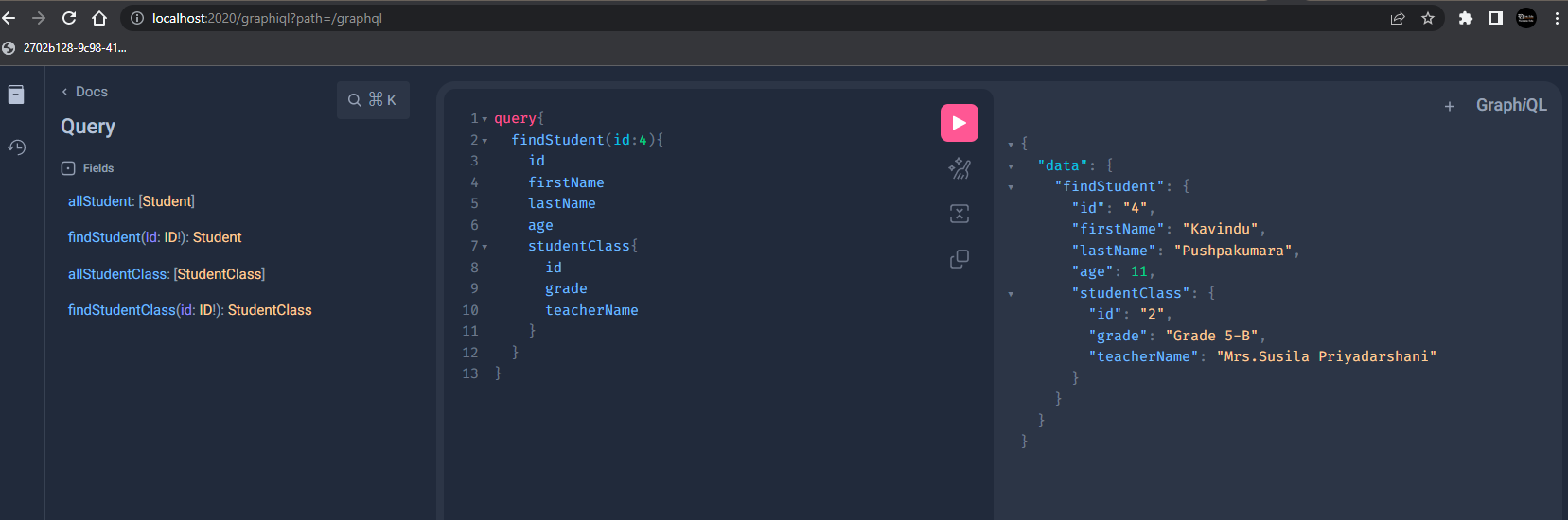
      grade

      teacherName

    }

  }

}



query{

  allStudentClass{

    id

    grade

    teacherName

    students{

      id

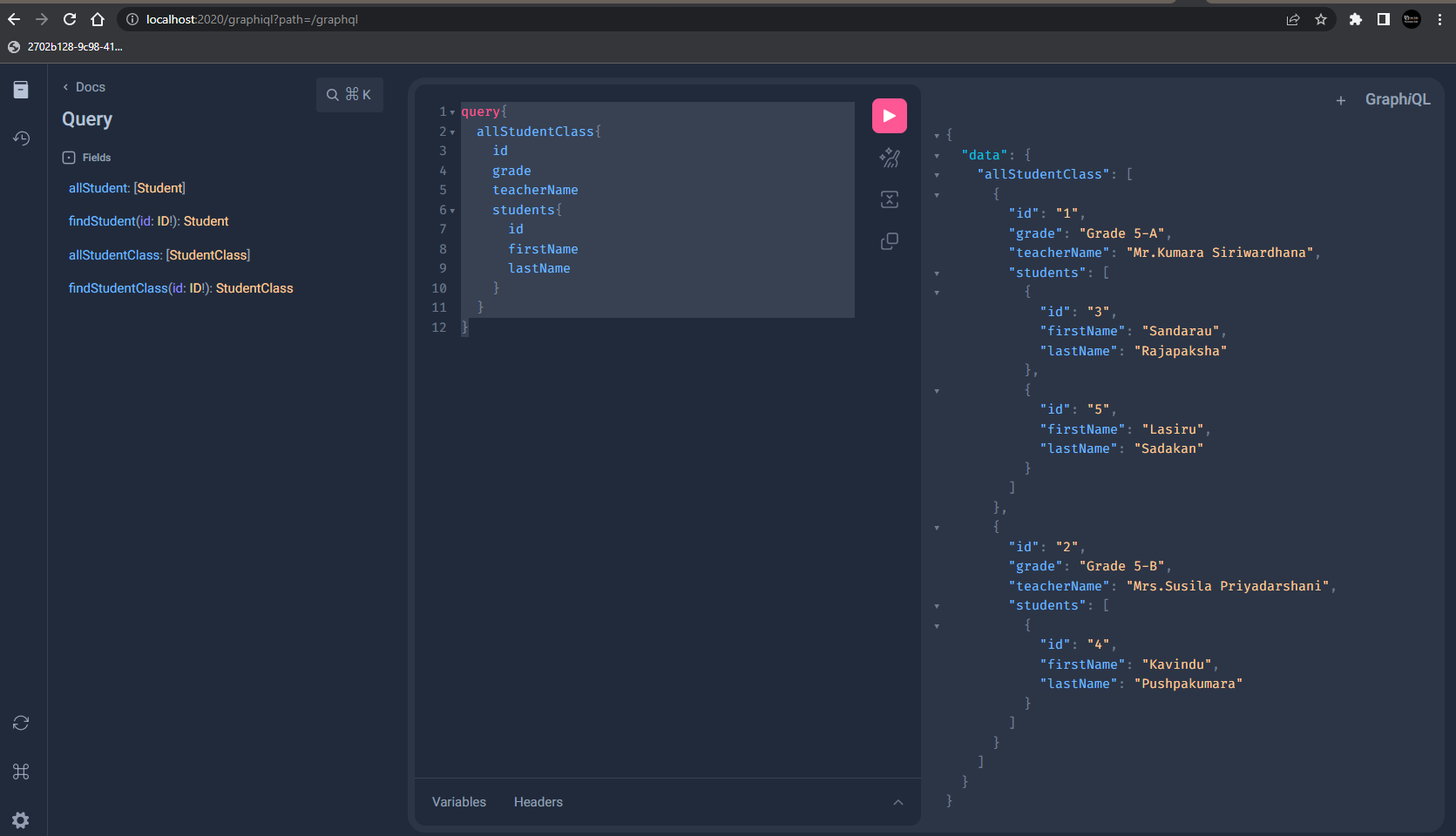
      firstName

      lastName

    }

  }

}



query{

  findStudentClass(id:2){

    id

    grade

    teacherName

    students{

      id

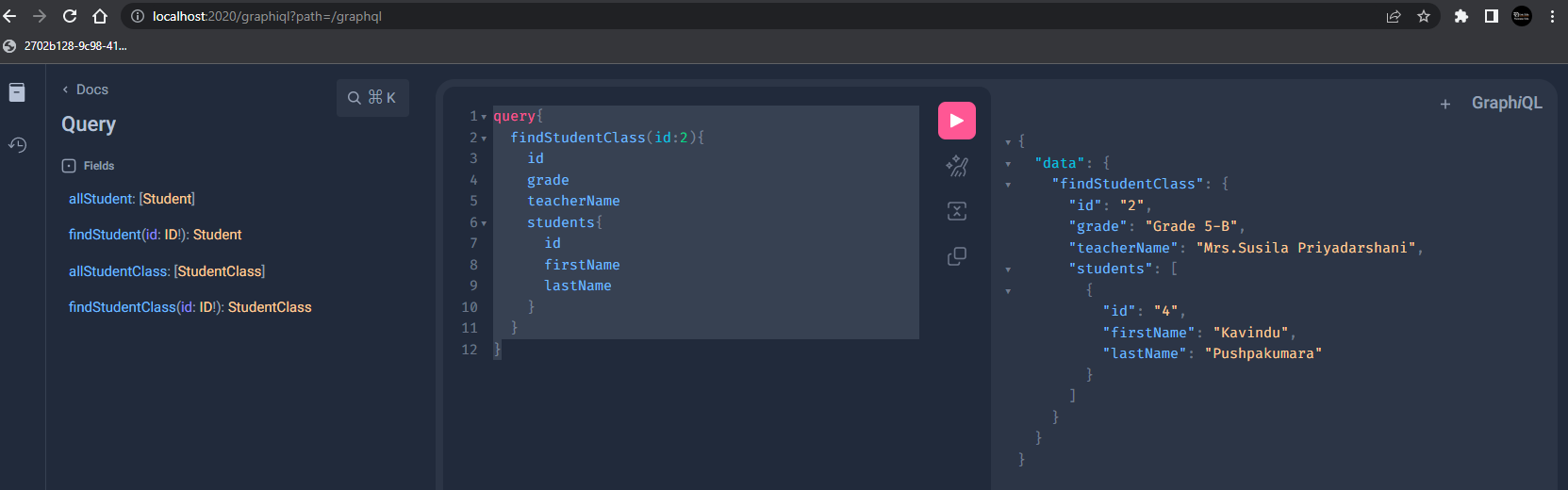
      firstName

      lastName

    }

  }

}



mutation{

  createStudentClass(grade:"Grade 6-A",teacherName:"Mr.fernando"){

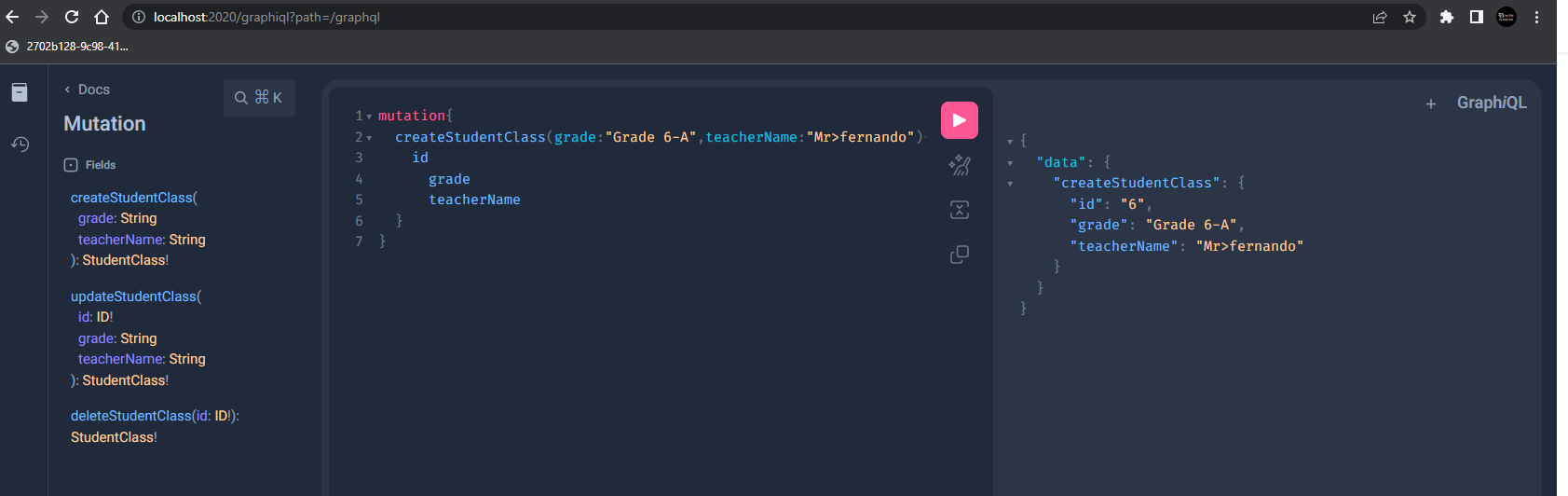
    id

      grade

      teacherName

  }

}



mutation{

  updateStudentClass(id:6,grade:"Grade 6-A",teacherName:"Mr.fernando"){

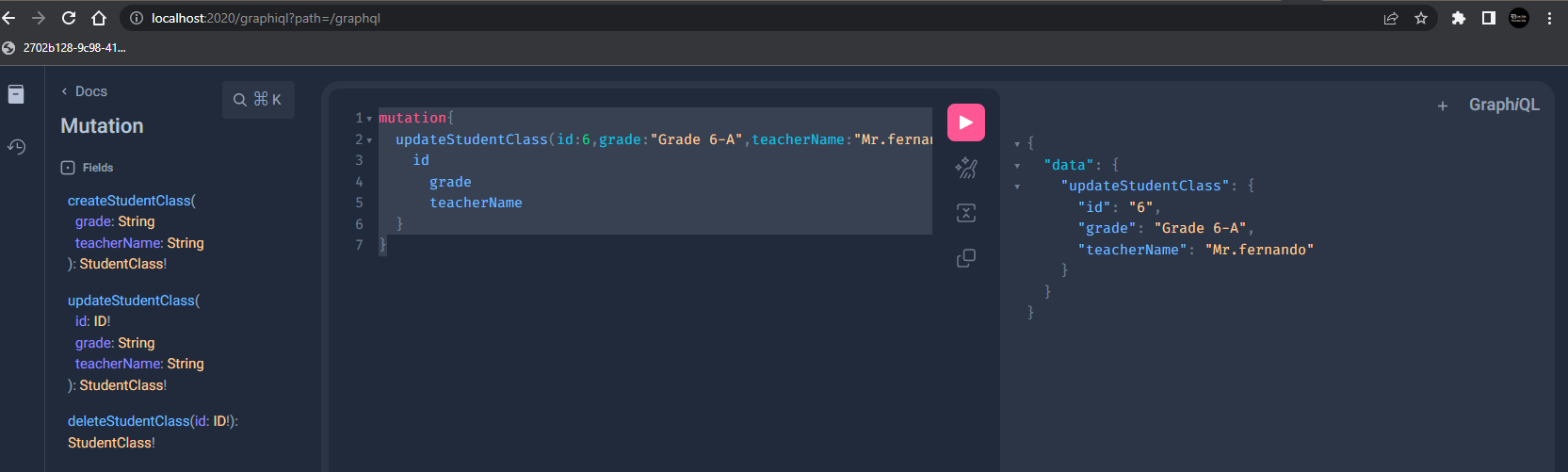
    id

      grade

      teacherName

  }

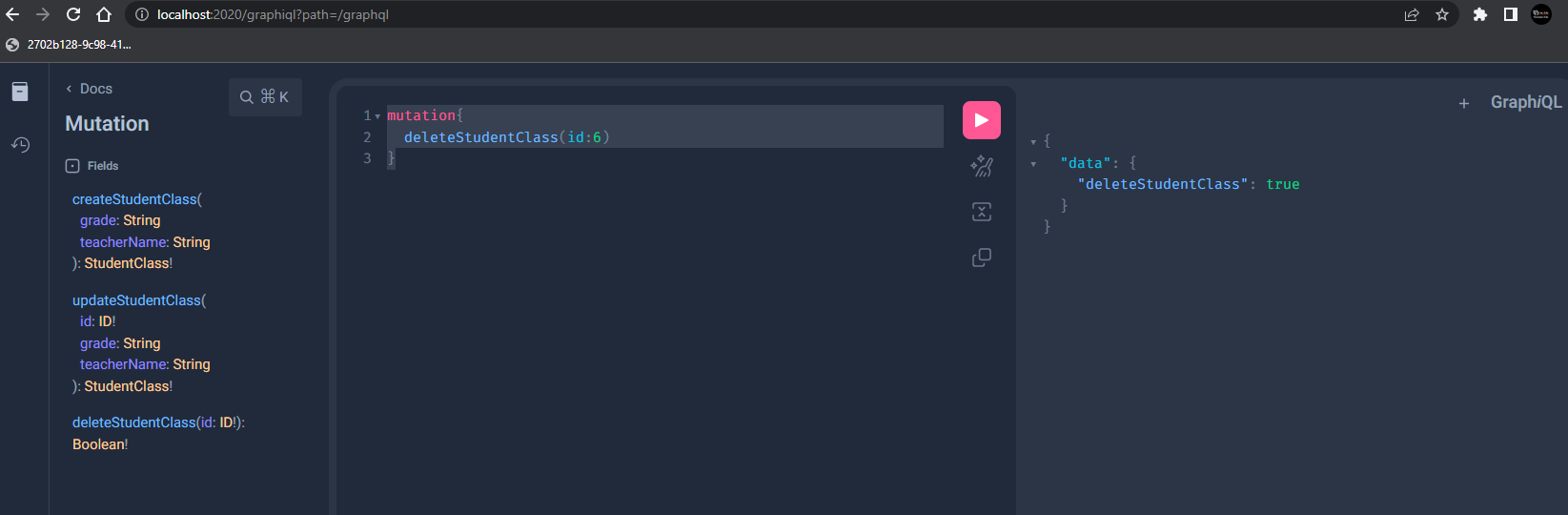
}



mutation{

  deleteStudentClass(id:6)

}



query{

  allStudentClassPage(page:0,size:2){

     id

    grade

    teacherName

    students{

      id

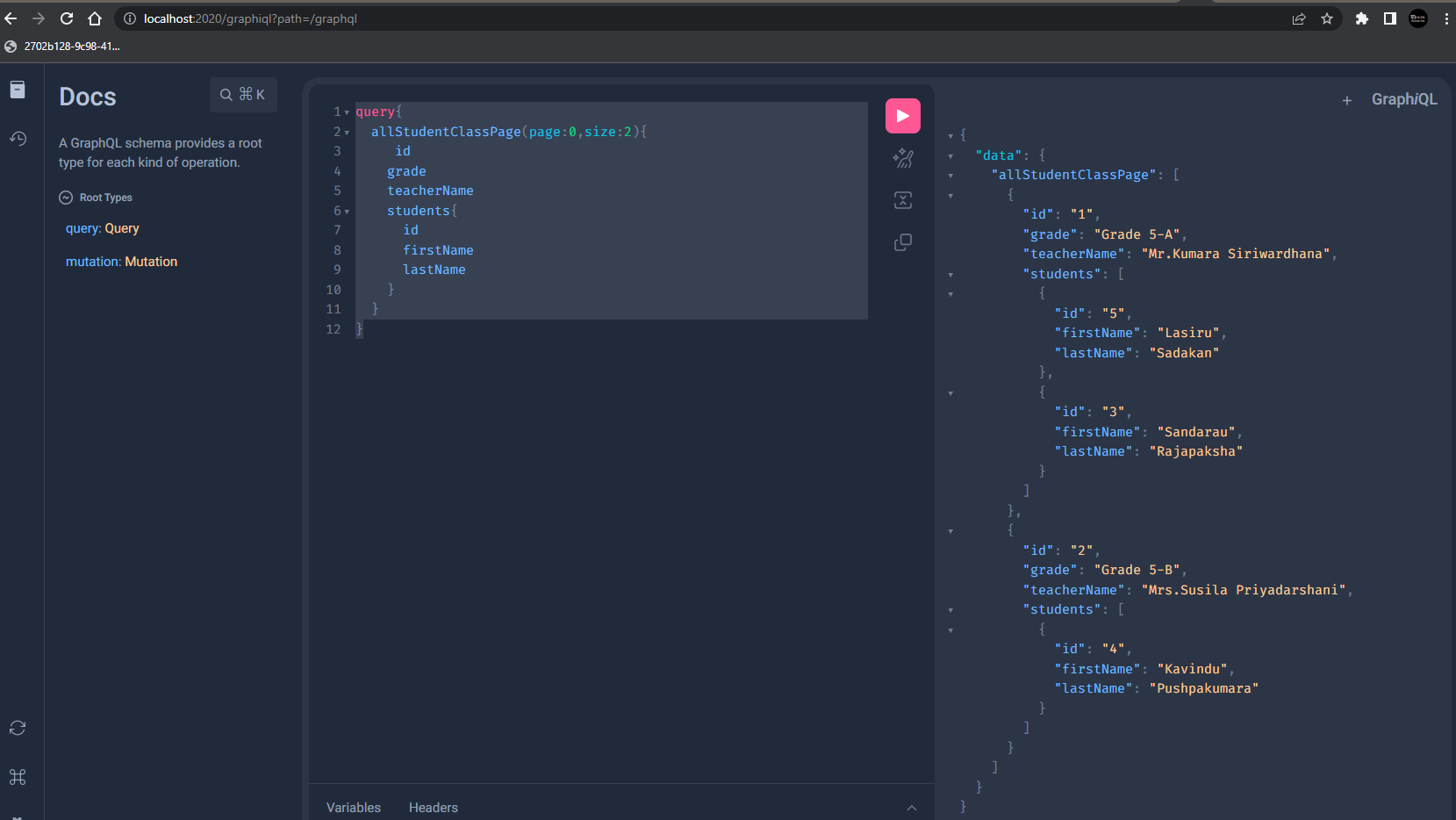
      firstName

      lastName

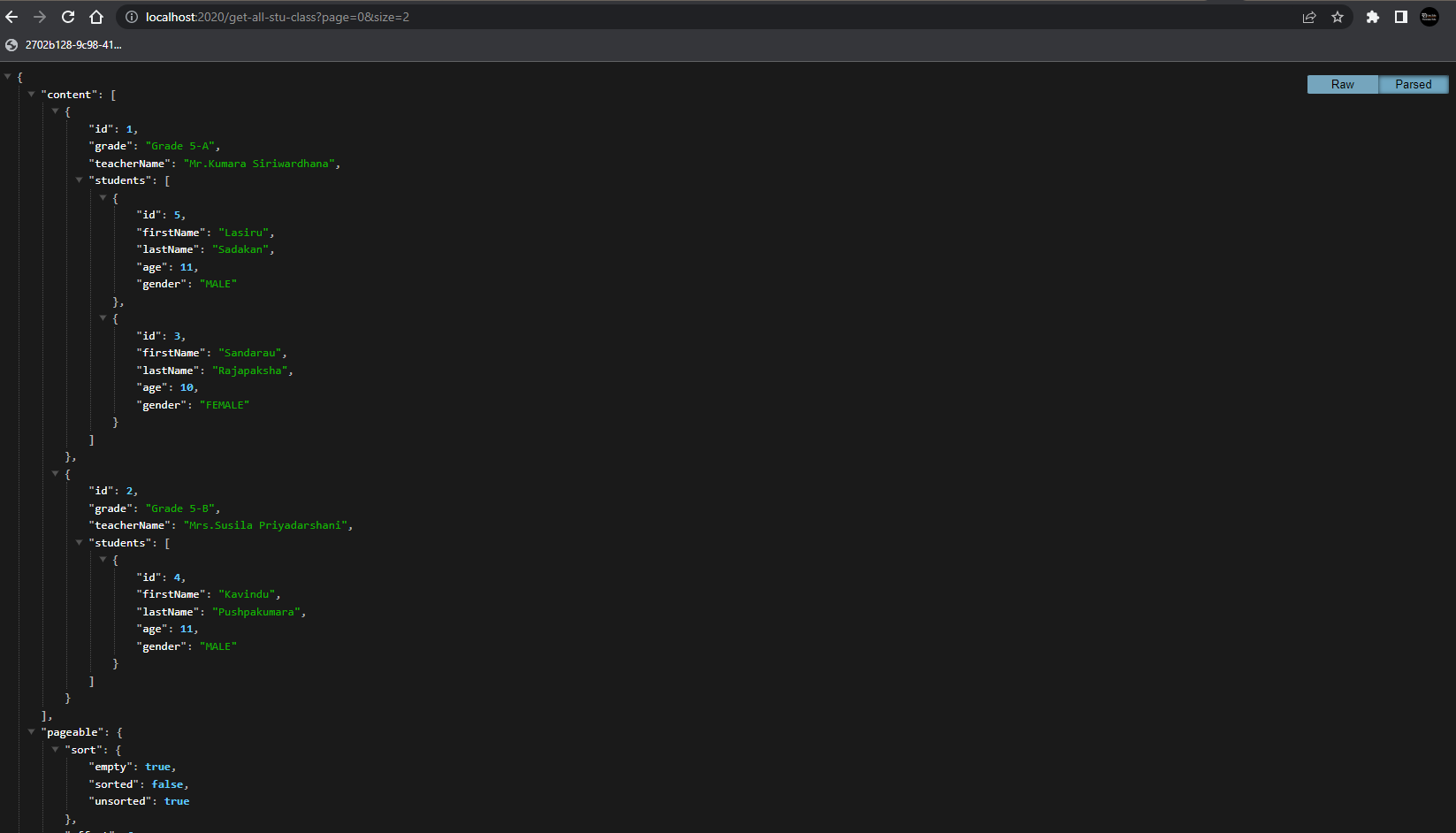
    }

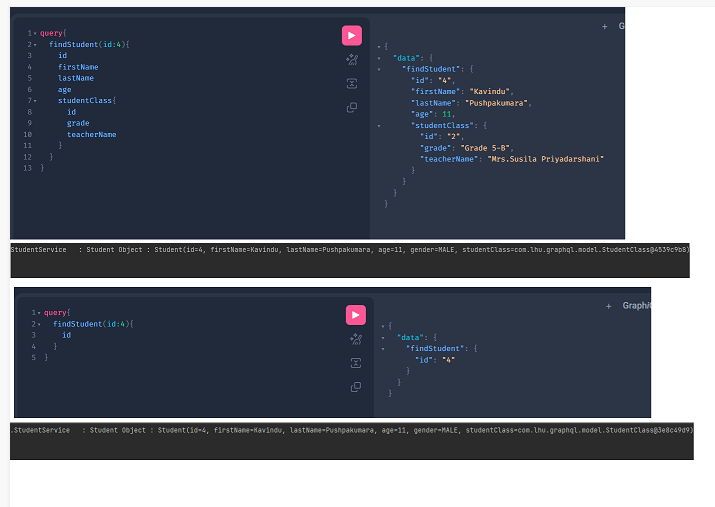
  }

}

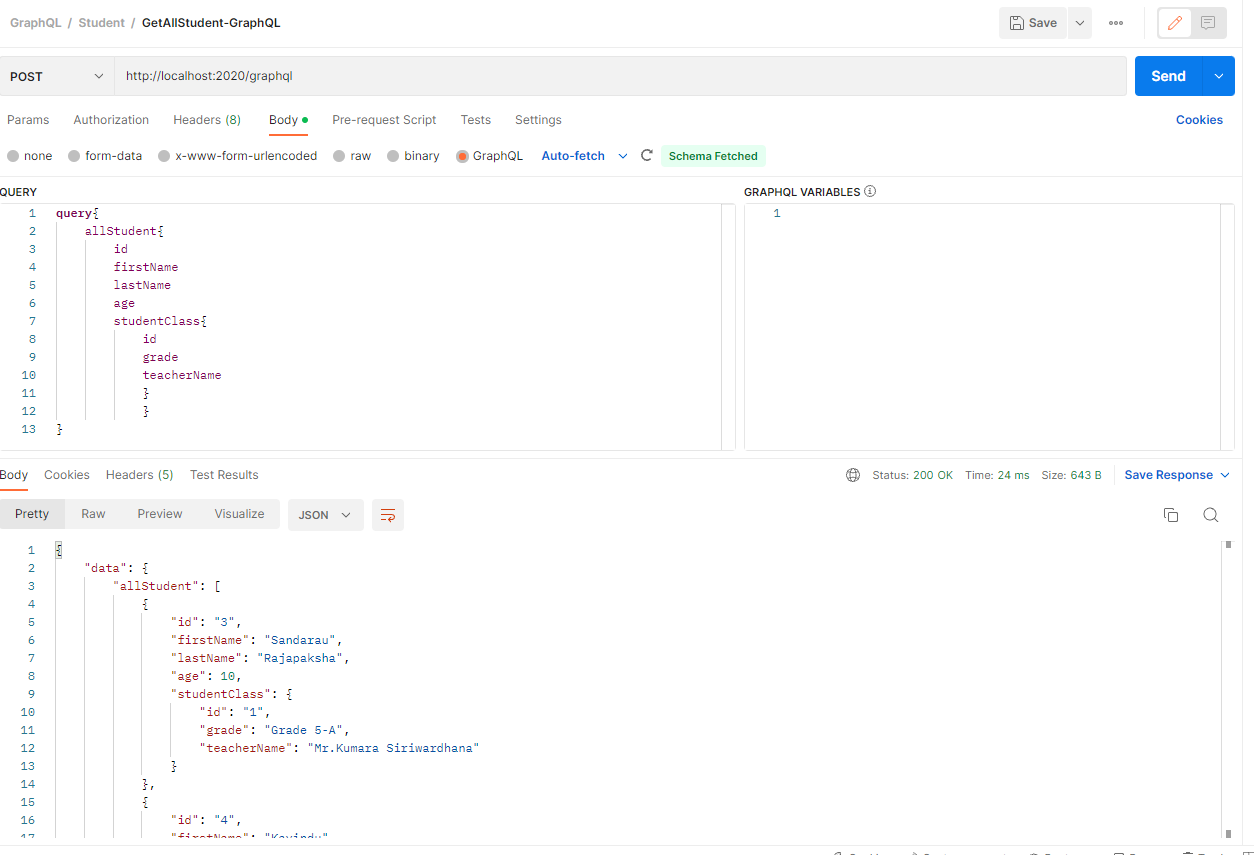


<http://localhost:2020/get-all-stu-class?page=0&size=2>





Test using Postman



RSET - Asynchromous

SOAP - Synchromase

Web Socket - There is an dedicated tunnal is always open with client and server.Bi direactional msg can be passed.

Pubs - Subs

AWS - SQS

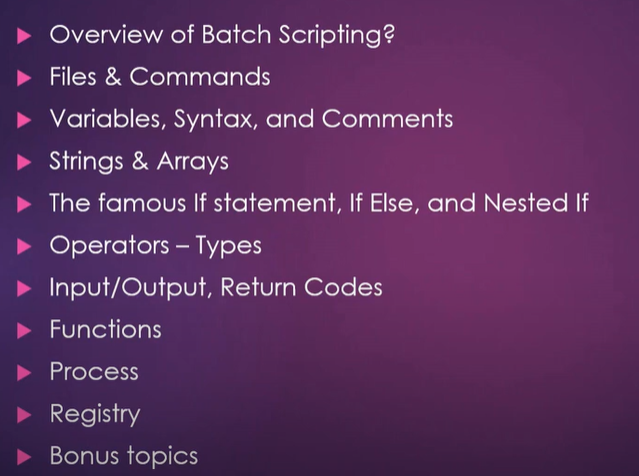
Kafka,RabbitMO

Redis - Event trigger haraha messaging system wenuwata redis use kala haka.

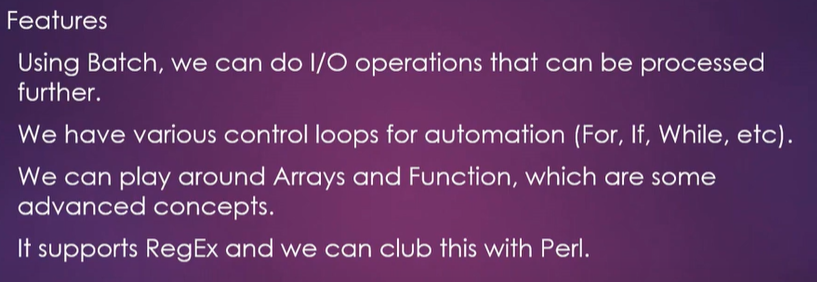
Sockect IO,Socket CLustoer,AWS api gatway already suport for scket APIs

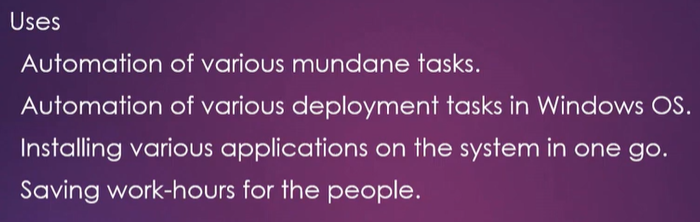
Java17 - record, sealed ,swtich

Batch Programming









HTTPS

Different teams which we can to secure a web service.

SSL- secure socket layer

TSL- Trabsport secure layer(More features added than ssl,Lot of certificates are created using TLS)

TrustStore and KeyStore

TrustStore - Certificate related to public information.

KeyStore- Certificate related to private information.