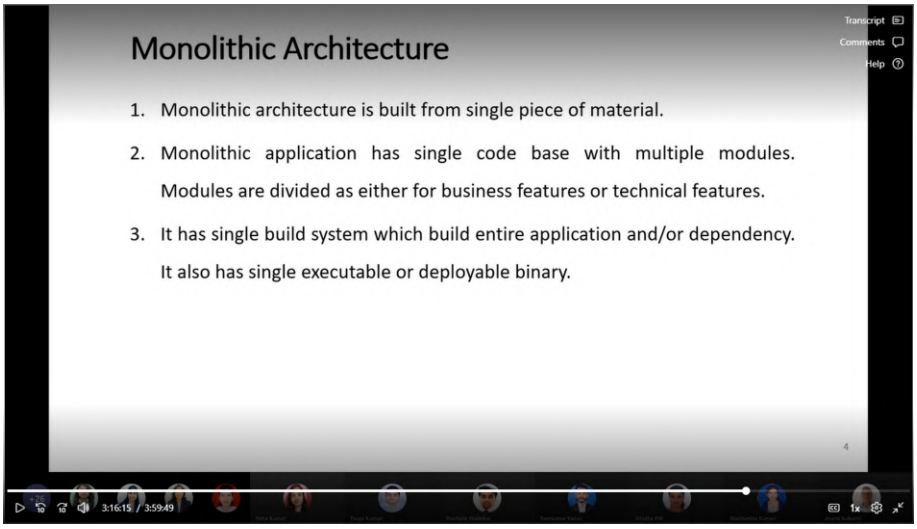
Microservices With Spring Boot

Wednesday, May 25, 2022 5:36 PM

The industry started using micro-services architecture nowadays. But before micro-services architecture industry were using service oriented architecture and before SOA industry were using Monolithic Architecture.

Monolithic Architecture :- All the application module or business logic modules treated as a single module suppose in Net Banking we also have a lot of modules like Login, Logout, profile, transaction statements, creating new account etc. if we club all modules into single code based is called Monolithic Architecture. It means if we building a complex application ultimately entire application we are going to build together into single development file. But tomorrow we are trying to make any changes into any particularly module then we have to do changes and build all the modules again then we have to build a single executable deployable file. We will get only single executable file.



Limitations Of Monolithic Architecture :::

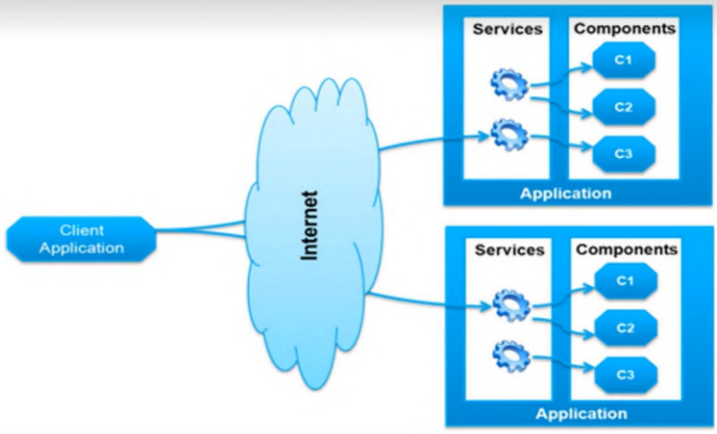
Monolithic Architecture is like a big container, wherein all the software components of an app are assembled and tightly coupled, i.e., each component fully depends on each other.

- For large or complex applications it is very difficult to maintenance because they are dependent on each other.
- It will slow down development process because for changes we have to redeploy are whole application instead of updating a part where we did some changes. So it takes more time or slow development.
- If one service or module goes down then it affects all the services or modules because all modules are depend upon each other.

This one used only or good for late 19th century but after 2000 industries started using SOA means service oriented architecture.

Service Oriented Architecture is becomes very popular in 2005. Every Industry Started using service oriented Architecture. SOA says I will create a lot of services For example:- for file transfers, user management etc. it means for per feature I am opening a new services and services can be called by a client but in services we do not write any code. Our business logic of particular services written inside components it means when a client call any services then services internally calls components where we written our business logic. In general, we expose services to client and services do not have a business logic the business logic is written inside in components.

Service Oriented Architecture (SOA)

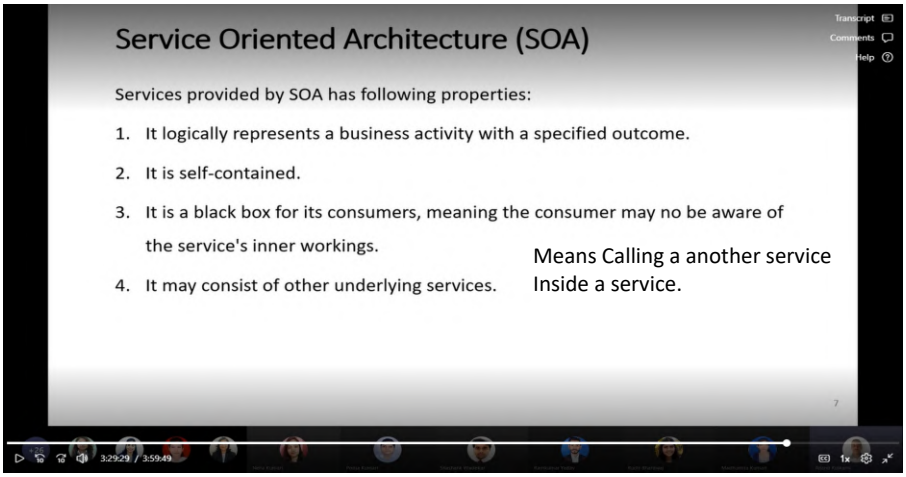


5

Service Oriented Architecture (SOA)

- SOA is a style of software design where services are provided to the other components by application components, through a communication protocol over a network.
- A SOA service is a discrete unit of functionality that can be accessed remotely and acted upon and updated independently.

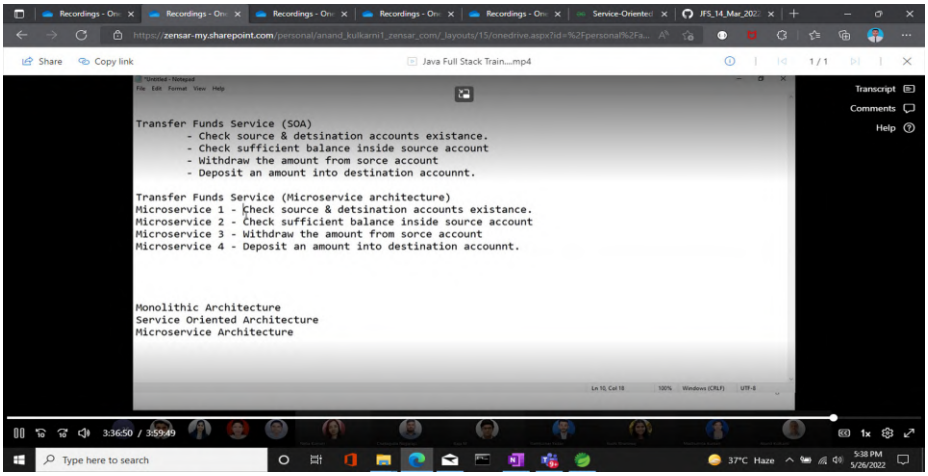
6



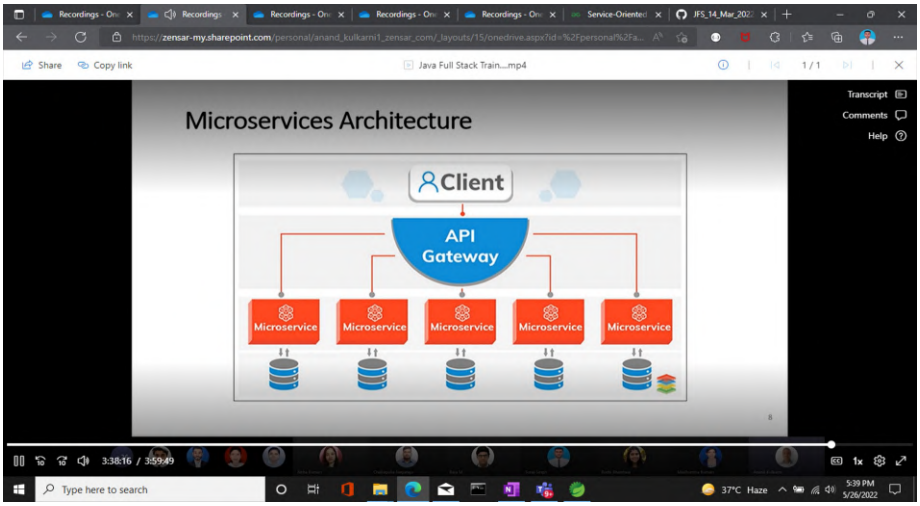
Limitations

- A huge initial investment is required for SOA
- When services interact they exchange messages to tasks. the number of messages may go in millions. It becomes a cumbersome task to handle a large number of messages.

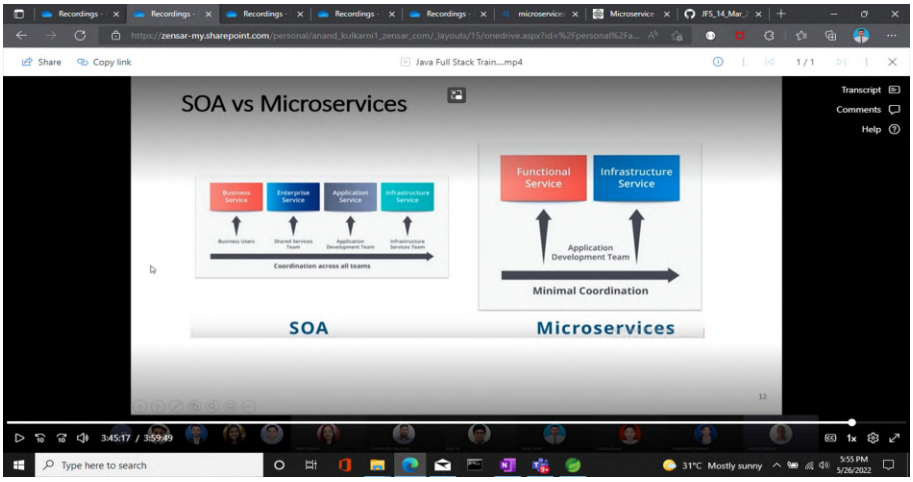
Microservices Architecture itself means a services very small in size. For every task we are creating microservices. It means microservices is going to be a task based application. For every task we are going to create separate microservice.



Client Will not Call directly Microservices. Client will call API Gateway and API Gateway will call Microservices might be possible every task based services (Microservice) using different Databases.



The monolithic is a single unit of Modules, dividing single unit or services based upon business logic is called SOA and SOA divided into further specific task is called Microservices.

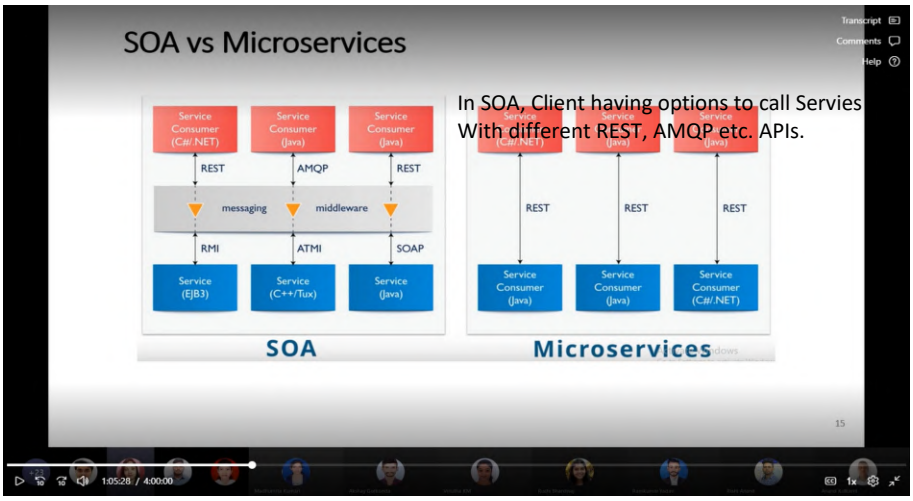


We have four types of services in SOA.

Business Service is a service which is exposed to client and client is always going to access business services in SOA. Then business service internally calls enterprises services but we do not write our business logic inside enterprises services we write our business logic inside application services. So enterprises services calls suitable application services and get the work done. But even for the application writing we have infrastructure services dependencies. Infrastructure services for providing messages services, email services, logging services etc. this is non logical or non technical service. For example showing some messages on services when we want to show any message.

We have 2 types of Services in Microservices

Functional Services is a services where we write our main business logic actual task specific business logic or we can say it is combination of business , enterprises, application services. And it calls Infrastructure services when required.

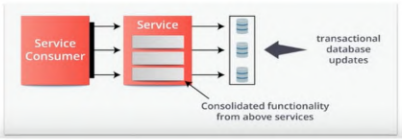


When a inter-communications microservices calls happening is called Remote Call. Here going to happen remote calls or inter-communications calls. When many remote calls happening between the microservices to full fill a Client requirement. Then client response time will increase. But due to best infrastructures support the performance is not sacrifice. That's why microservices heavily used nowadays.

But in SOA, when a client calls any service internally it calls enterprise service and enterprise service will call application service. And there is not happening any remote call due to this client response time is not increased.

Service Granularity - SOA vs Microservices

Microservices



SOA

communication call it's a remote call.

16

What is Microservice?

Microservice is an architectural style that structures an application as a collection of services that are:

1) Highly maintainable and testable
2) Loosely coupled
3) Independently deployable
4) Organized around business capabilities
5) Owned by a small team

The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications.

Seat microservice 8 is calling microservice B.

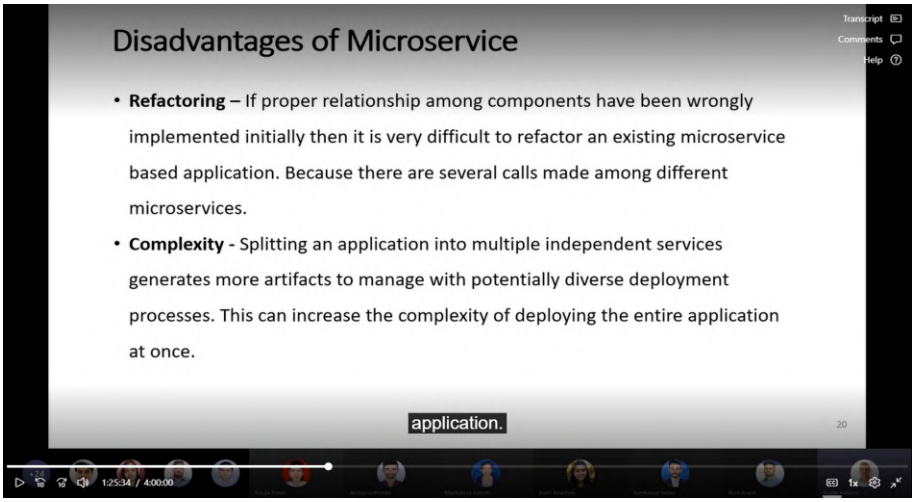
18

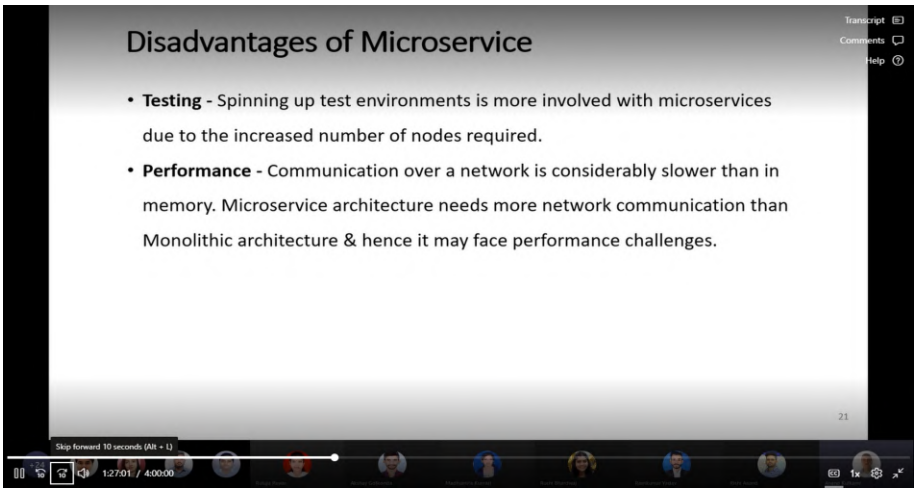
Advantages of Microservices

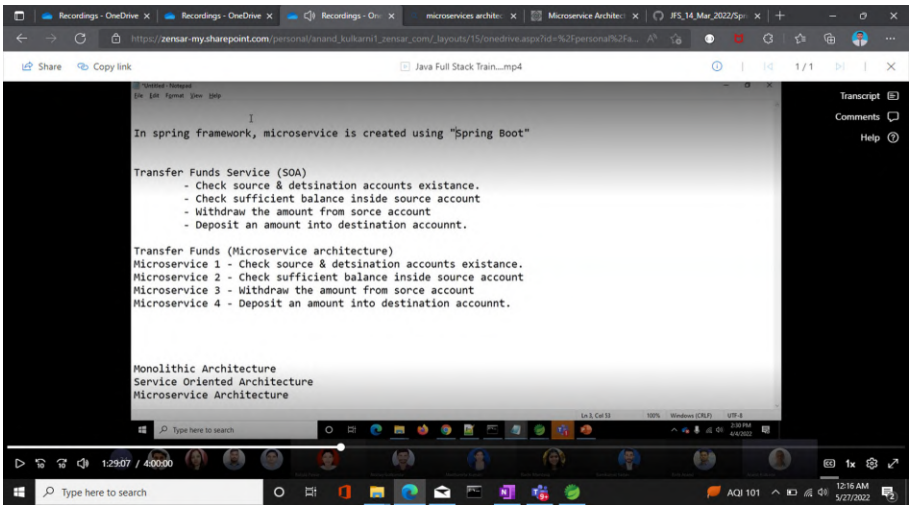
- **Simple To Deploy** - Deploy in pieces without affecting other services.
- **Simpler To Understand** - Follow code easier since the function is isolated and less dependent.
- **Reusability Across Business** - Share small services like payment or login systems across the business.
- **Faster Defect Isolation** - When a test fails or service goes down, isolate it quickly with microservices.
- **Minimized Risk Of Change** - Avoid locking in technologies or languages - change on the fly without risk.

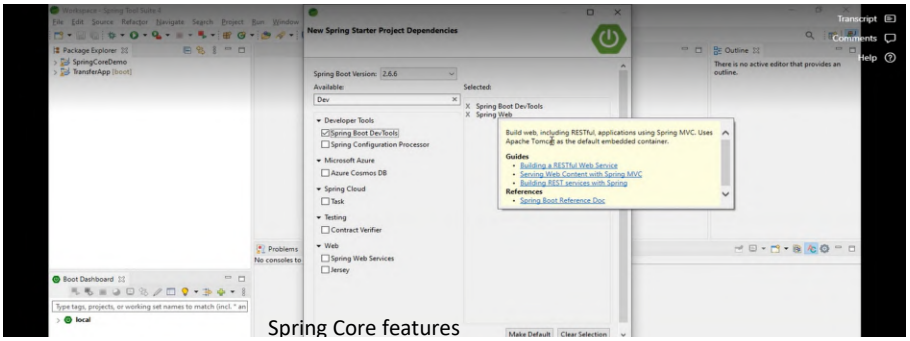
So these are the some of the important point about advantages

19

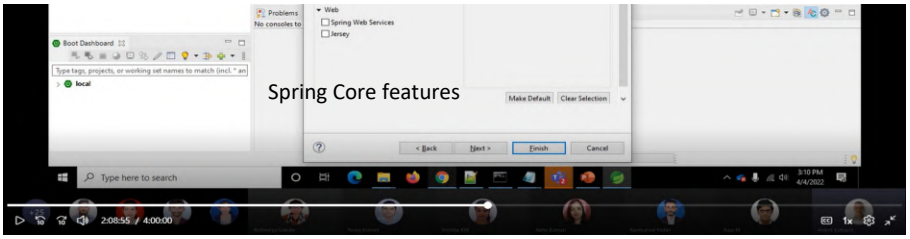


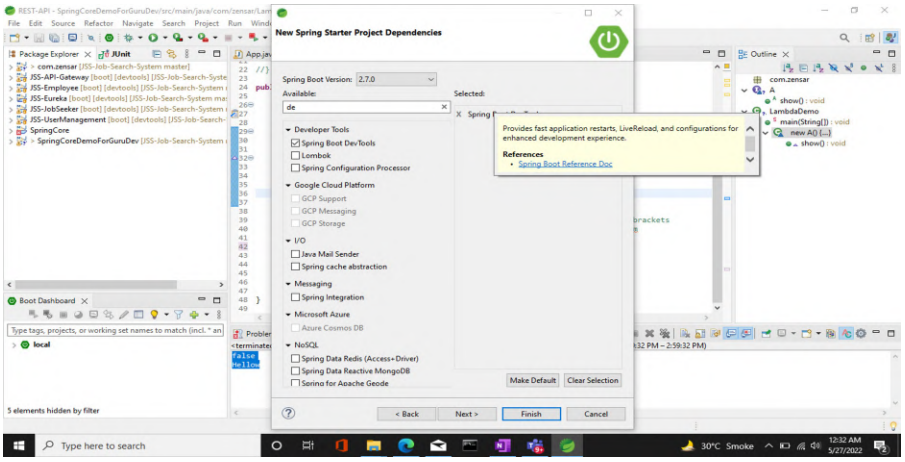




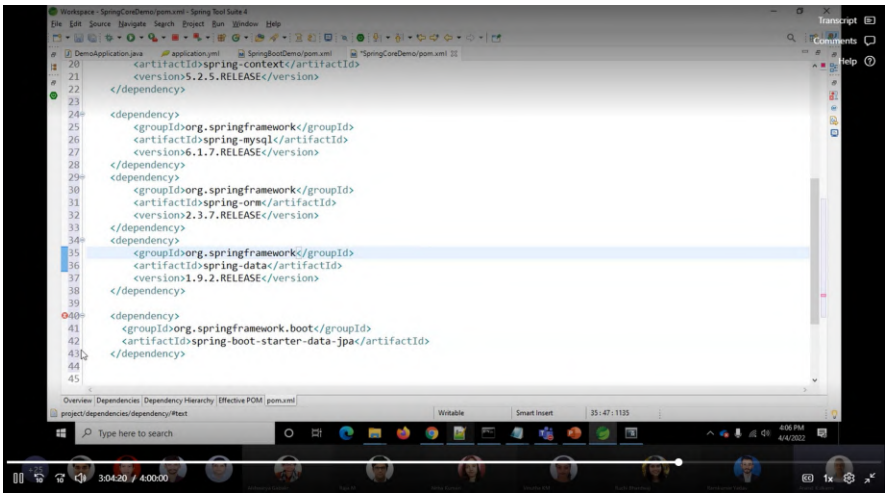


Spring Core features



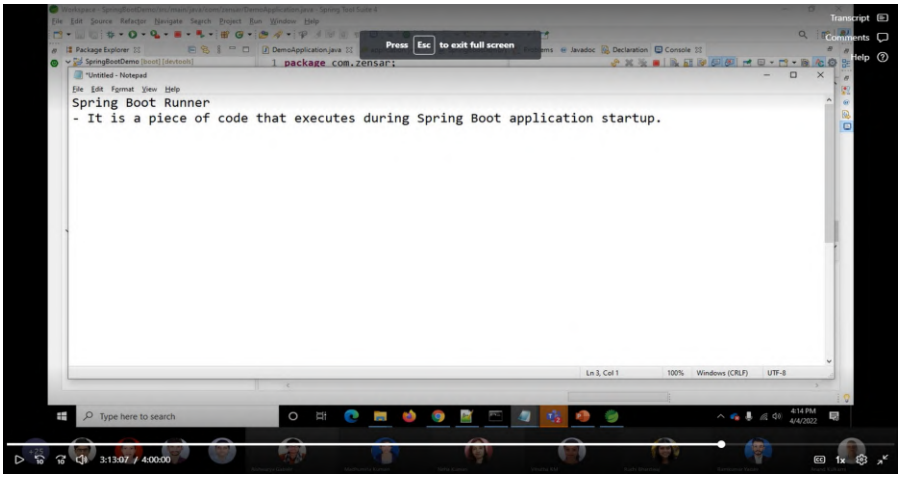


Common application properties of spring boot : [Common Application Properties (spring.io)]



Spring-boot-starter dependency is a collection of small-small dependencies. The spring boot provides Spring-boot-starter dependency concept helps to speed up development process.In Spring Core, We have to add 3 dependencies for database connection like ORM, mysql, data etc. but with the help of Spring boot we have to add only one dependency like spring-boot-starter-data-jpa instead of adding 3 dependencies.

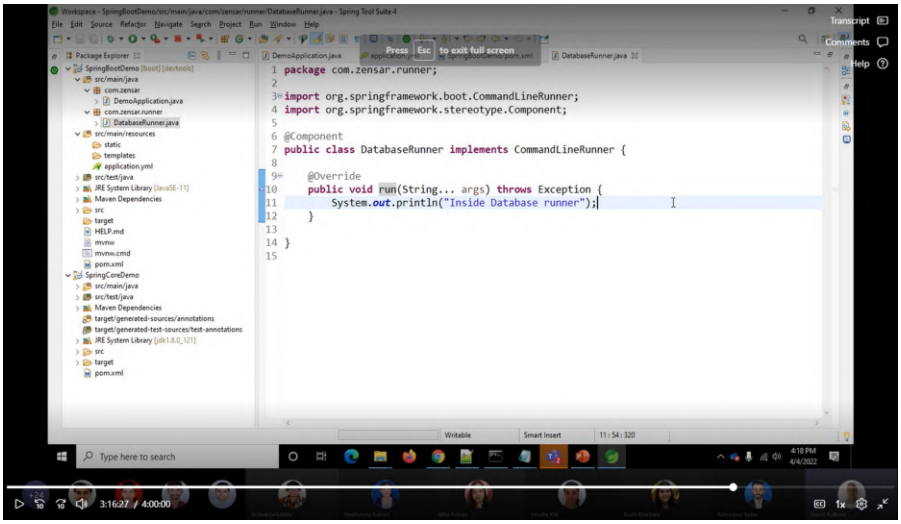
Spring boot runner



Spring Boot provides two interfaces, CommandLineRunner and ApplicationRunner, to run specific pieces of code when an application is fully started. These interfaces get called just before run() once SpringApplication completes.

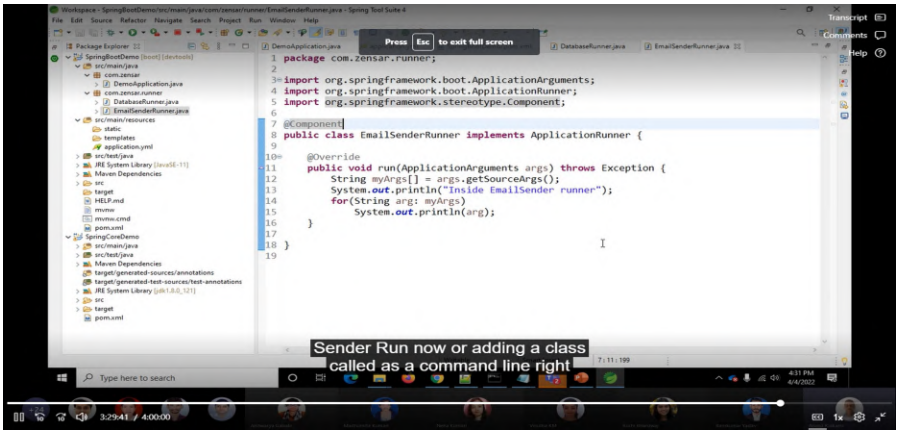
Args is act as a program arguments means we can pass arguments using CMD In STS. The give args by -> Run as -> Run Configuration -> Choose Spring boot main application an pass Arguments In Program arguments.

This interface provides access to application arguments as string array. Let's see the example code for more clarity.



ApplicationRunner wraps the raw application arguments and exposes the ApplicationArguments interface, which has many convinent methods to get arguments, like getOptionNames() to return all the arguments' names, getOptionValues() to return the argument value, and raw source arguments with method getSourceArgs().

@Component if we do not use this annotation then spring boot will not scan your class, it will ignore it. Just like in spring, creating our classes and it will not create objects until we are not registering them into beans.xml files similarly also by writing a class Email Sender Runner or Database runner as a command line runner it will not make any sense until we did not tell spring boot that hey spring boot please read this class. We can specify this by adding @Component annotation.

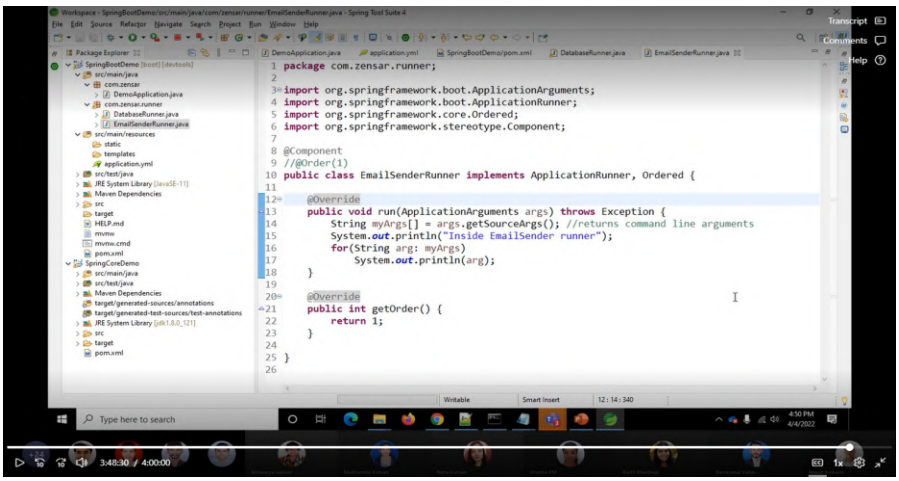


Command Line Runner is a Old runner and Application Runner is a new Runner.

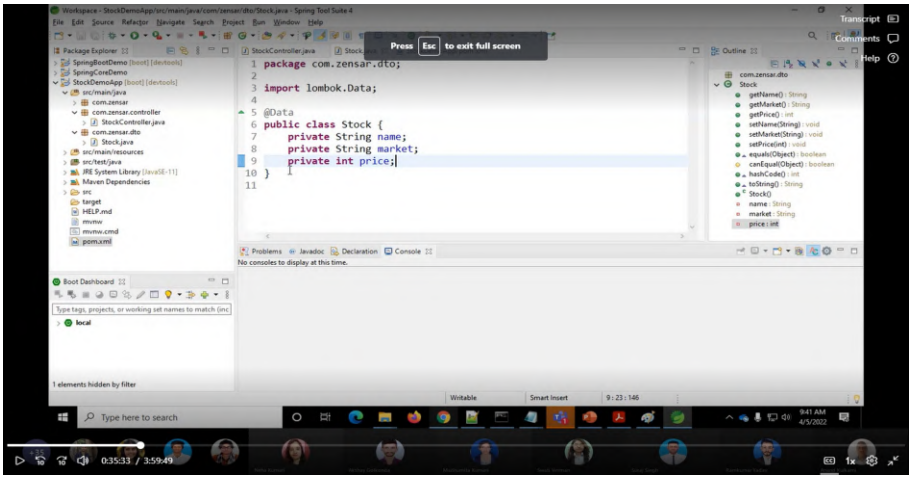
We can also specify order of Runners by providing @Order() annotation

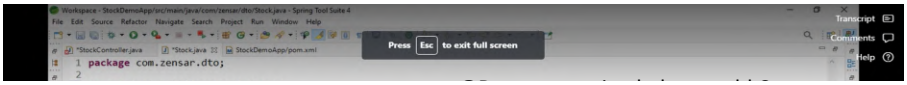
@Order(1) like this after @Component annotation
One for @Order(1) and for another @Order(2)

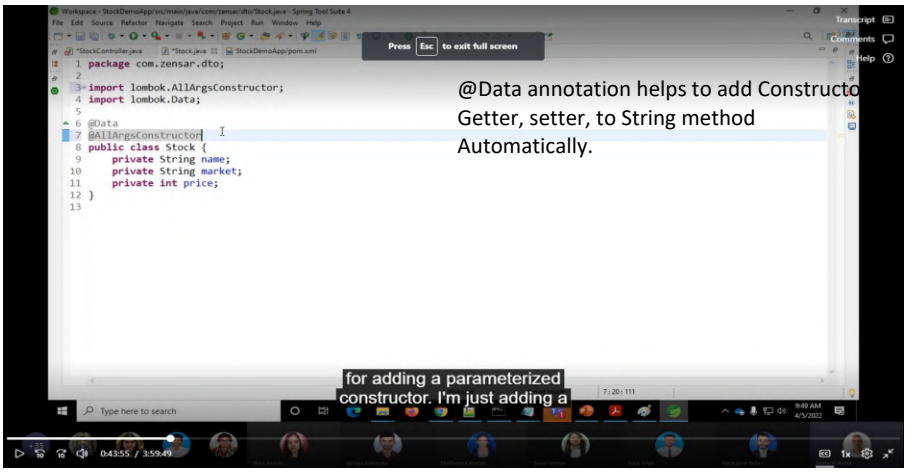
Another way to specify order we can use Interface called Ordered instead of using @Order annotation.

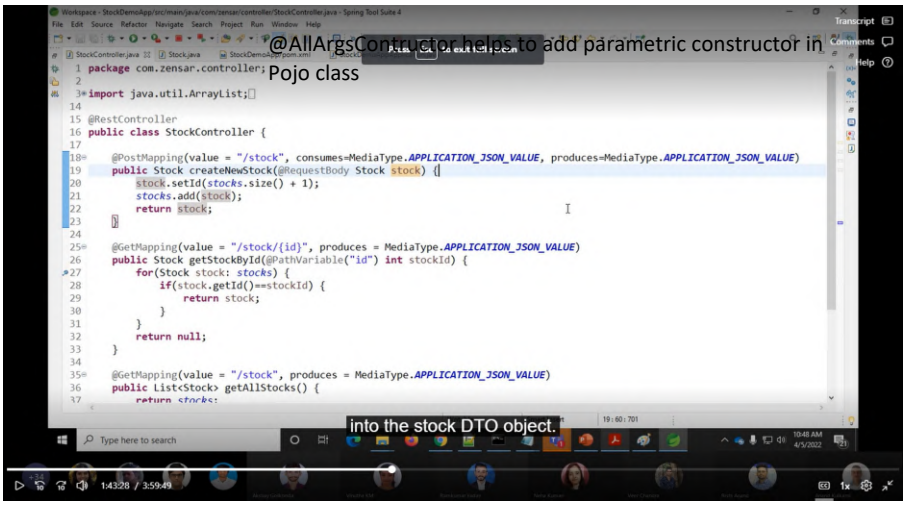


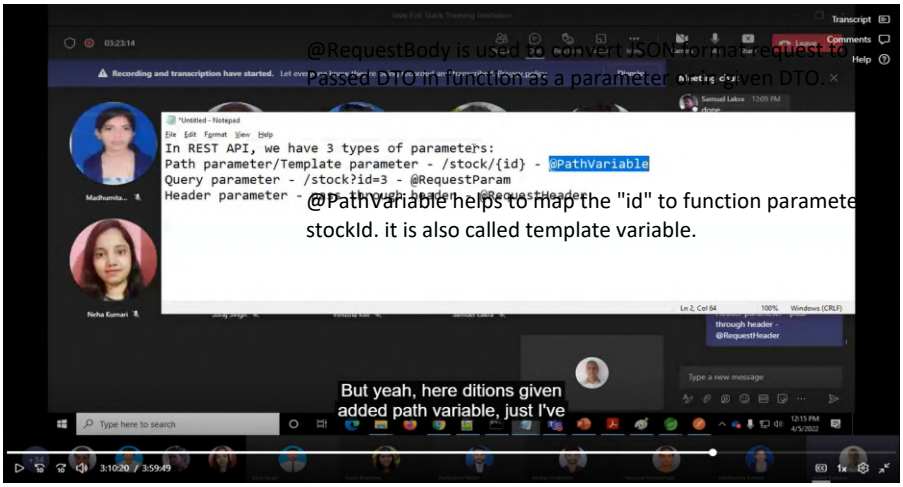
@RestController is used when I want to expose my controller as a restful server and when I want to expose services like creating, updating, deleting etc. all these functionalities for this purpose then I have to expose my entire class as a restful controller.

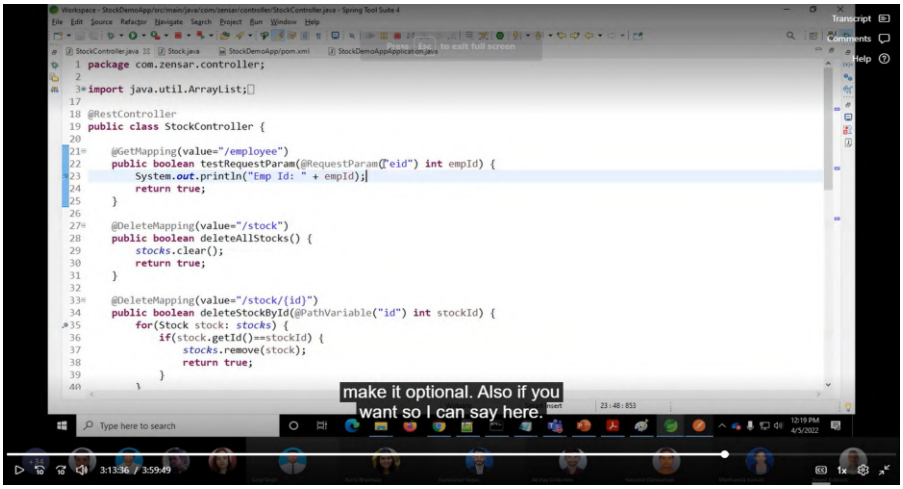




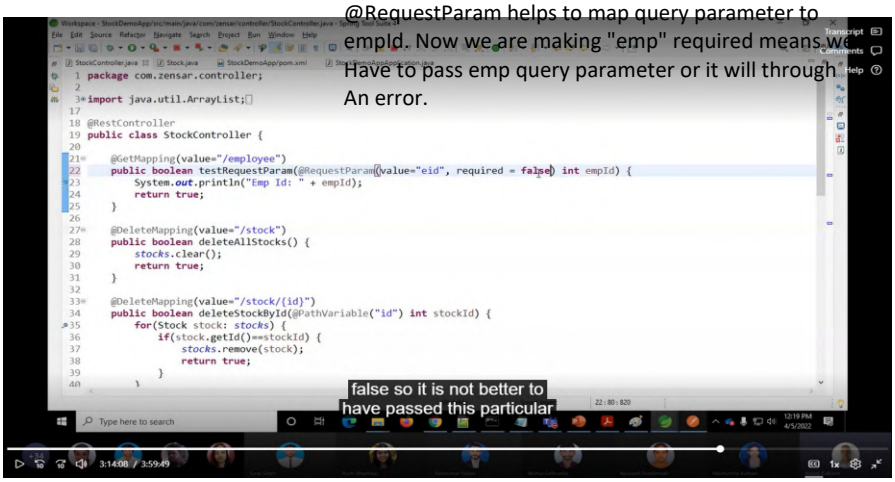

@Data annotation helps to add Constructo
Getter, setter, to String method
Automatically.
for adding a parameterized
constructor. I'm just adding a


@AllArgsConstructor helps to add parametric constructor in
Pojo class
into the stock DTO object.

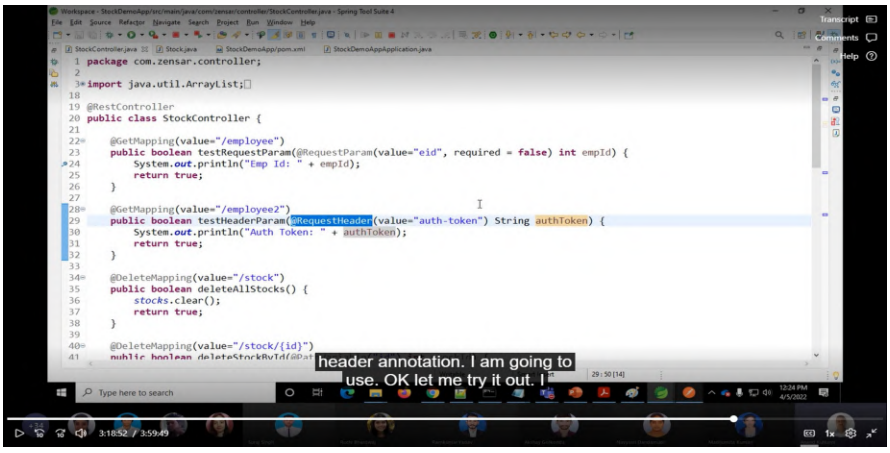

@RequestBody is used to convert JSON format request to
Passed DTO in function as a parameter of given DTO.
In REST API, we have 3 types of parameters:
Path parameter/Template parameter - /stock/{id} - @PathVariable
Query parameter - /stock?id=3 - @RequestParam
@PathVariable helps to map the "id" to function paramete
stockId. it is also called template variable.
But yeah, here ditions given
added path variable, just I've

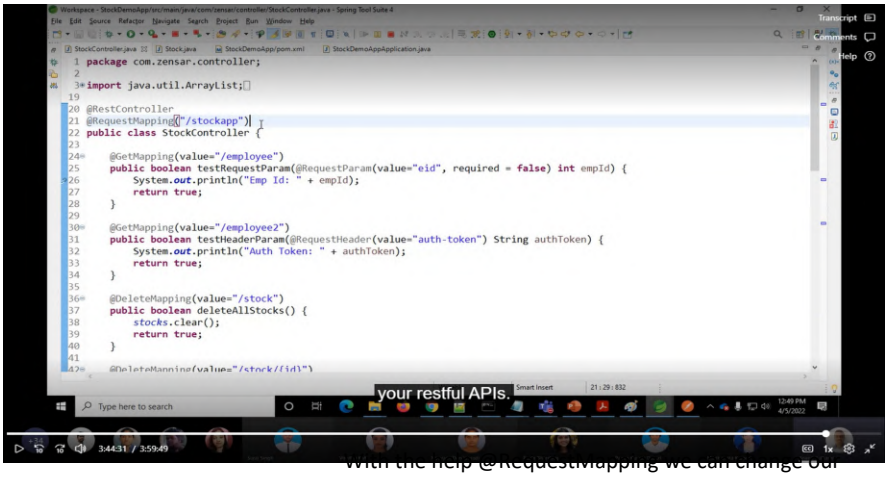


Making Query parameter optional by doing required false.



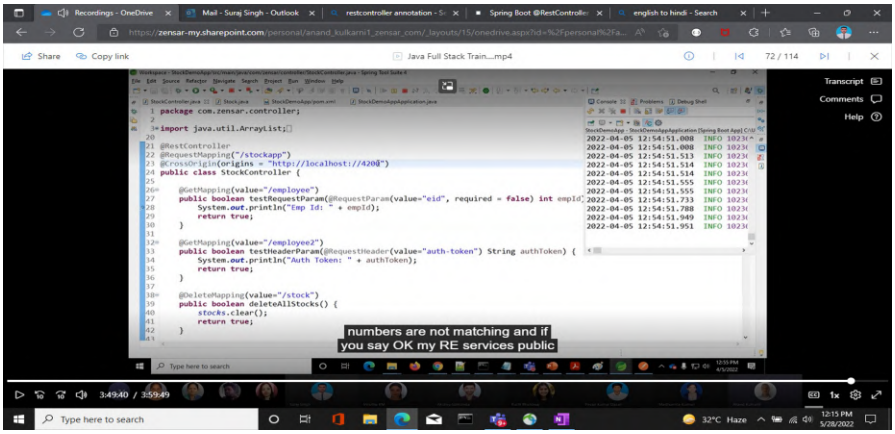
Header parameter

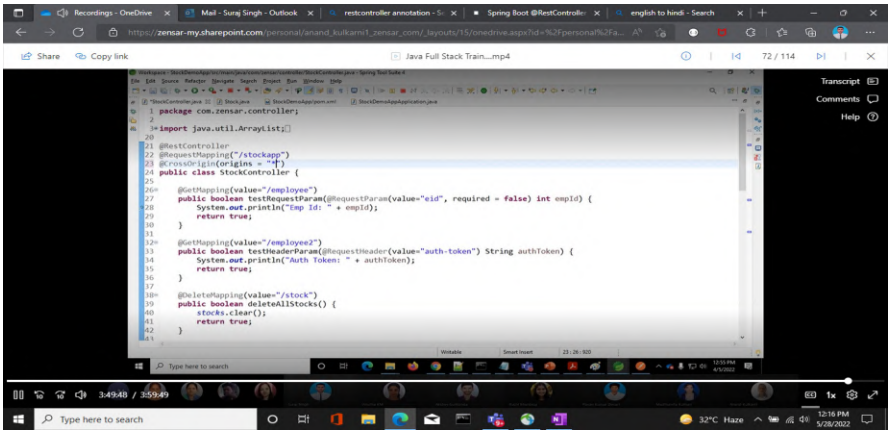




With the help @RequestMapping we can change our Base url

@Cross-Origin allow Client side to call server side rest api no matter if working on different port numbers.





Content negotiation is a technique described in the hypertext transfer protocol (HTTP) specification that allows a remote client to make a more specific request of a **web server**. The method gives the server the ability to host multiple types of files, documents in different languages, and more, each with varying quality levels.
In Spring boot in-build means automatically convert JSON request into DTO means we don't have to add any dependency because it happens implicitly but for XML we have to add dependency explicitly to convert XML request to DTO.

<dependency>
<groupId>com.fasterxml.jackson.dataformat</groupId>
<artifactId>jackson-dataformat-xml</artifactId>
</dependency>

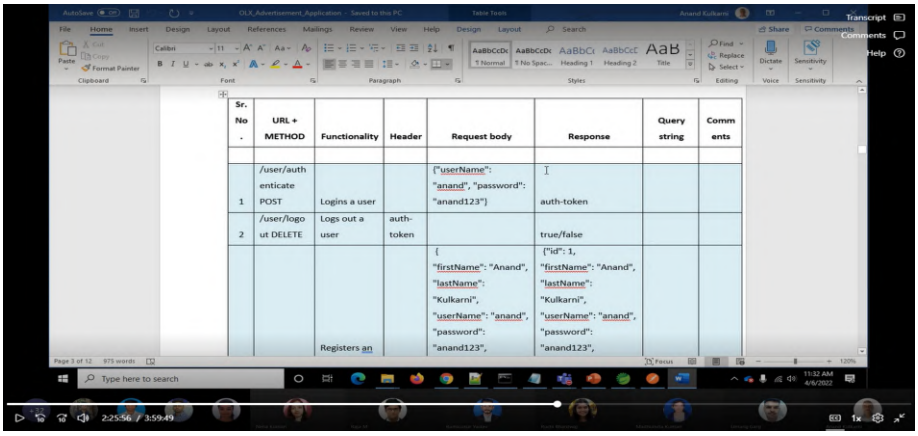
Content-type is a key in a header which is typically send during the sending a request from the client to the rest API server. Content-type tell weather the request body that you are sending in the request body in the xml formal or in the JSON format. For example : Content-Type : "application/json". Content-Type for request type.

Now by default Spring will return data in the form of JSON format. If we want to return response in the form xml format then we have to tell the spring to return response in the form of xml formal for that we have to do : Accept : "application/xml". Accept for response type.

These 2 things must be set on client side request. Because on server side we already added produces and consumes.

API Documentation is documentation of an API which gives information like API, Method type,
Request body required or not , what we have to attach in request body, what kind of response data we will get. This should documentation should be created automatically for that we will use OAS (Open API Specification) .Swagger is one of the OAS implementation of OAS. Swagger also give one advantage to allow to call or hit REST API using browser. For example giving information just like below.

This API Documentation provided to Client so that they can understand which API Will be stable for me to call and what kind of parameter we have to pass to get that response.

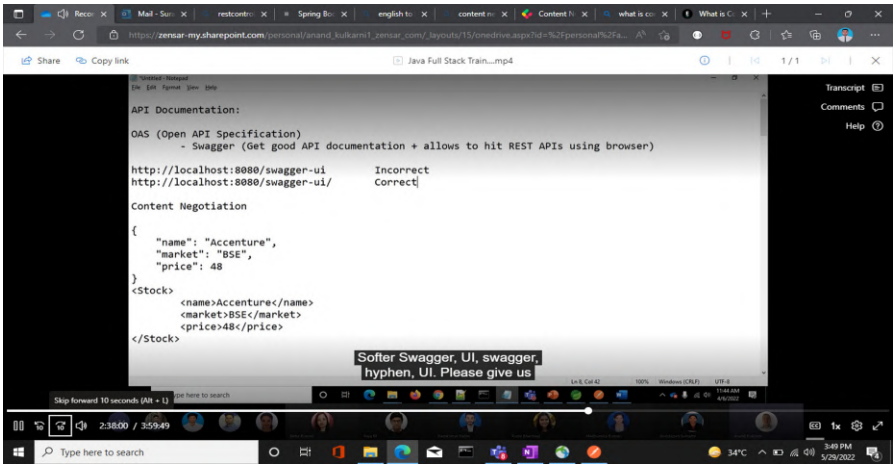


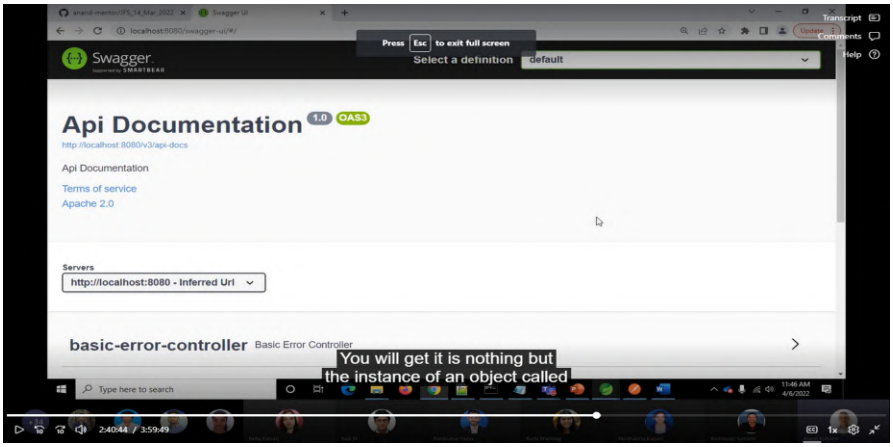
Sr. No.	URL + METHOD	Functionality	Header	Request body	Response	Query string	Comments
1	/user/authenticate POST	Logins a user		{"userName": "anand", "password": "anand123"}	auth-token		
2	/user/logout DELETE	Logs out a user	auth-token		true/false		
		Registers an		{ "firstName": "Anand", "lastName": "Kulkarni", "userName": "anand", "password": "anand123",	{"id": 1, "firstName": "Anand", "lastName": "Kulkarni", "userName": "anand", "password": "anand123",		

Dependency :::

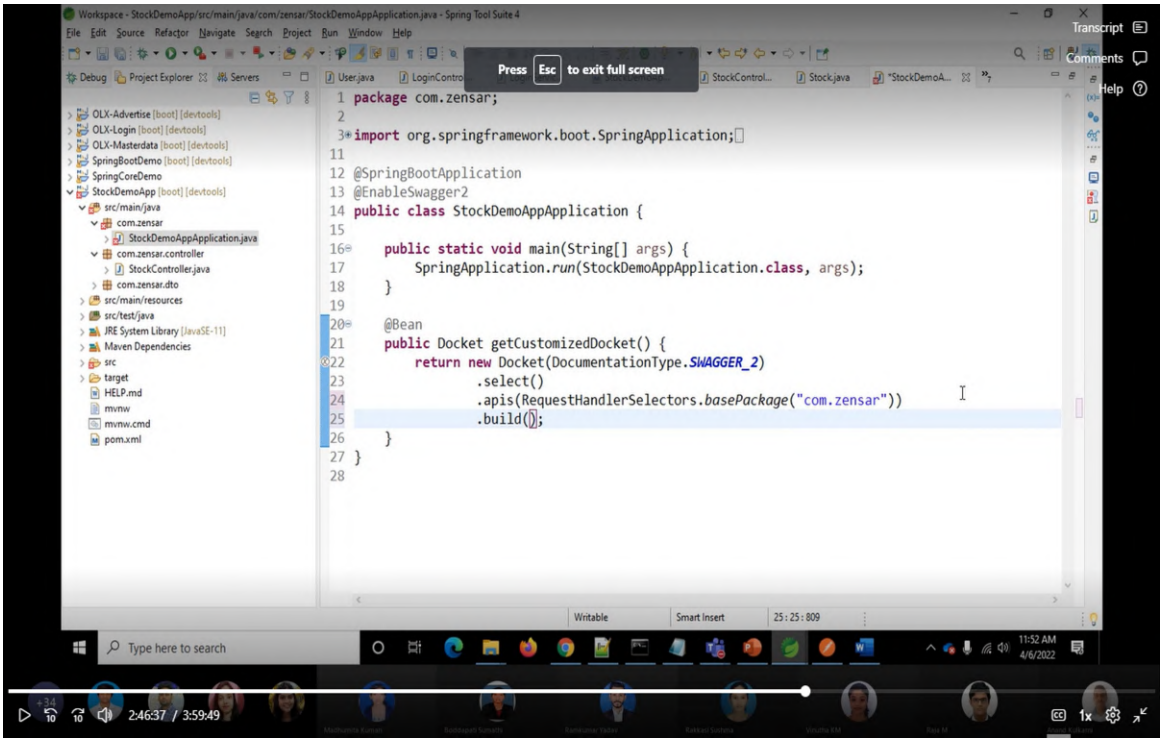
<dependency>
<groupId>io.springfox</groupId>
<artifactId>springfox-boot-starter</artifactId>
<version>3.0.0</version>
</dependency>

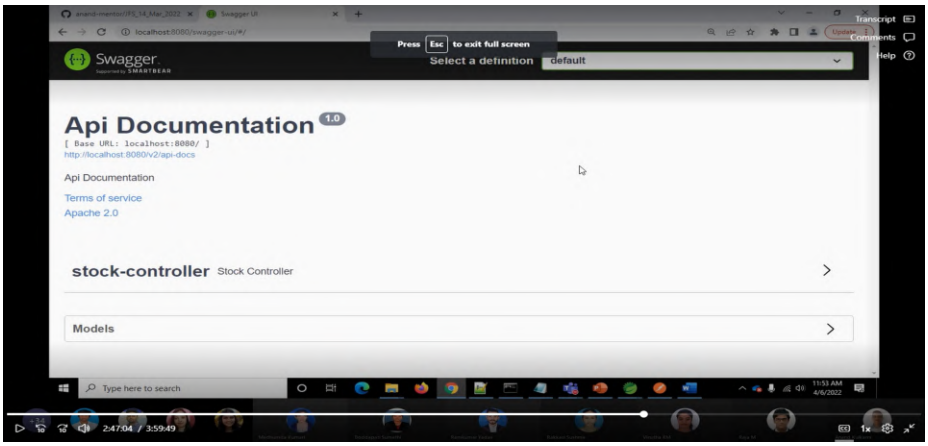
And Also change Spring boot version to 2.5.6



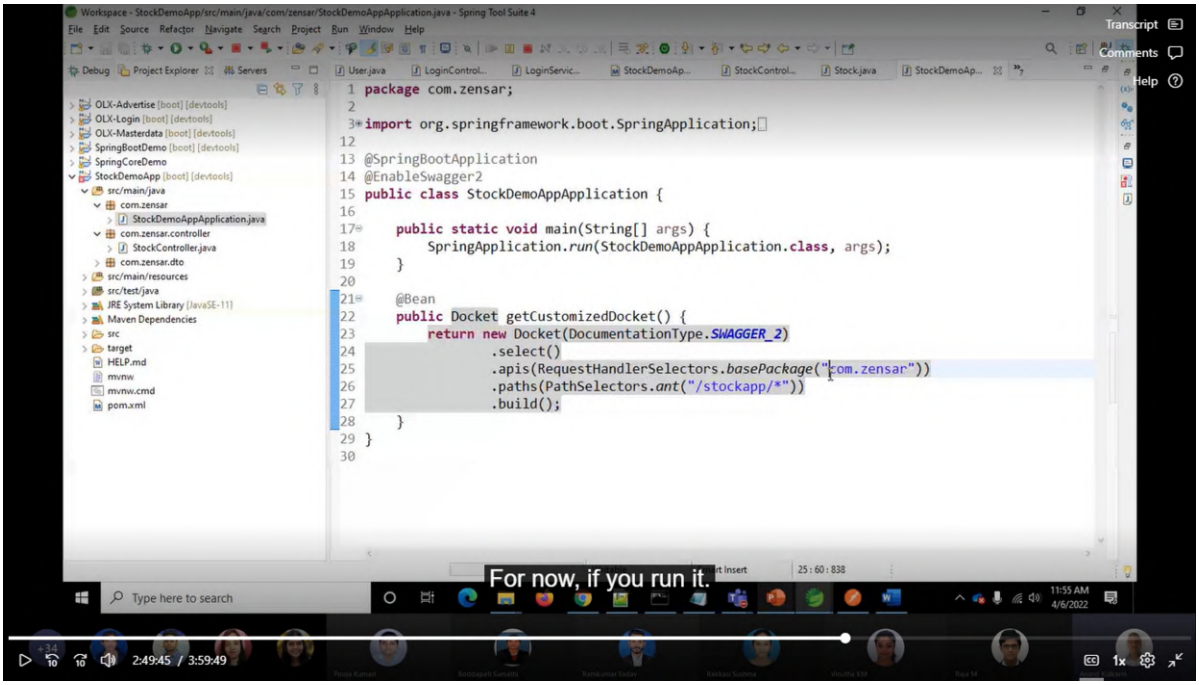


UI Page we are getting over there it is nothing but the instance of object is called as a Docket object.
Docket represents the customization of Swagger API documentation page. If we want to customize a Swagger API Documentation page then we have to return customize Docket object.

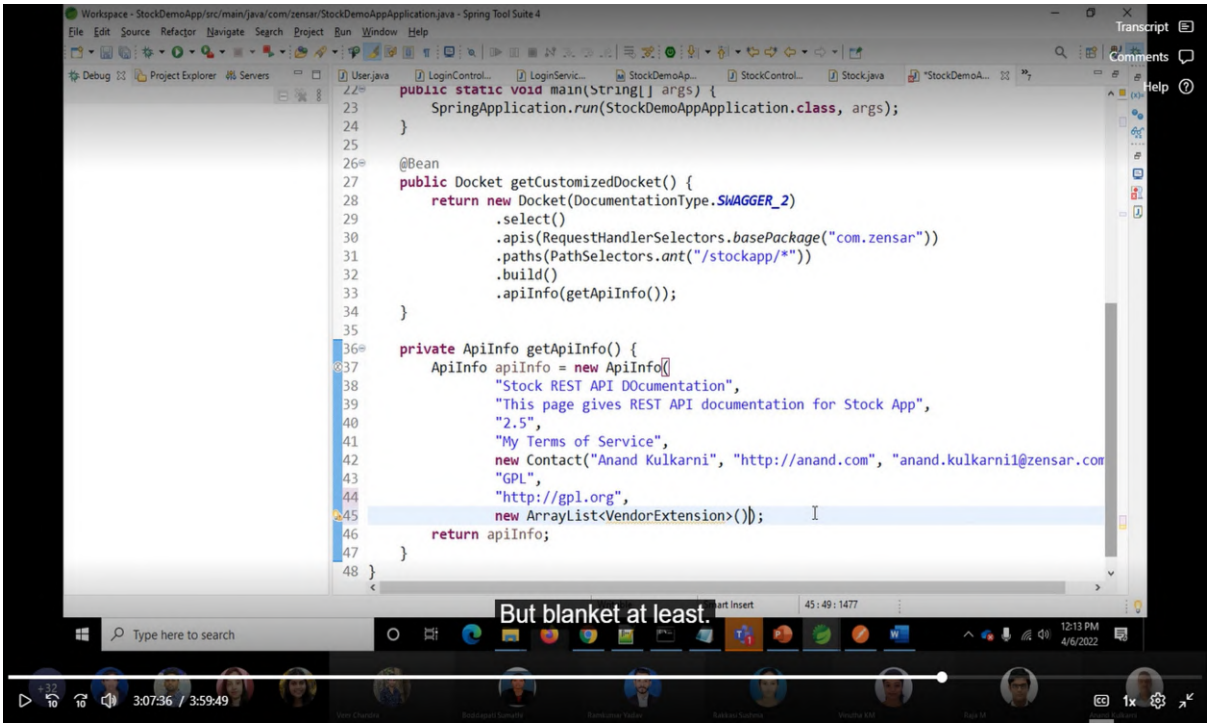


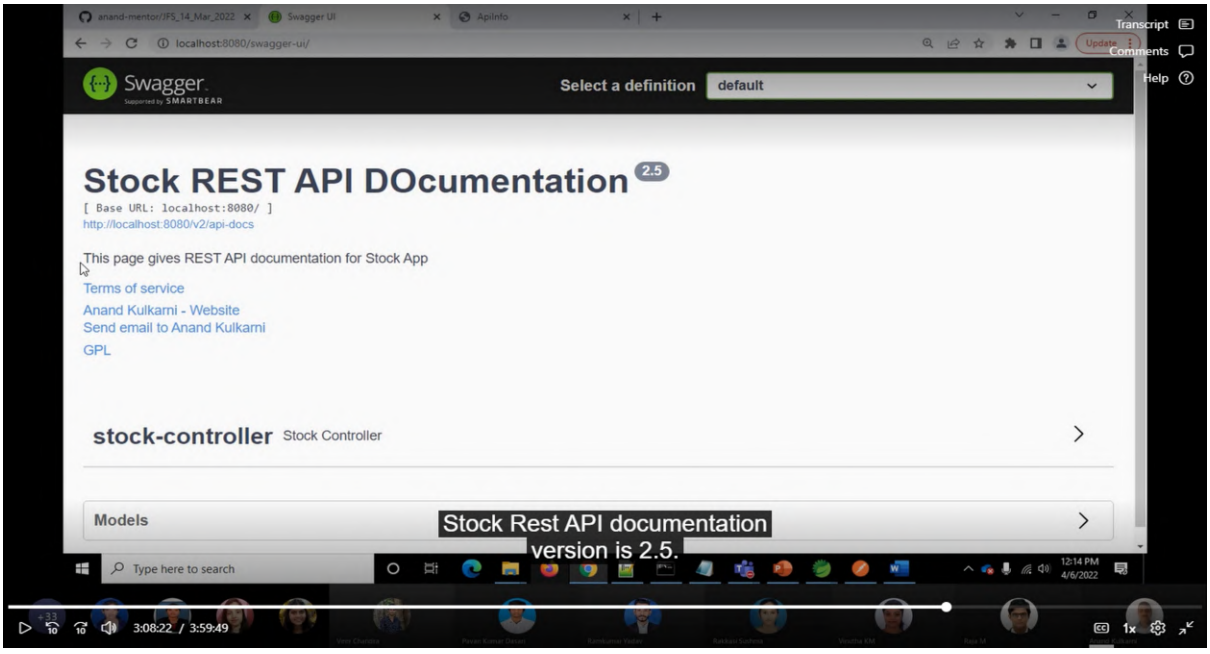


We can also add Path Selector so that only URL or APIs starts with "/stockapp/ " will be visible. But by adding this path parameter will not visible on Swagger API Documentation page. So we have to replace this apis() with base request to any instead of "/stockapp/ ".

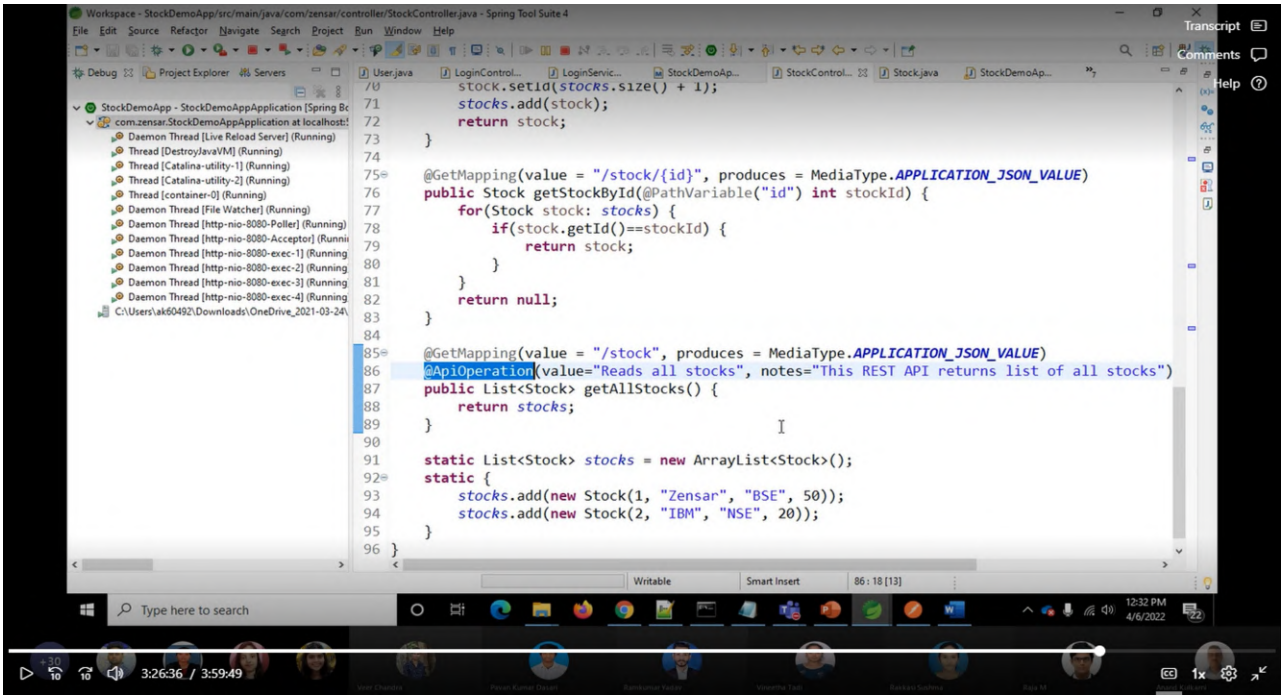


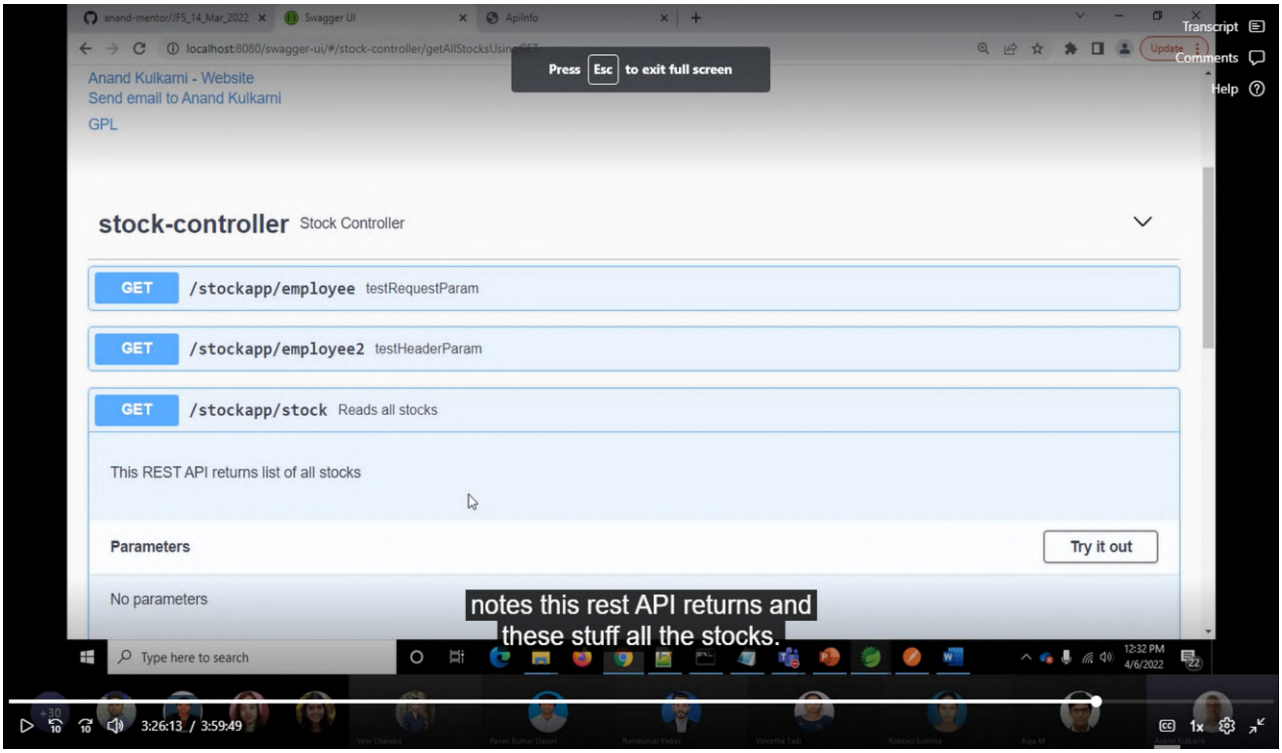
Customize more information on Swagger Web page we can do by adding some below line code.





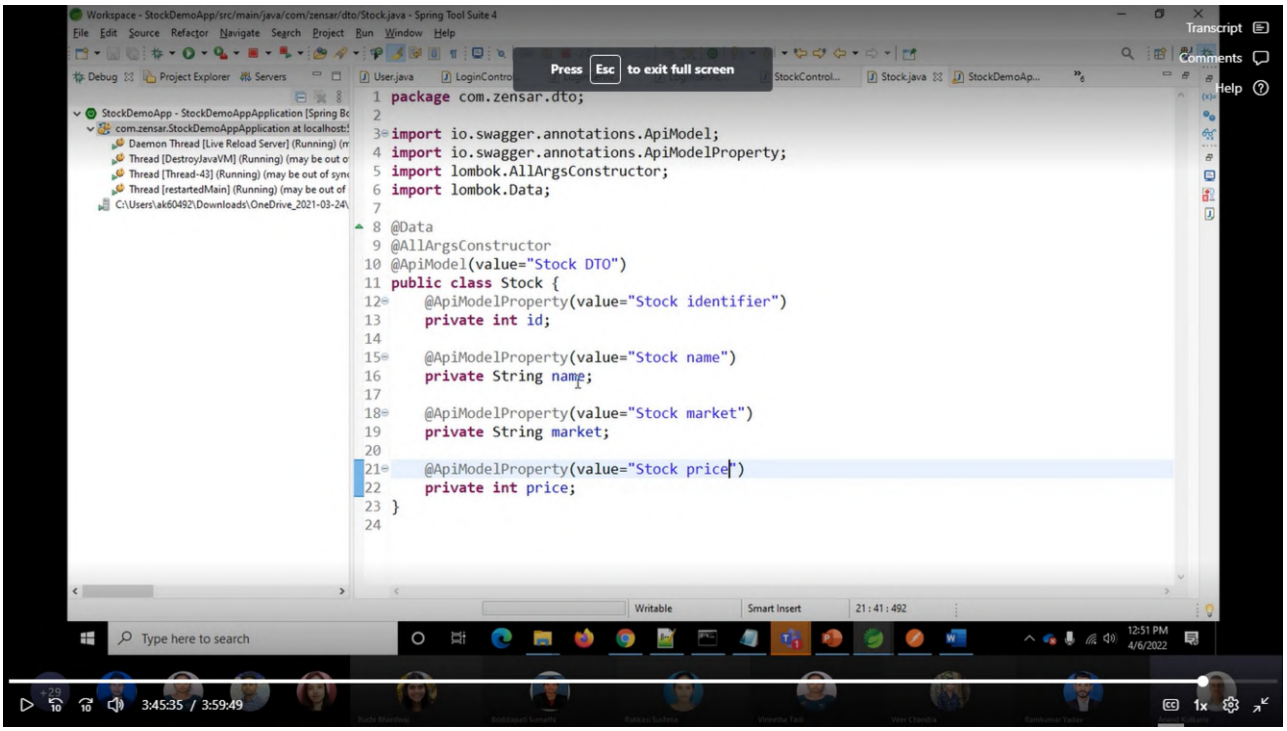
By seeing the API Client can not get more information about the what kind of data API will get so to give more information about API just Add @ApiOpeartion().

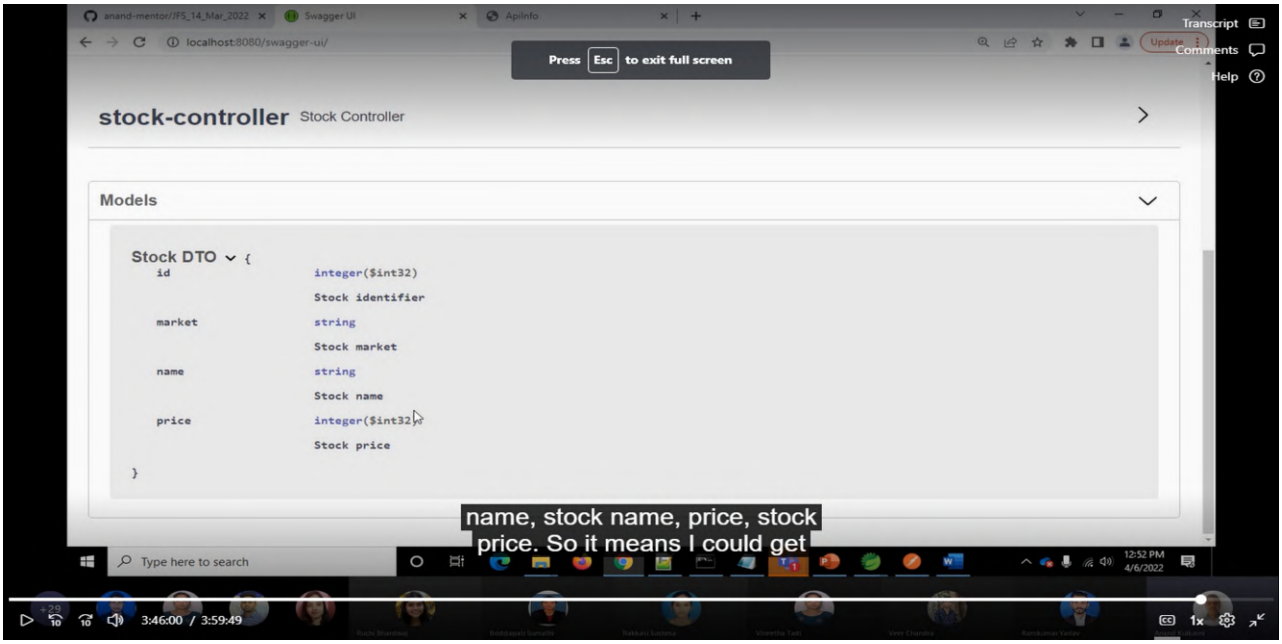




Adding parameter information add @ApiParam(value="" , name=""). Name must match with path variable.

To Add some DTO information to client add below line of code.





Spring Data

JDBC's full form is Java Database Connectivity. It is basically a Java API that is used to execute and connect query along with the database. It is low level a low level API. Cause every time we have to covert my Java object into SQL query.

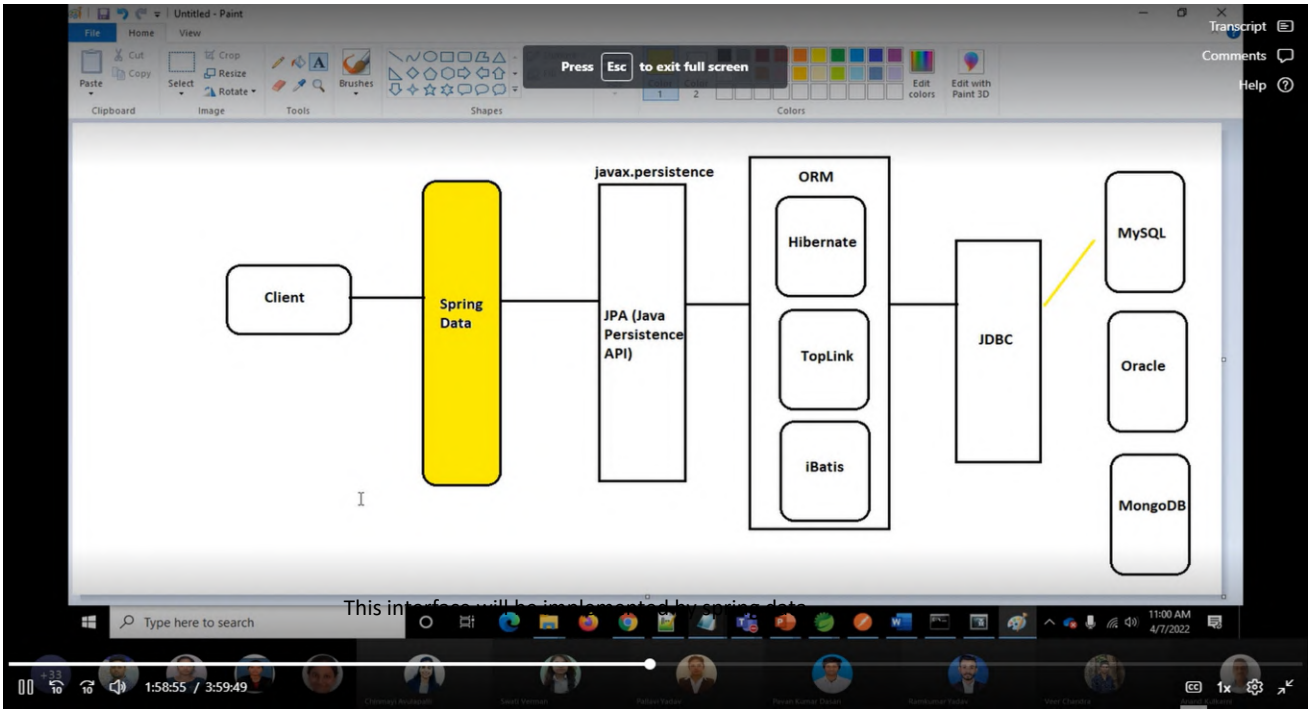
For more details visit to [What is JDBC? | Comprehensive Guide to Java Database Connectivity (educba.com)](#) given link.

We need some high level API which will allow us to work with Java objects only and we no need to bother about the SQL query. And by calling some function it persist the data inside database. The API which provides high level APIs or high level Functions where we completely hide JDBC code under the hood such API is called ORM.

ORM's full form is Object Relational Model. ORM internally calls JDBC code. It provide some high level function which ask for Object which we want to persist inside database. ORM is a concept which is implemented using Hibernate, Torque, TopLinks etc. Suppose after 1 year development of Application using Hibernate client says to change the ORM Hibernate to ORM TopLinks then we have to change all java Hibernate code to Toplinks. To Avoid this thing we go for ORM Independent feature this is provided by JPA.

JPA's full form is Java Persistent API. It provides common specification for ORM. Which is implemented most of the ORM tools so instead of working for a specific ORM tool always work on JPA because JPA provides a package called javax.persistance. JPA provide us ORM independent facilities but still we have to write a lot of code. To avoid this thing we will use one more interface is called Spring Data.

Spring Data now as a client I will talk to Spring Data and Spring Data will call JPA under the hood. Spring Data allows me to work at abstract label where developer effectively communicate with database system without write a single piece of code.



Spring Data Dependency

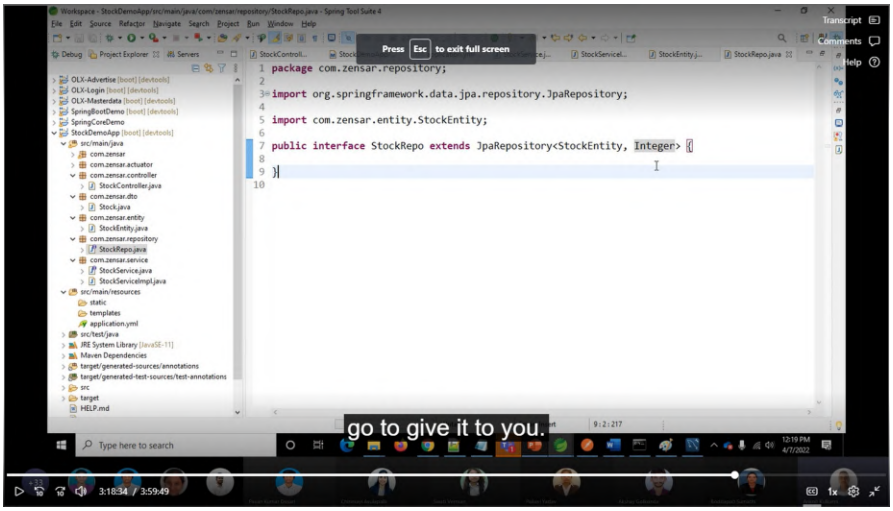
```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Dependency for Sql Driver

```
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<scope>runtime</scope>
</dependency>
```

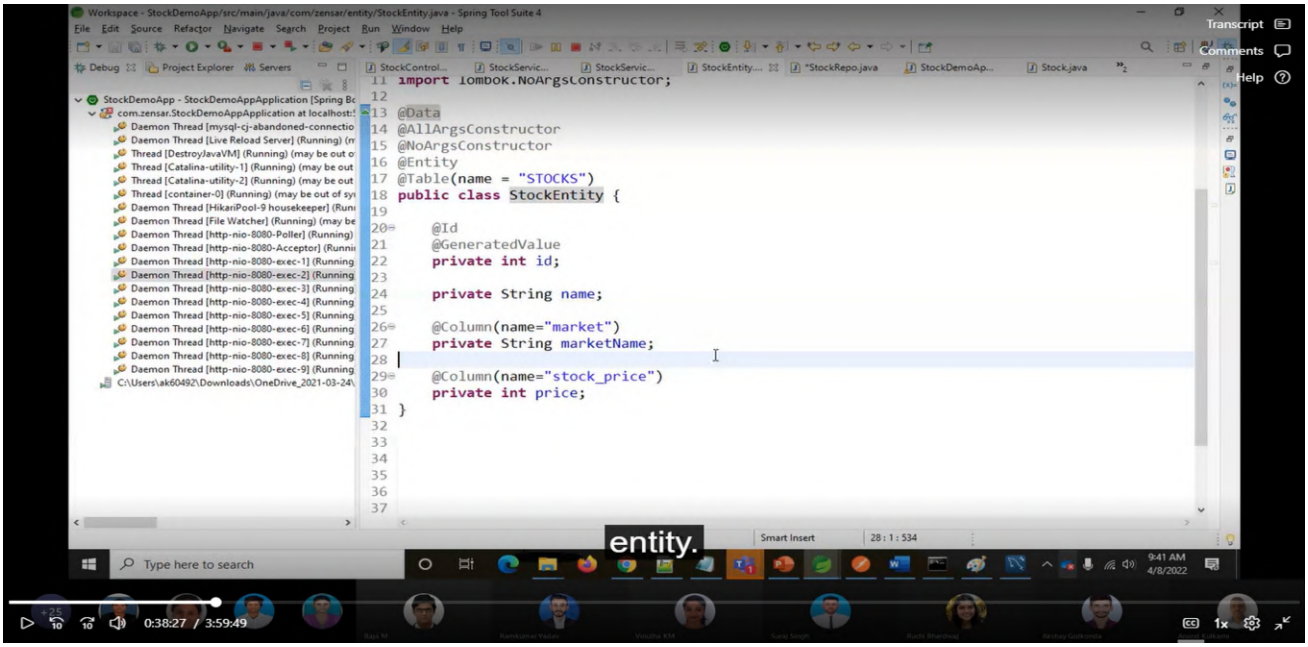
```
spring:
   datasource:
   url: jdbc:mysql://localhost:3306/jss_db_user
   username: root
   password: root
  jpa:
   hibernate:
     ddl-auto: update
   properties:
     hibernate :
       dialect: org.hibernate.dialect.MySQL8Dialect
```

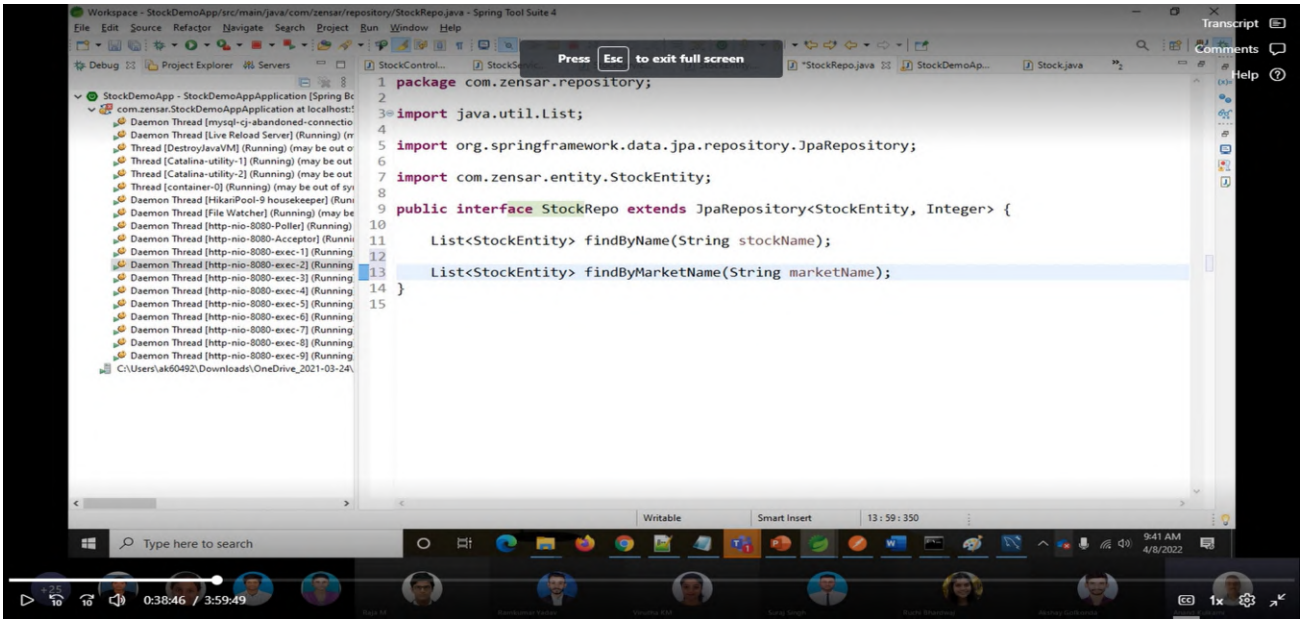
Spring Data provide a JPArepository which internally talk to Database using Model.
@Id annotation tells that this is a primary key. But to generate a primary key we use @GeneratedValue annotation .



Reason why we creating DTO & Entities because DTO is used to display data to client or talk to client but Entity is used to talk to table in Database in the form of row & column. And also possible that attributes in Entity is more than DTO or attributes in DTO more than entity.

If we want to create a custom finder function in Repo interface than we have to go into entity class and with the help of attribute we can create custom finder function but first character of attribute should be capital.





Only one thing is that we have to follow naming conversion and spring data implements automatically. but If we don't follow naming conversion still wants that spring data implements your custom finder function then we have to say spring data to fire a query when a function will be called.

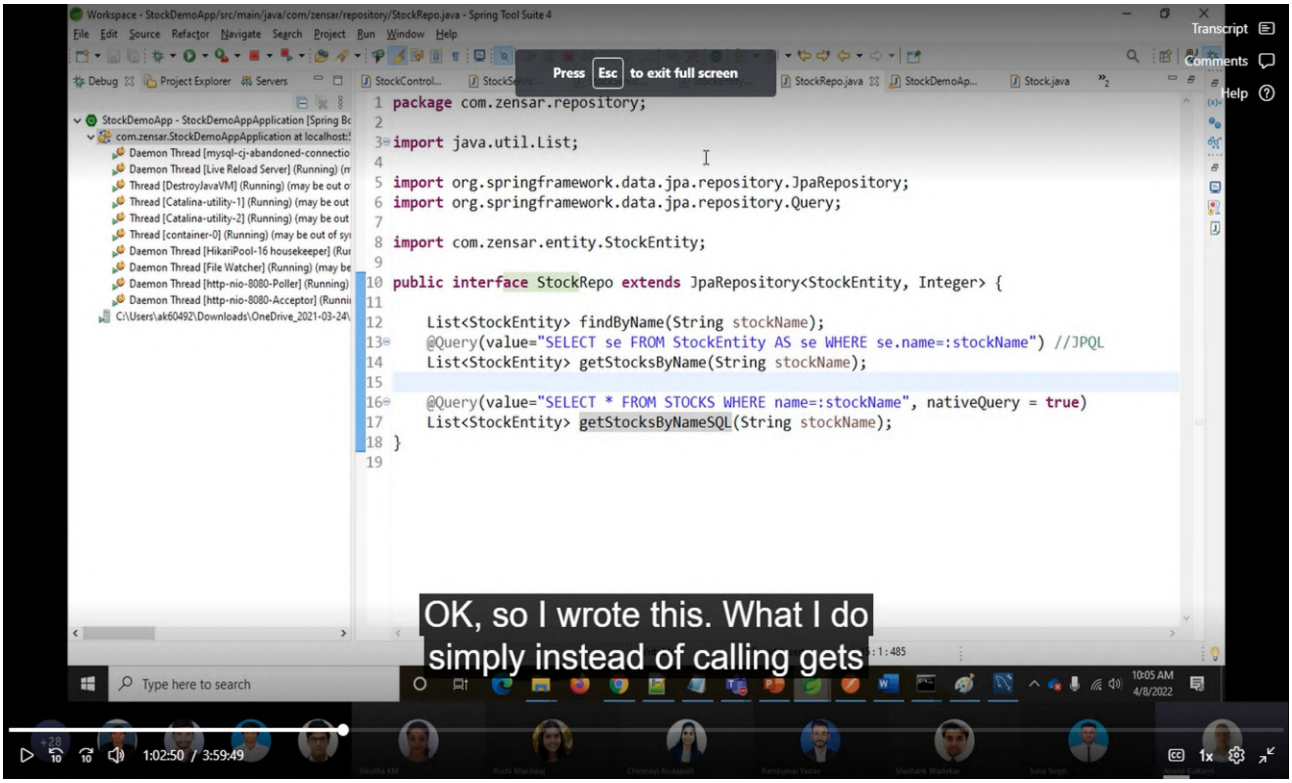
Two types of query we can write

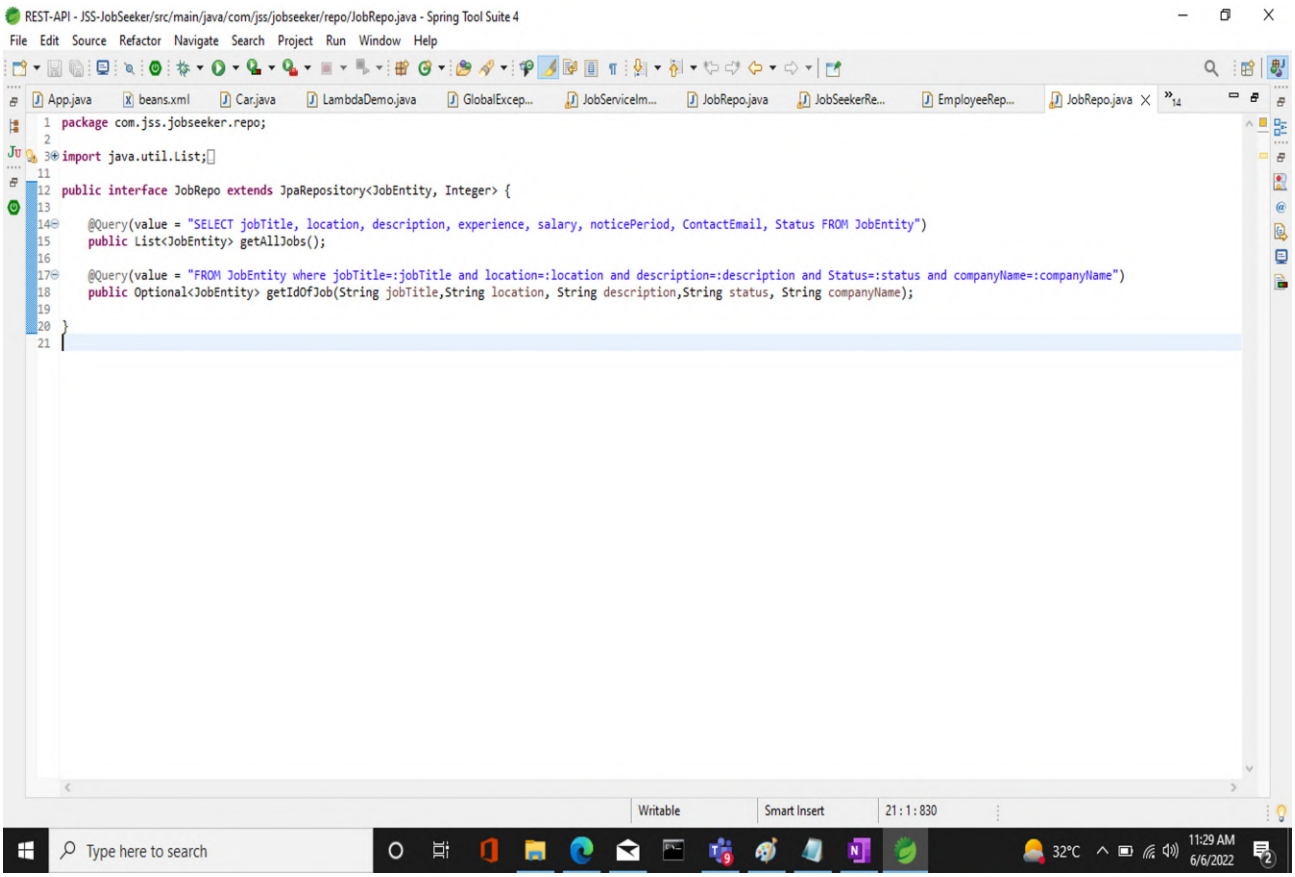
(i) JPQL

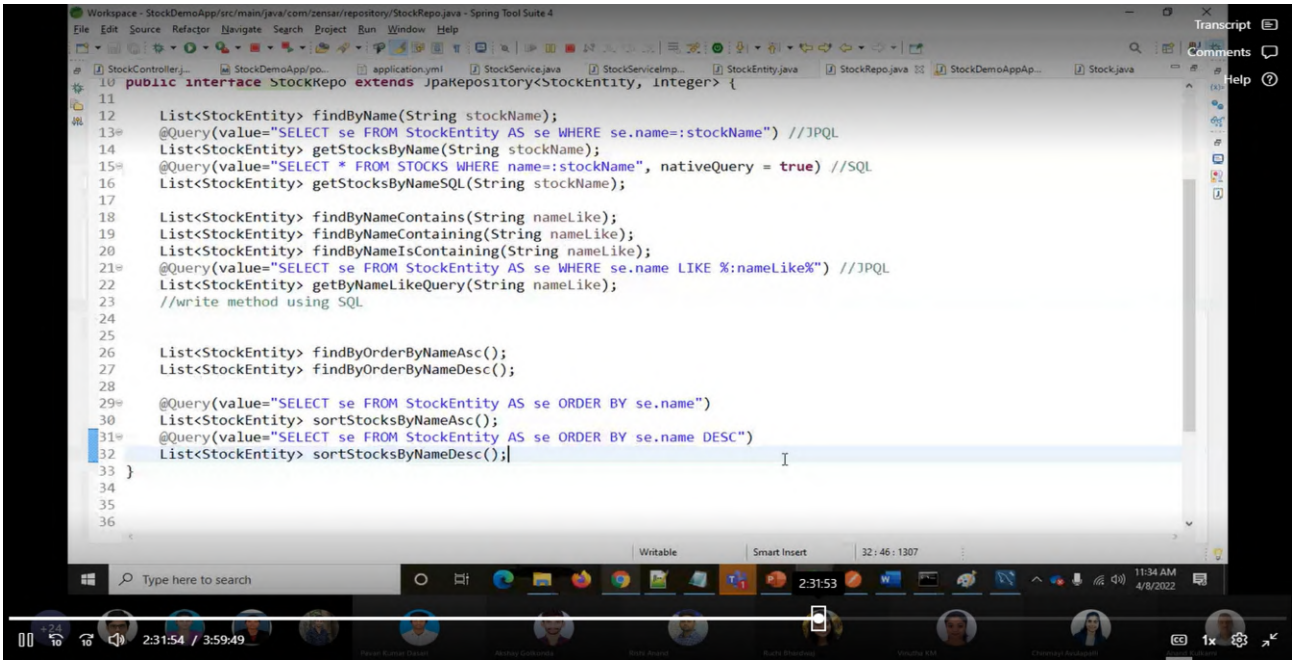
(ii) SQL

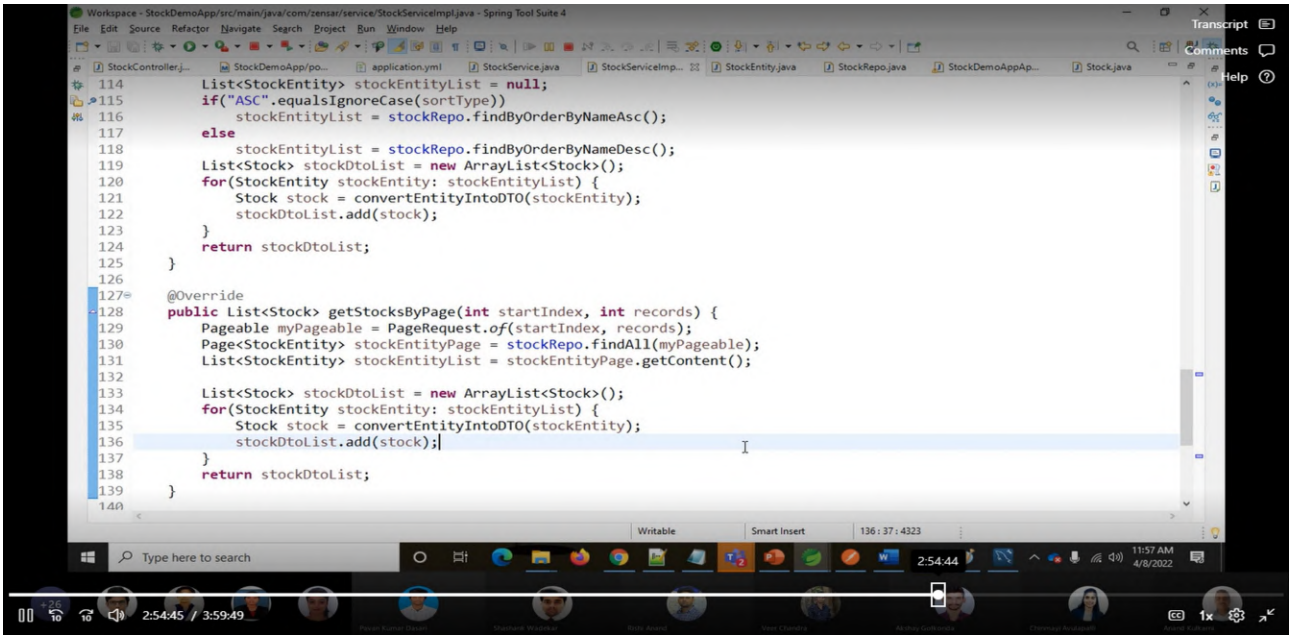
JPQL is Java Persistence Query Language defined in JPA specification. It is used to create queries against entities to store in a relational database. JPQL is developed based on SQL syntax. But it won't affect the database directly. It

SQL is a standard language for storing, manipulating and retrieving data in databases. In Spring data every query we wrote act as a JPQL. To make it SQL query we have to use nativeQuery = true in @Query annotation.




REST-API - JSS-JobSeeker/src/main/java/com/jss/jobseeker/repo/JobRepo.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
App.java beans.xml Car.java LambdaDemo.java GlobalExcep... JobServiceIm... JobRepo.java JobSeekerRe... EmployeeRep... JobRepo.java
1 package com.jss.jobseeker.repo;
2
3 import java.util.List;
11
12 public interface JobRepo extends JpaRepository<JobEntity, Integer> {
13
14 @Query(value = "SELECT jobTitle, location, description, experience, salary, noticePeriod, ContactEmail, Status FROM JobEntity")
15 public List<JobEntity> getAllJobs();
16
17 @Query(value = "FROM JobEntity where jobTitle=:jobTitle and location=:location and description=:description and Status=:status and companyName=:companyName")
18 public Optional<JobEntity> getIdOfJob(String jobTitle,String location, String description,String status, String companyName);
19
20 }
21
Writable
Smart Insert
21 : 1 : 830
Type here to search
32°C
11:29 AM
6/6/2022


Workspace - StockDemoApp/src/main/java/com/zensar/repository/StockRepo.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Transcript
Comments
Help
public interface StockRepo extends JpaRepository<StockEntity, Integer> {
List<StockEntity> findByName(String stockName);
@Query(value="SELECT se FROM StockEntity AS se WHERE se.name=:stockName") //JPQL
List<StockEntity> getStocksByName(String stockName);
@Query(value="SELECT * FROM STOCKS WHERE name=:stockName", nativeQuery = true) //SQL
List<StockEntity> getStocksByNameSQL(String stockName);
List<StockEntity> findByNameContains(String nameLike);
List<StockEntity> findByNameContaining(String nameLike);
List<StockEntity> findByNameIsContaining(String nameLike);
@Query(value="SELECT se FROM StockEntity AS se WHERE se.name LIKE %:nameLike%") //JPQL
List<StockEntity> getByNameLikeQuery(String nameLike);
//write method using SQL
List<StockEntity> findByOrderByNameAsc();
List<StockEntity> findByOrderByNameDesc();
@Query(value="SELECT se FROM StockEntity AS se ORDER BY se.name")
List<StockEntity> sortStocksByNameAsc();
@Query(value="SELECT se FROM StockEntity AS se ORDER BY se.name DESC")
List<StockEntity> sortStocksByNameDesc();
}
Writable
Smart Insert
32 : 46 : 1307
Type here to search
2:31:53
11:34 AM
4/8/2022
2:31:54 / 3:59:49
1x

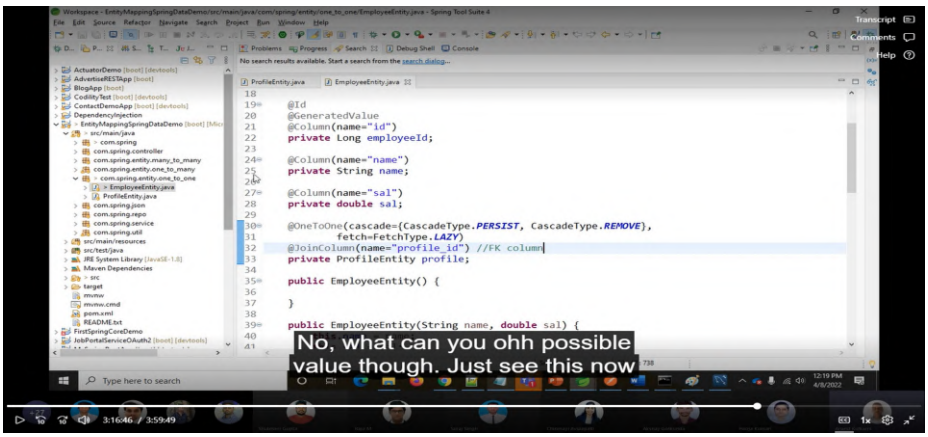



```
114         List<StockEntity> stockEntityList = null;
115         if("ASC".equalsIgnoreCase(sortType))
116             stockEntityList = stockRepo.findByOrderByNameAsc();
117         else
118             stockEntityList = stockRepo.findByOrderByNameDesc();
119         List<Stock> stockDtoList = new ArrayList<Stock>();
120         for(StockEntity stockEntity: stockEntityList) {
121             Stock stock = convertEntityIntoDTO(stockEntity);
122             stockDtoList.add(stock);
123         }
124         return stockDtoList;
125     }
126
127     @Override
128     public List<Stock> getStocksByPage(int startIndex, int records) {
129         Pageable myPageable = PageRequest.of(startIndex, records);
130         Page<StockEntity> stockEntityPage = stockRepo.findAll(myPageable);
131         List<StockEntity> stockEntityList = stockEntityPage.getContent();
132
133         List<Stock> stockDtoList = new ArrayList<Stock>();
134         for(StockEntity stockEntity: stockEntityList) {
135             Stock stock = convertEntityIntoDTO(stockEntity);
136             stockDtoList.add(stock);
137         }
138         return stockDtoList;
139     }
140
```

Relationships we have 3 types of relationships.

(i) One to One
(ii) one to Many
(iii) Many to Many

Cascading means whatever the operations we applied on parent same operations we want to apply on child or not. That particular option we can configure using a type called as cascade Type.




```
18
19      @Id
20      @GeneratedValue
21      @Column(name="id")
22      private Long employeeId;
23
24      @Column(name="name")
25      private String name;
26
27      @Column(name="sal")
28      private double sal;
29
30      @OneToOne(cascade={CascadeType.PERSIST, CascadeType.REMOVE},
31                fetch=FetchType.LAZY)
32      @JoinColumn(name="profile_id") //FK column
33      private ProfileEntity profile;
34
35      public EmployeeEntity() {
36
37      }
38
39      public EmployeeEntity(String name, double sal) {
40
41
```

CascadeType.Persist is used when we try to save are Parent data into database then save Child data into database also.

CascadeType.Remove is also carried forward parent to child entity also.

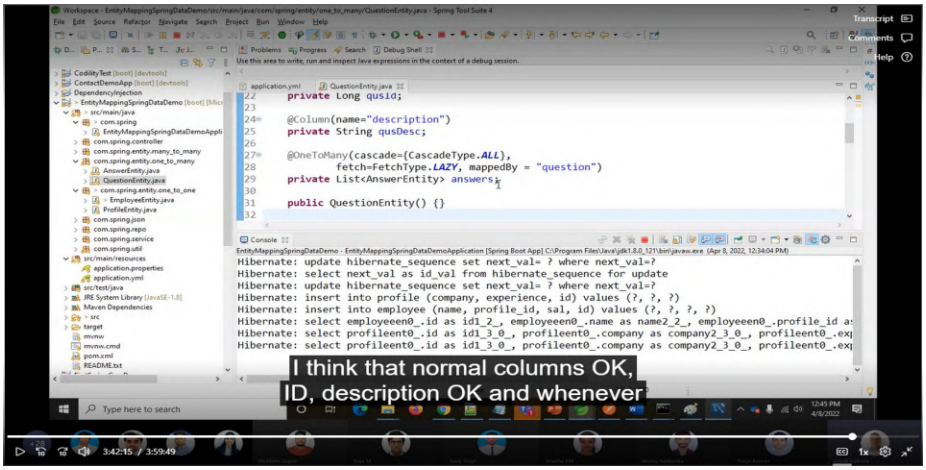
Fetch used to control a data which we want to fetch from database or not. Suppose we want to control data of child entity, to fetch or not while we getting data of parent we can do that using fetchType.

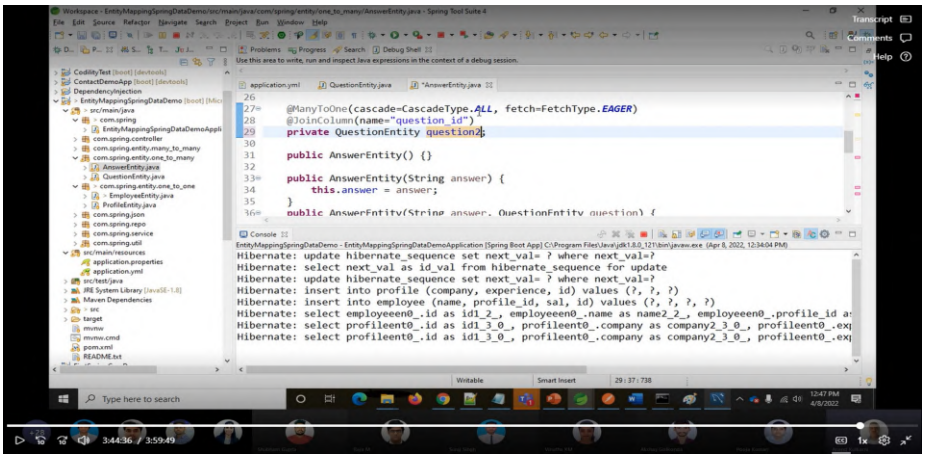
FetchType having 2 types.

fetchType.LAZY is used when we don't want to load data of child entity.
fetchType.EAGER is used when no matter child entity have multiple row and column you want to fetch the data of parent entity you want to load data.

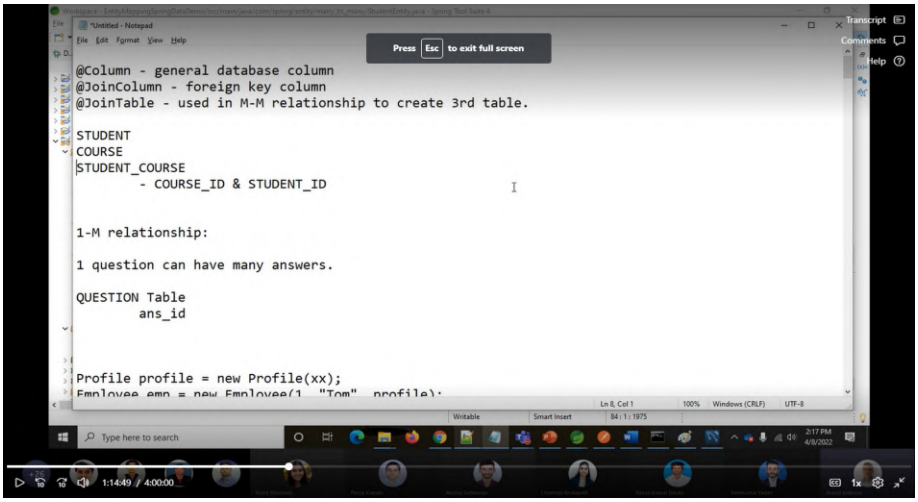
A one-to-many relationship between two entities is defined by using the @OneToMany annotation in Spring Data JPA. It declares the mappedBy element to indicate the entity that owns the bidirectional relationship. Usually, the child entity is one that owns the relationship and the parent entity contains the @OneToMany annotation.

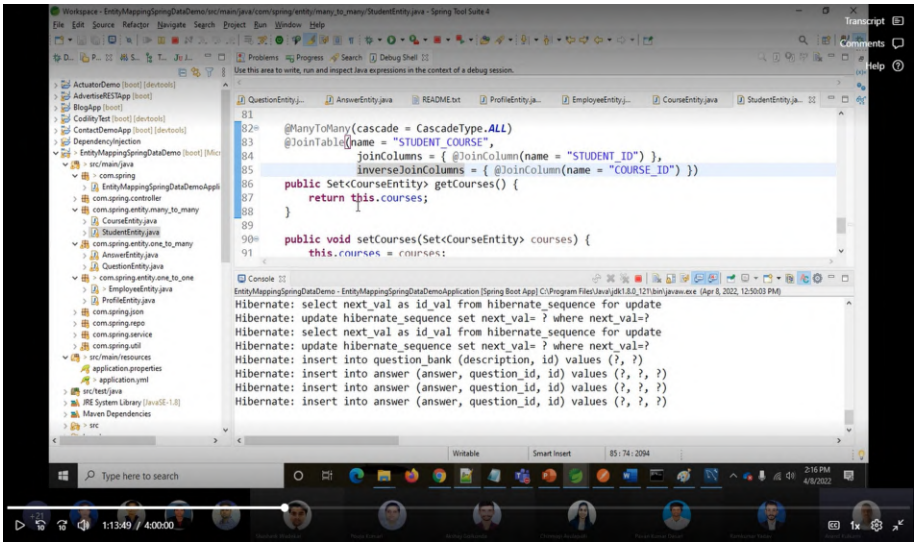
mappedBy element indicate the attribute name which defined inside the child entity to map the one to many relation.

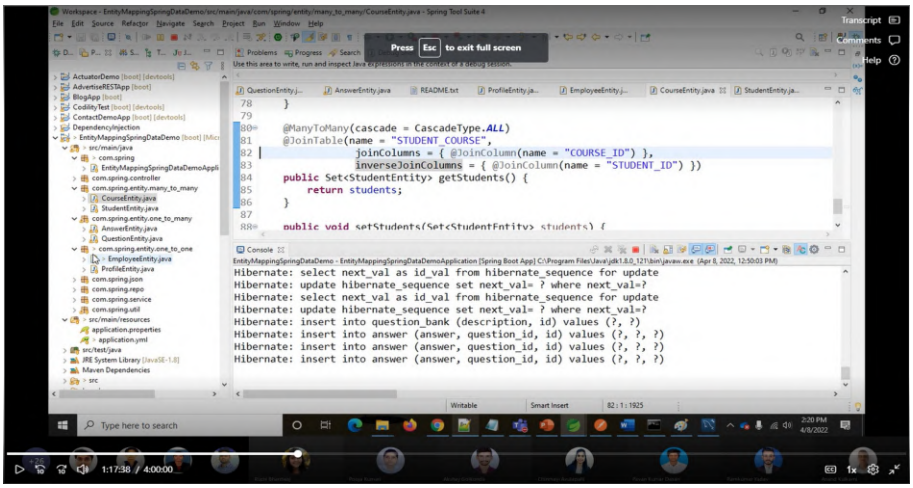


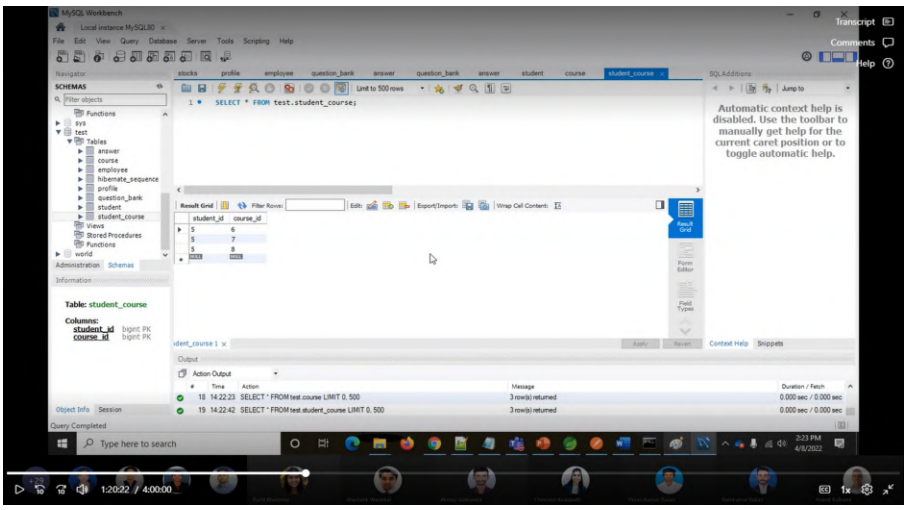


In Many to Many 3rd table is required. It is complosory to create 3rd table.









JPA provide us Criteria API. JPA says that whenever you want to create Criteria Dynamically. JPA provide a object which is called as entity manager. Entity manager is a heart of a JPA. Using this object we can implement criteria api. Using this object we are able to create criteria builder object.

the criteria builder is **used to find one or more records that match the criteria**.

the criteria query is used for creating query for a specific entity.

The root object which is used return specific field. Every criteria written predicate.

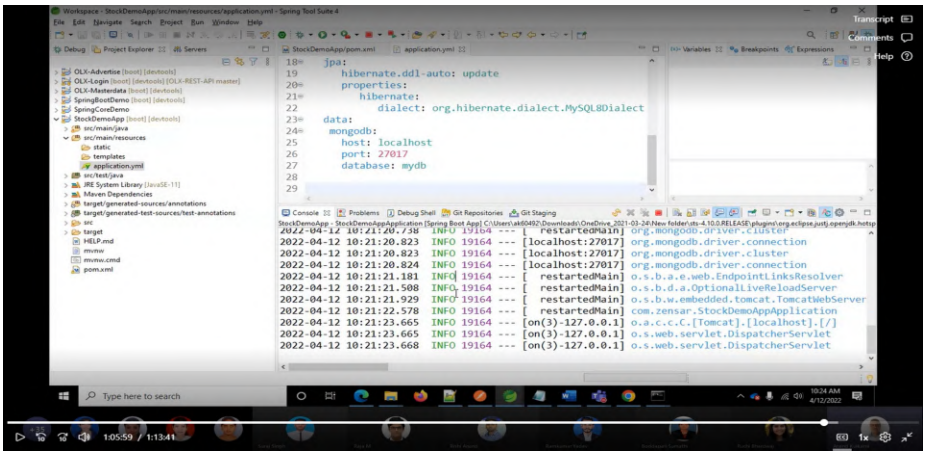
The Predicate class is part of the Criteria API and is **used to construct where clauses**.

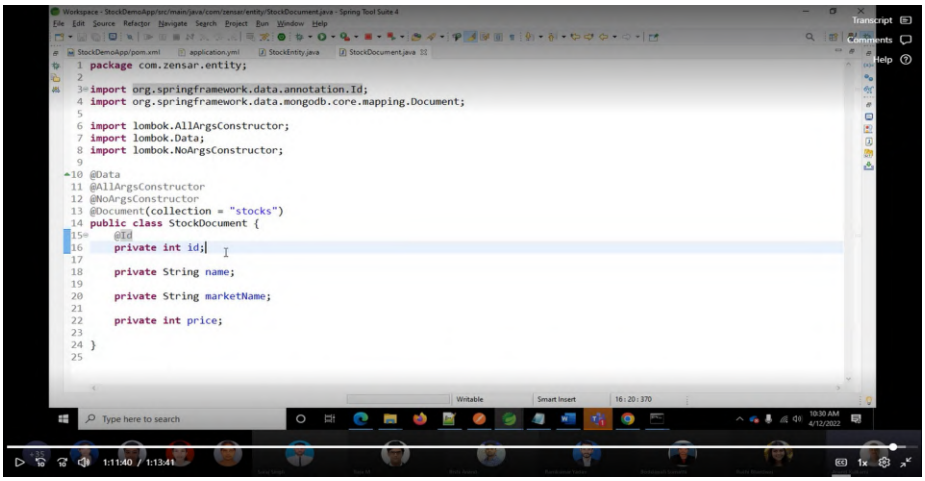
Mongo DB

Dependency for MongoDB Driver

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

Always run mongod on cmd to run server and mongo to run client.

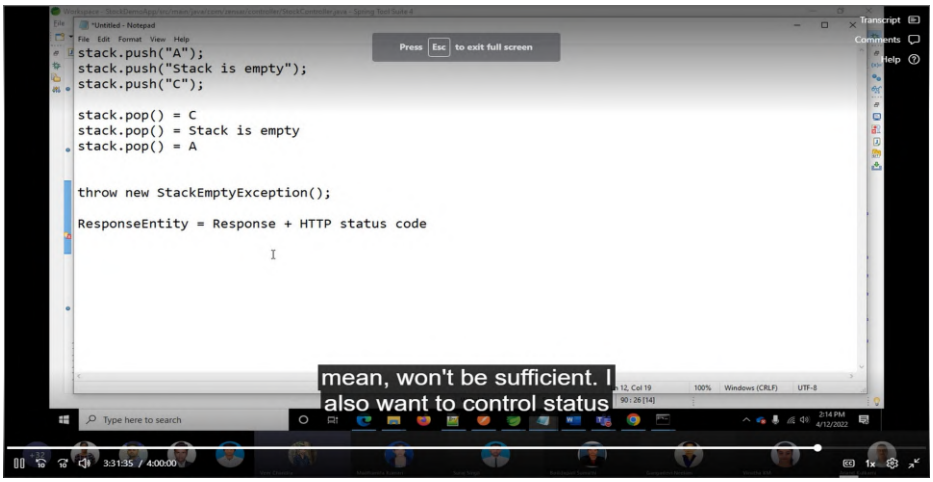


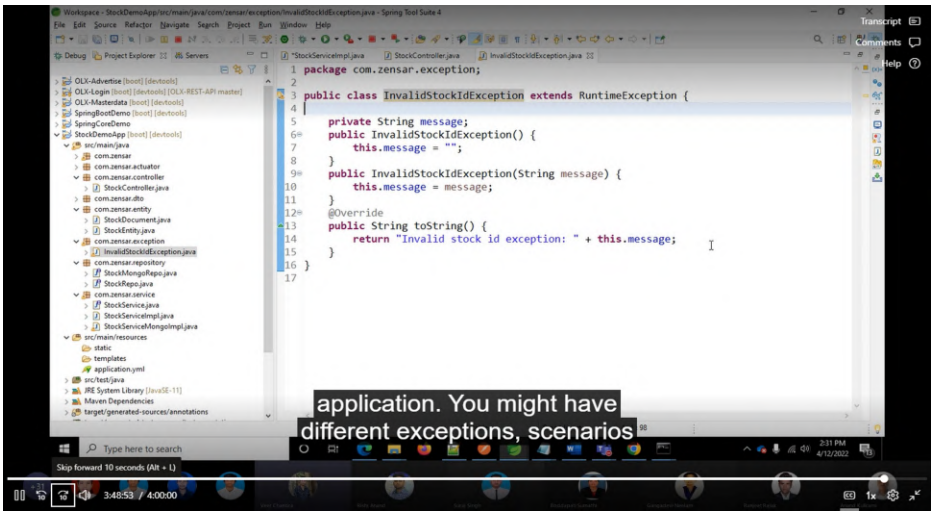


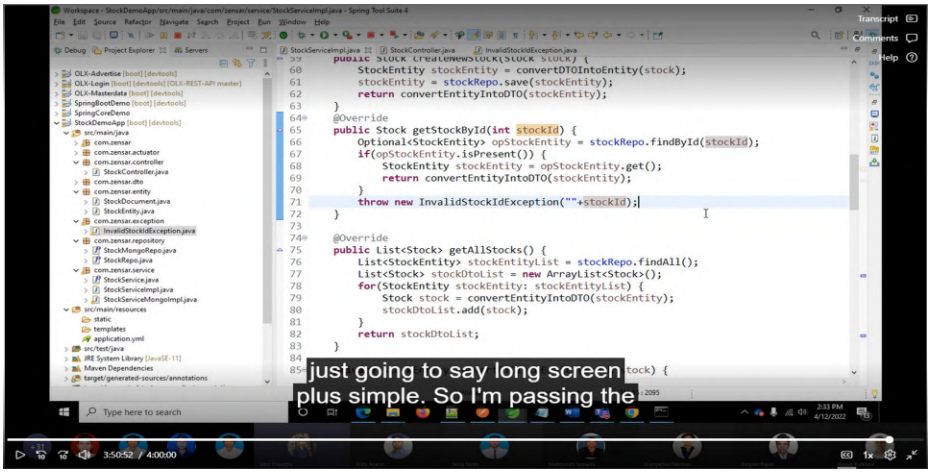
Exception Handling in java, is not only display messages to client. We actually generate exception classes. And we will throw object of classes. That is why in java every exception is mapped to corresponding java objects. So when exception arise then JVM creates the object of exception class and throws the objects so that a user can survive else terminate the application. Now we will handle exception in spring boot.

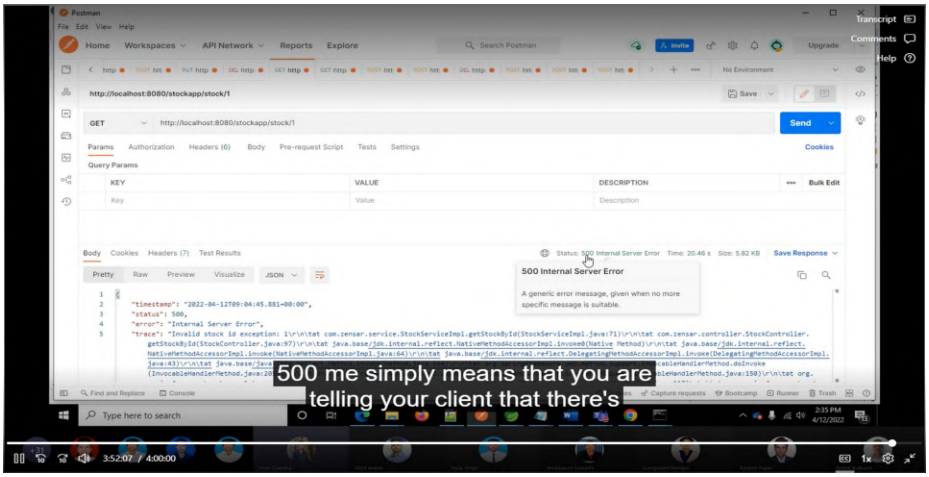
How my Client get to know it is error message or not we can handle this thing with the help of Response Status, but by default response status is 200. if we also want to control to Response status the we can also do that just like we controlled our response.

For that we will use **ResponseEntity**. It is a Combination of Response data and Response Status.


Press Esc to exit full screen
stack.push("A");
stack.push("Stack is empty");
stack.push("C");
stack.pop() = C
stack.pop() = Stack is empty
stack.pop() = A
throw new StackEmptyException();
ResponseEntity = Response + HTTP status code
mean, won't be sufficient. I
also want to control status
Transcript
Comments
Help
3:31:35 / 4:00:00

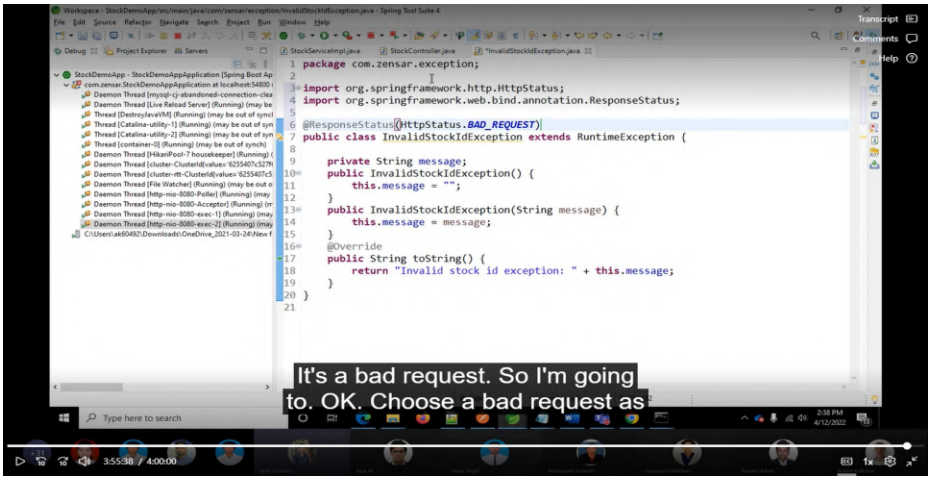

package com.zensar.exception;
public class InvalidStockIdException extends RuntimeException {
private String message;
public InvalidStockIdException() {
this.message = "";
}
public InvalidStockIdException(String message) {
this.message = message;
}
@Override
public String toString() {
return "Invalid stock id exception: " + this.message;
}
}
application. You might have
different exceptions, scenarios
Transcript
Comments
Help
Skip forward 10 seconds (Alt + L)
3:48:53 / 4:00:00


@Override
public Stock getStockById(int stockId) {
Optional<StockEntity> opStockEntity = stockRepo.findById(stockId);
if(opStockEntity.isPresent()) {
StockEntity stockEntity = opStockEntity.get();
return convertEntityIntoDTO(stockEntity);
}
throw new InvalidStockIdException(""+stockId);
}
@Override
public List<Stock> getAllStocks() {
List<StockEntity> stockEntityList = stockRepo.findAll();
List<Stock> stockDtoList = new ArrayList<Stock>();
for(StockEntity stockEntity: stockEntityList) {
Stock stock = convertEntityIntoDTO(stockEntity);
stockDtoList.add(stock);
}
return stockDtoList;
}
just going to say long screen
plus simple. So I'm passing the
Transcript
Comments
Help
3:50:52 / 4:00:00

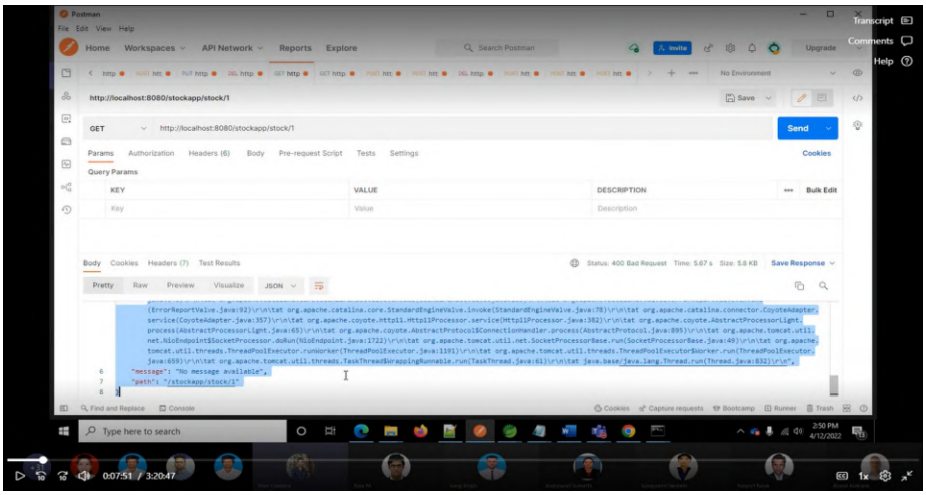


First Client will not understood what kind of error message he is getting and also getting big chunk of data.
Two Problem we are facing over here.

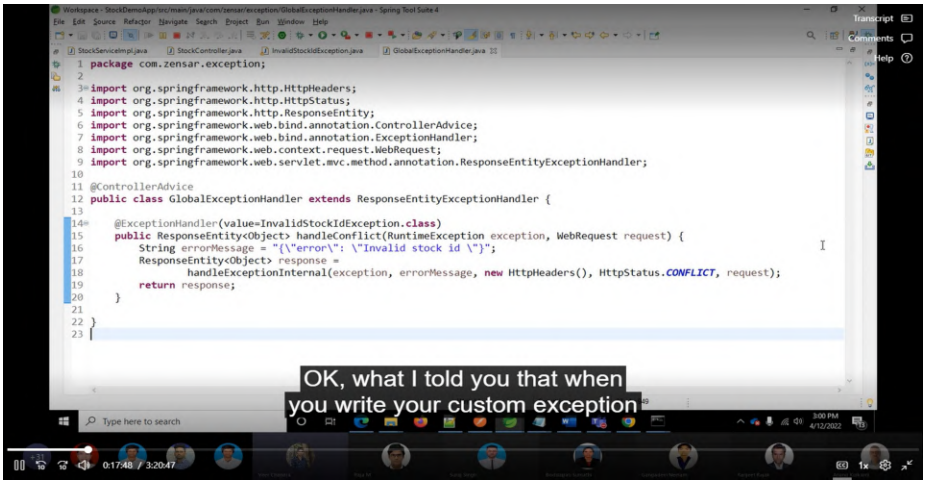
(i) Status Code with 500 its too generic error and it means something wrong happened server side. To override this thing we have to do.

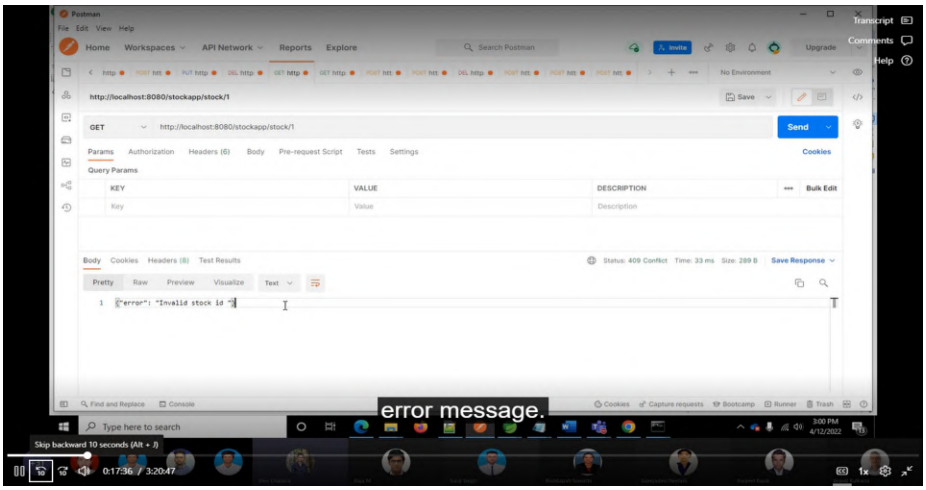


Bad request means wrong request made by client side. Meaning full status. But getting same big message not understandable.



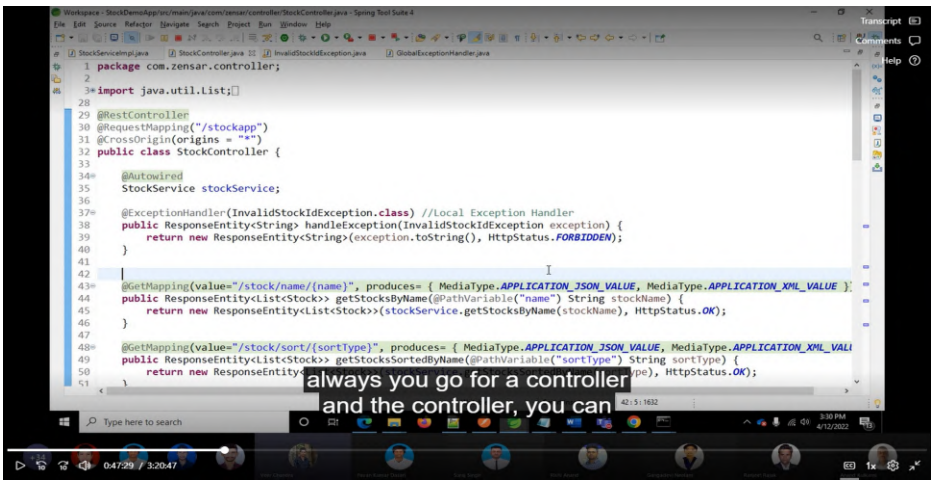
(ii) Getting big error message can be controlled using Global Exception class which will handle the message.

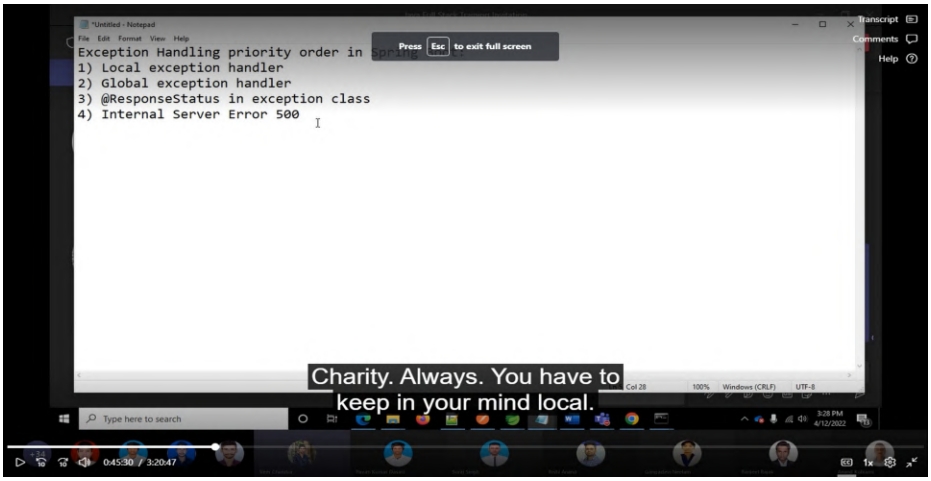




Use of Global Handler, we can have dozens of custom exception handler classes. And we can easily able to handle classes' response status and response body. It is central class which handles all the classes easily.

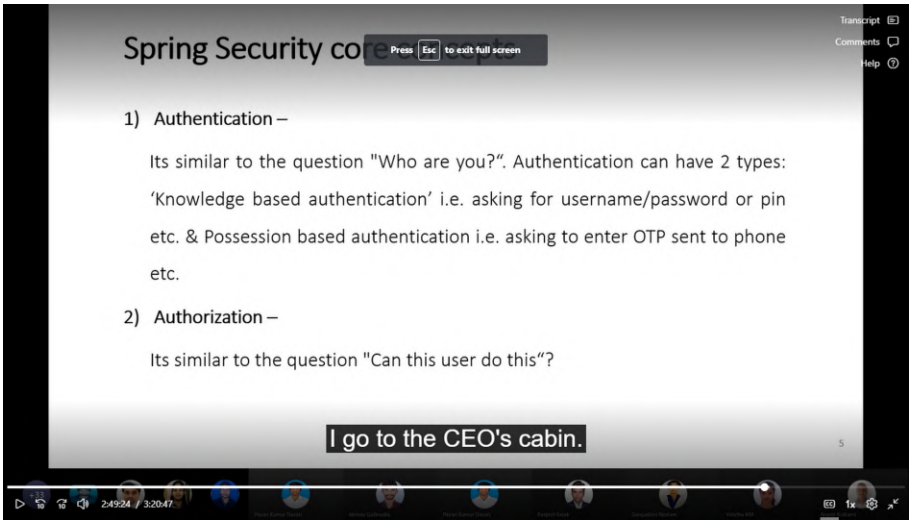
@ControllerAdvice annotation allows the handling of exceptions across the application.
Before this, Spring offered another annotation `@ExceptionHandler` for exception handling. But, you have to add this annotation in each controller class of your application. It doesn't help on the application level.

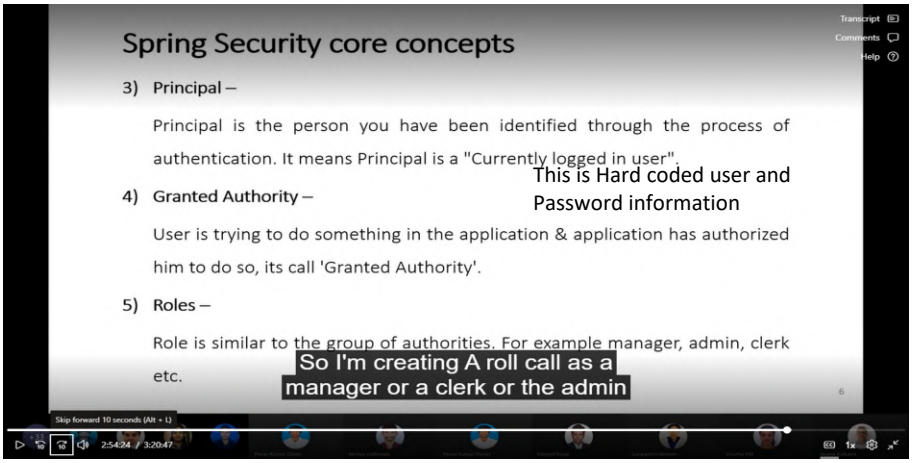




Spring Security

SSO can be used by enterprises, smaller organizations and individuals to ease the management of various usernames and passwords.





This is Hard coded user and Password information

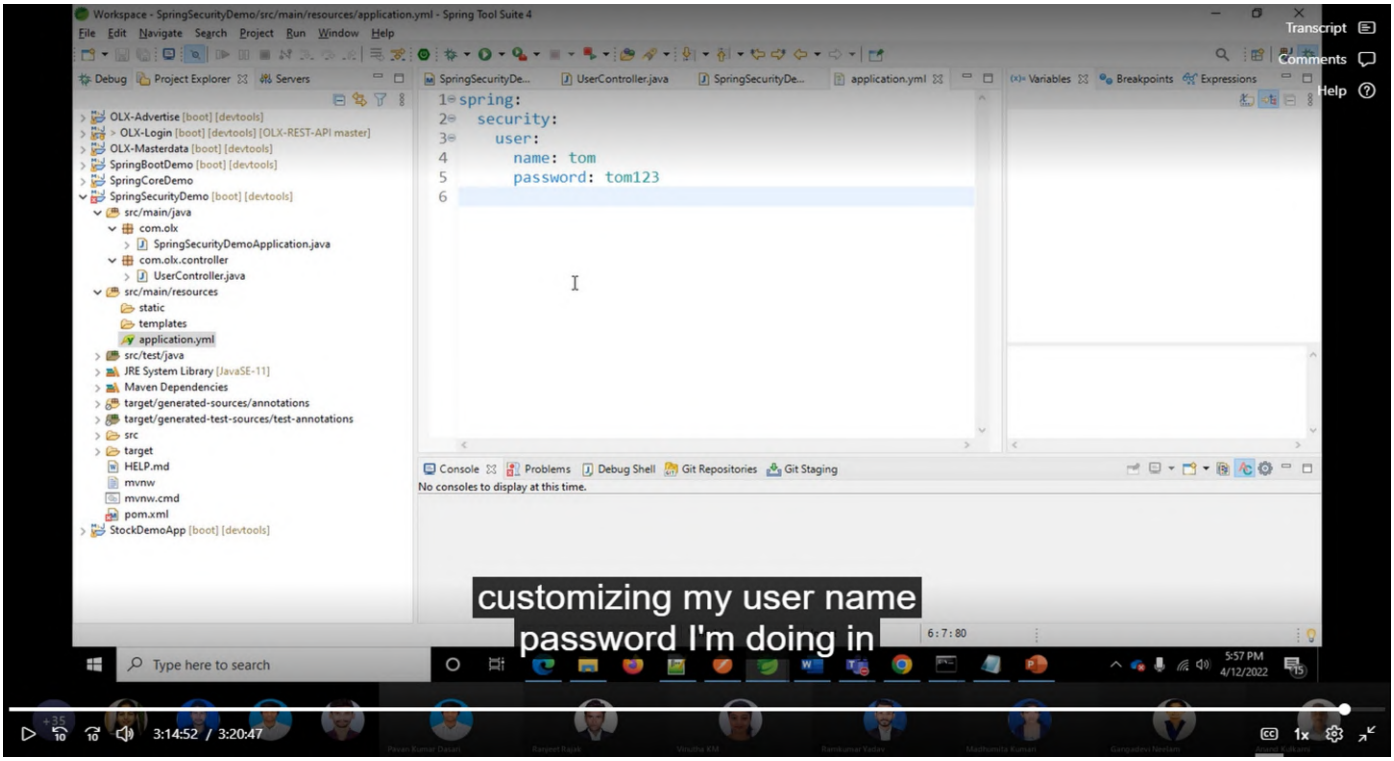
```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

After adding this dependencies our application act as a security manager.

3 Ways to add Security into application.

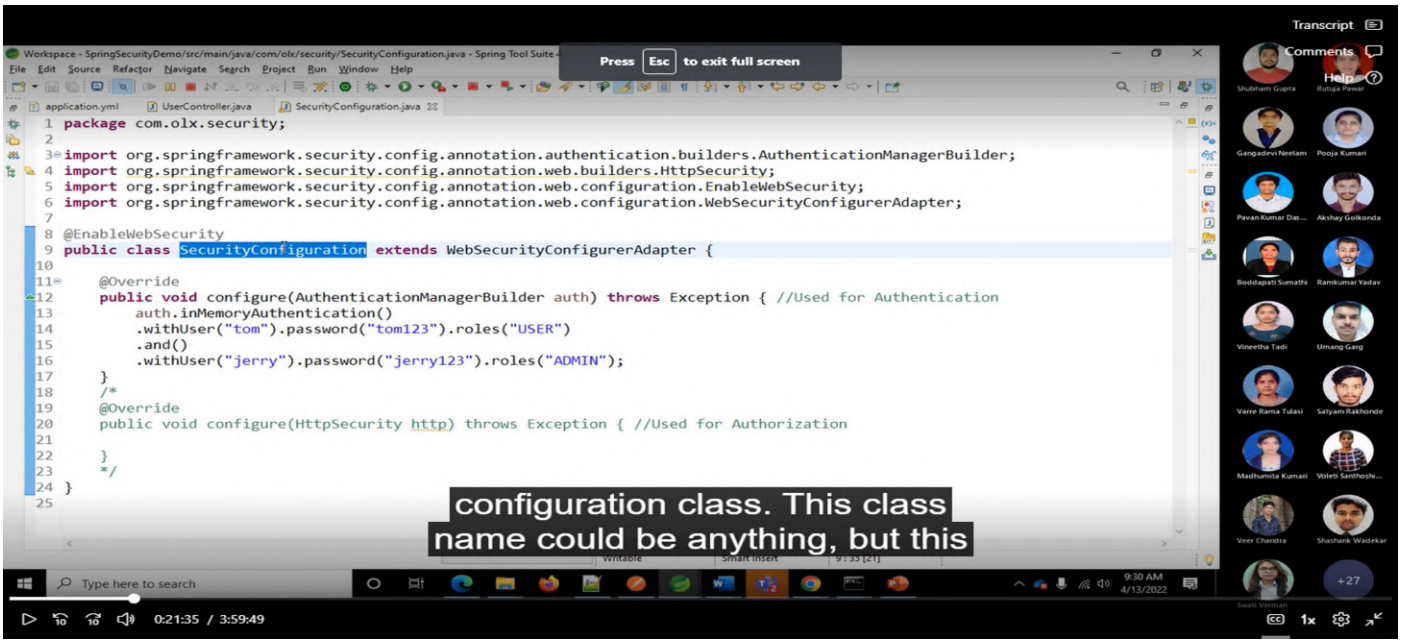
(i) Security added by default by Spring Security

(ii) Adding Configurations into application.yml file



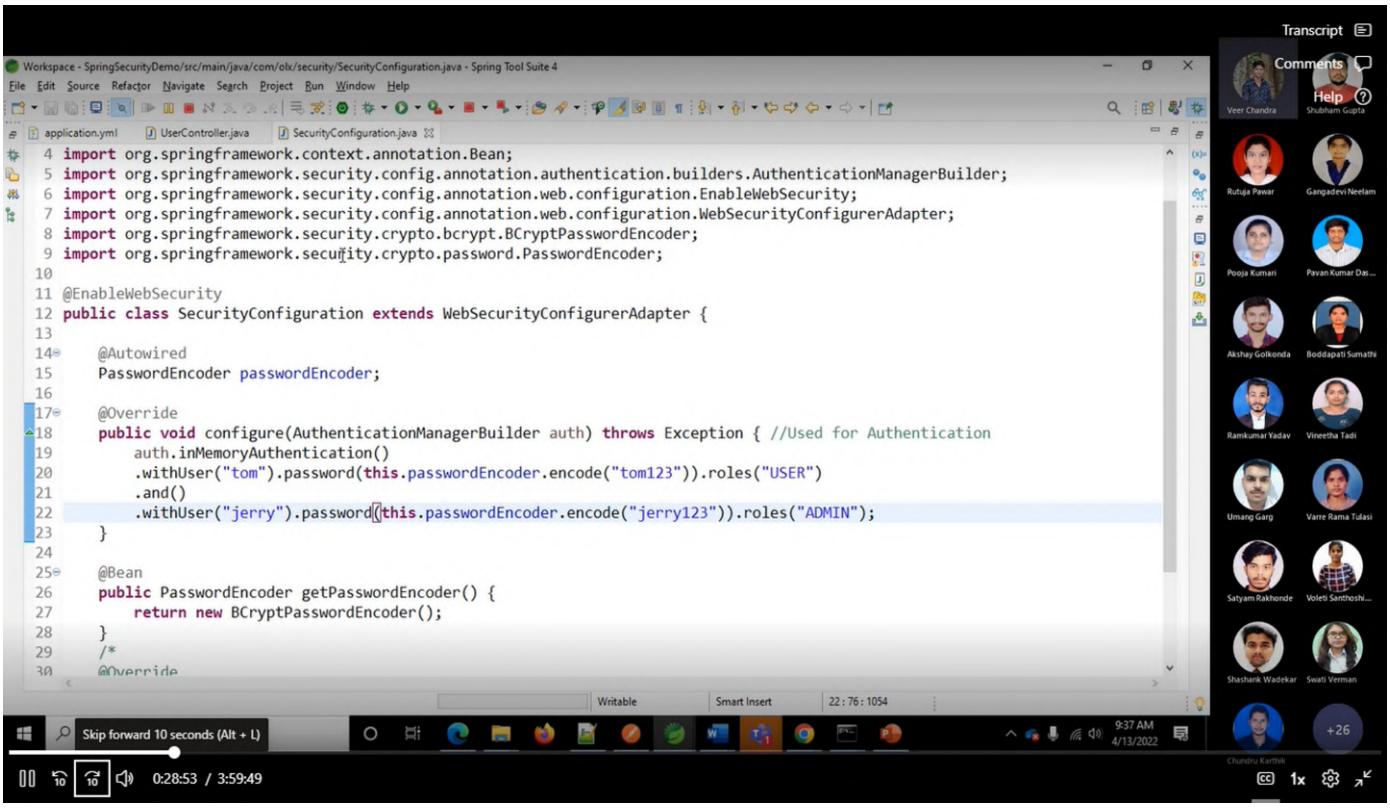
(iii) Creating the user and password in memory side

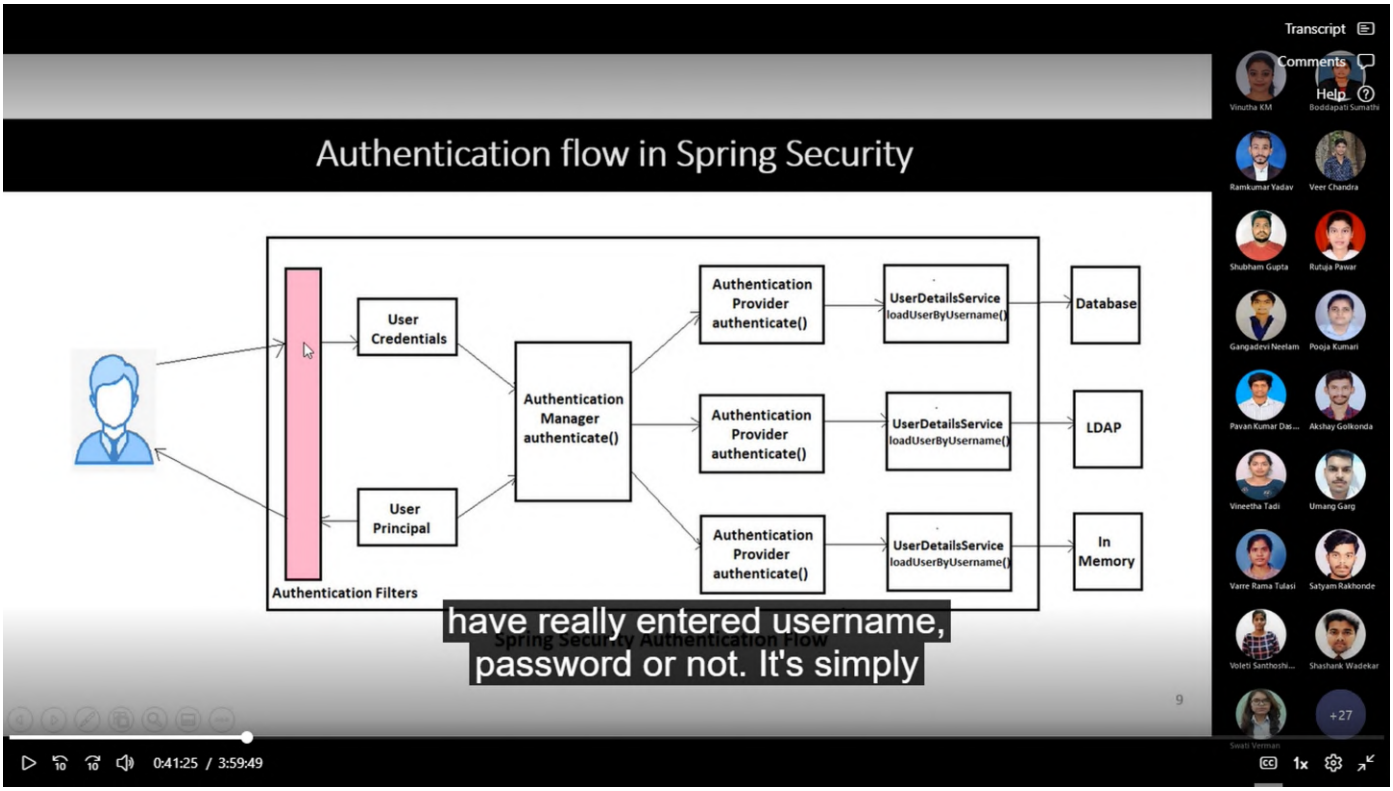
Because we can have n number of users with password so that will be good if we go for storing data in memory.



The entire configuration of a security for a application is configured inside in this class.

Security framework is a secure framework. It says that if you creating n memory user then you cannot keep the password in the memory in open format. We must have to encode password then only we are able to allowed to create memory for users. To do that we have to use Password Encoder.





Authentication Filters is used to check did you entered username and password or not.

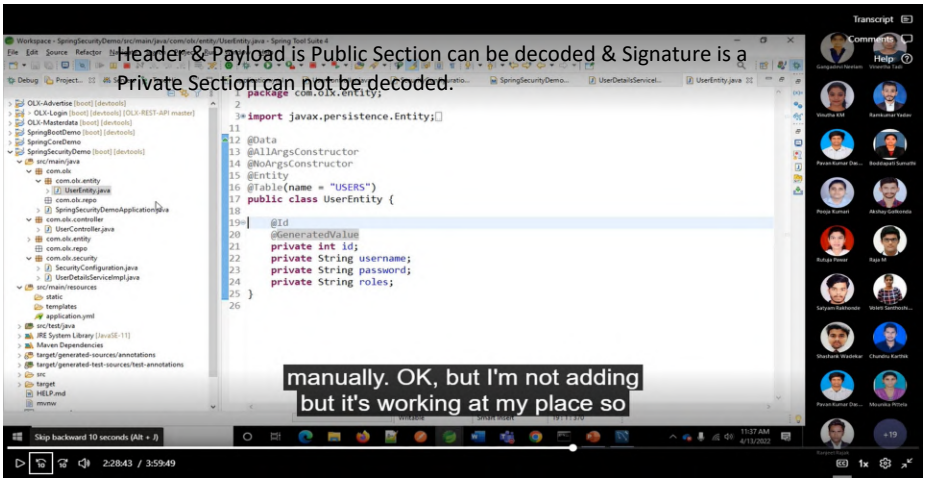
Authentication Manager is a heart of authentication mechanism in the spring security framework. When we want to authenticate ourselves we have to call authenticate function on the authenticate manager. But authenticate function of Authenticate manager. It will not write authentication code. It internally going to call authentication method inside of authentication provider object which is a class given by spring security. The authentication provider method will say implements business logic of authentication clients. The authentication provider will not interact with database directly. It will simply calls loaduserbyusername function on UserDetailsService. Authentication provider simply pass Username to loadUserByUsername() so loadUserByUsername() talk to the database & ask to get
Information about user from database. Then loadUserByUsername() return data to Authentication provider & Authentication provider just check is password matching from returned information of user with user's password. If matched return true value to authentication

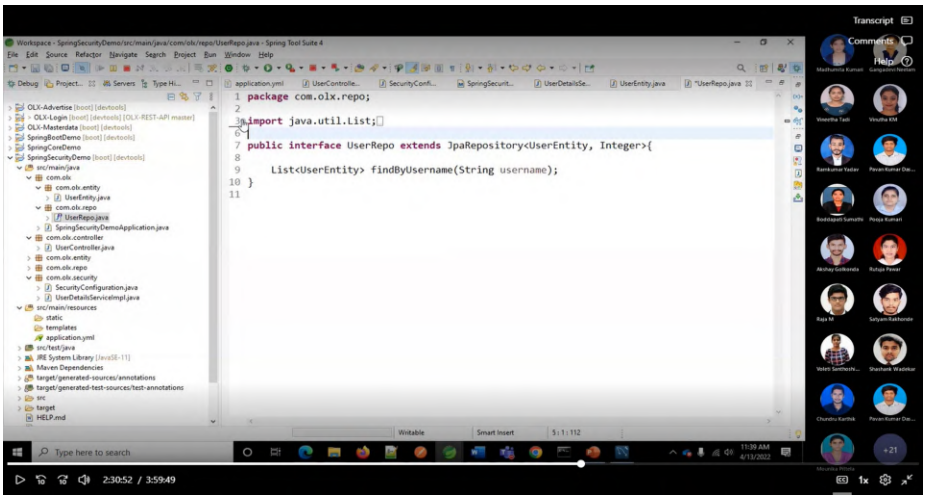
manager. After that authentication manager creates object of user's full information as a principal object.

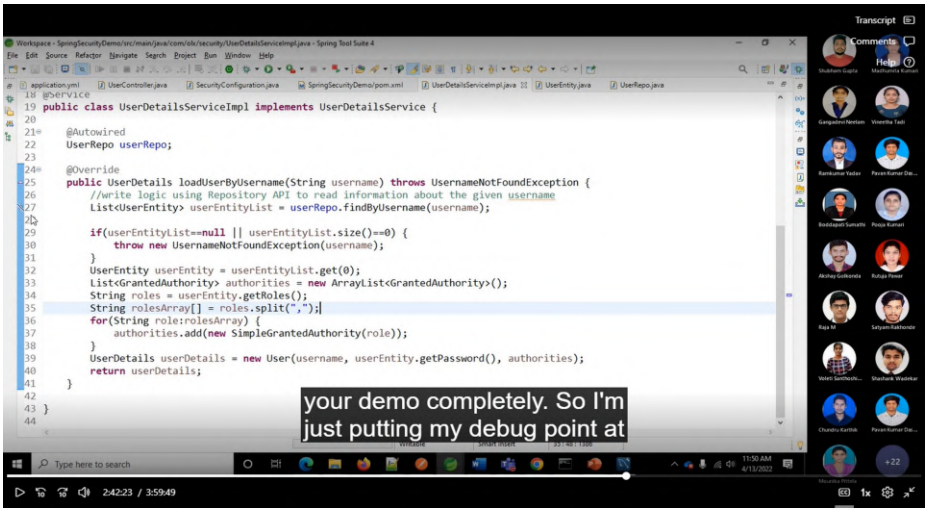
Spring Dependencies ::::

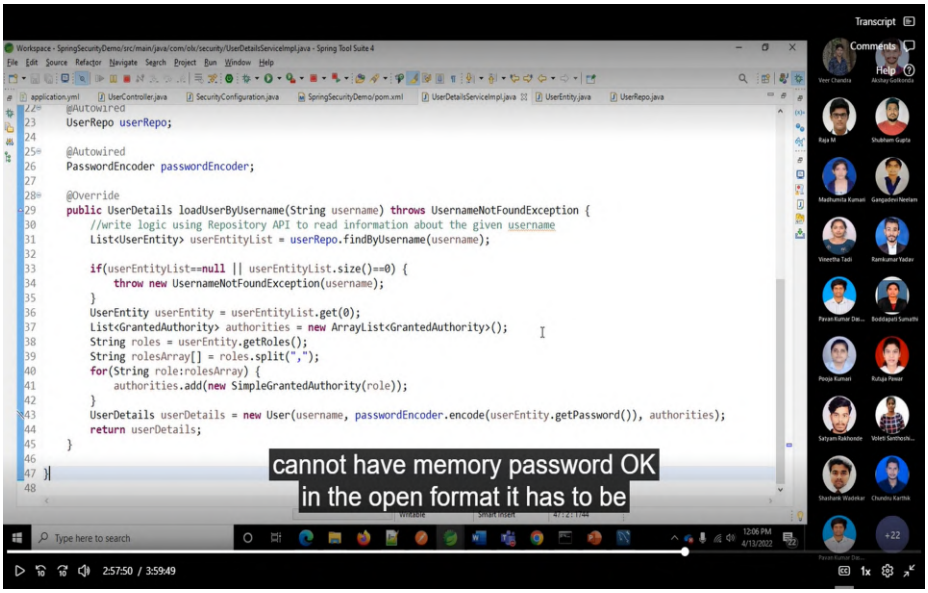
```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Public section

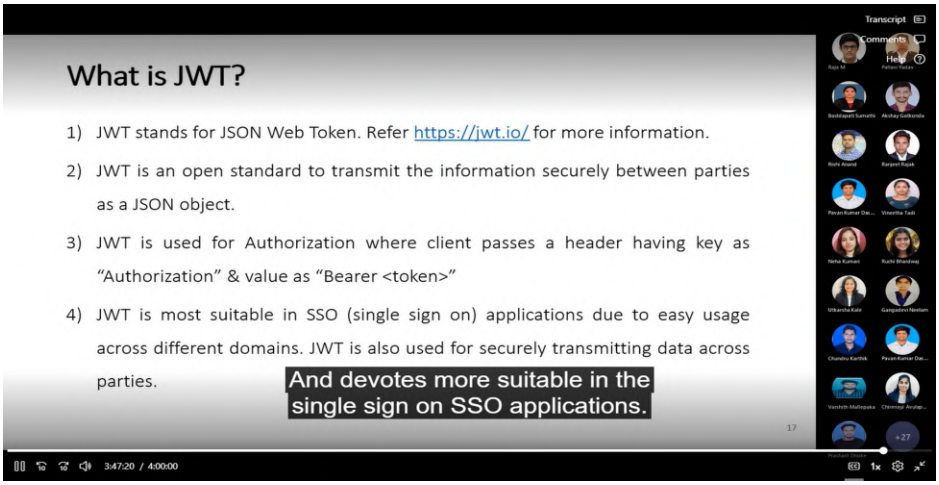


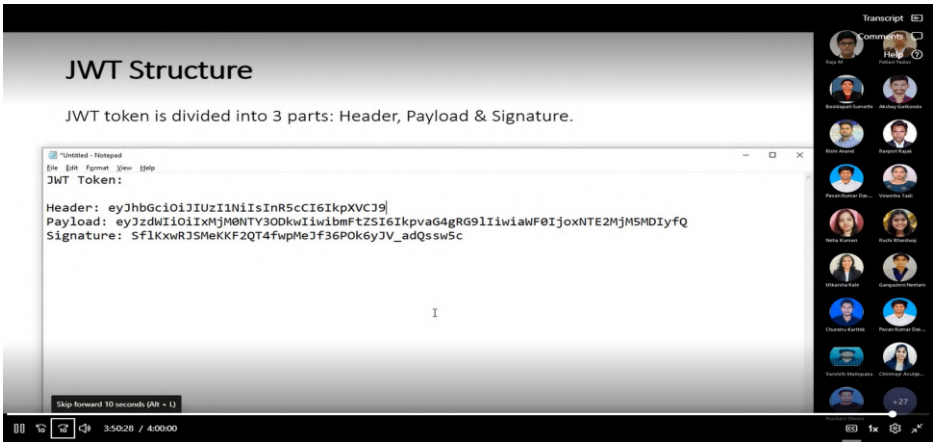


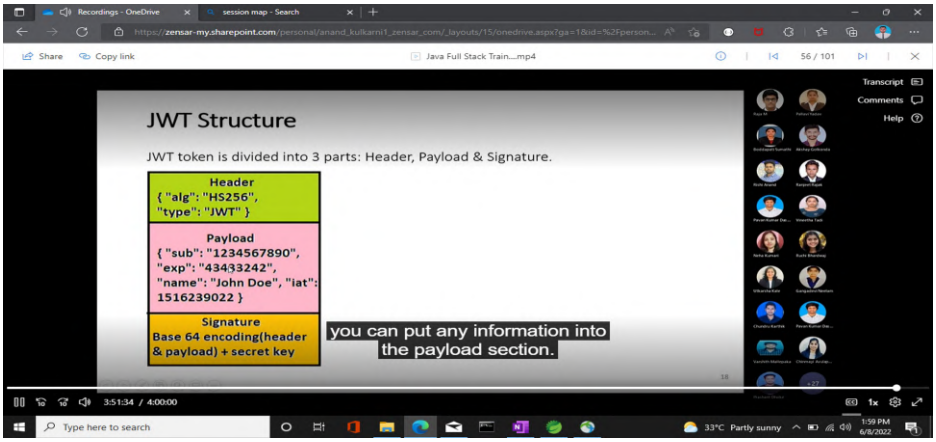




Session Map is a architecture for creating session for user with the combination of session id and state of client. Good for creating limited number of user's session, consuming memory and at some point of time it will full your heap memory. It might crash your system as well. So for heavy applications we go for the solution where I can maintain the session of the user without having session map concept. That's why we go for JWT. In JWT, we nothing need to maintain on server side.

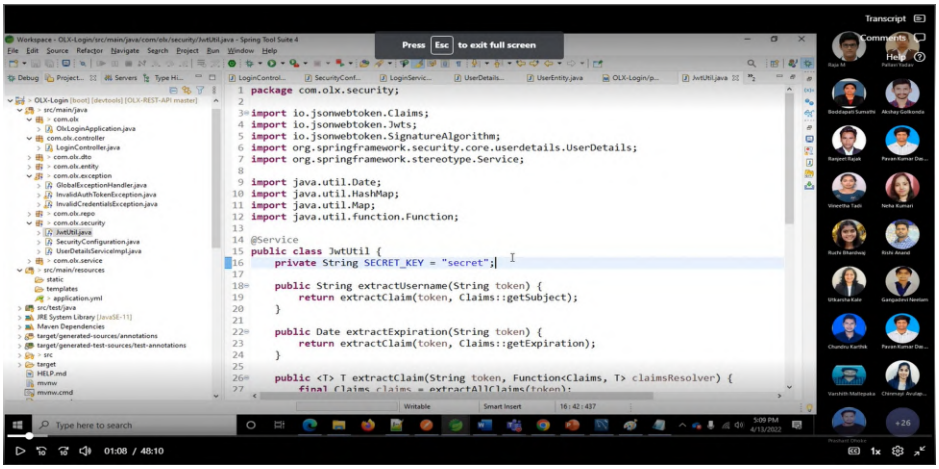


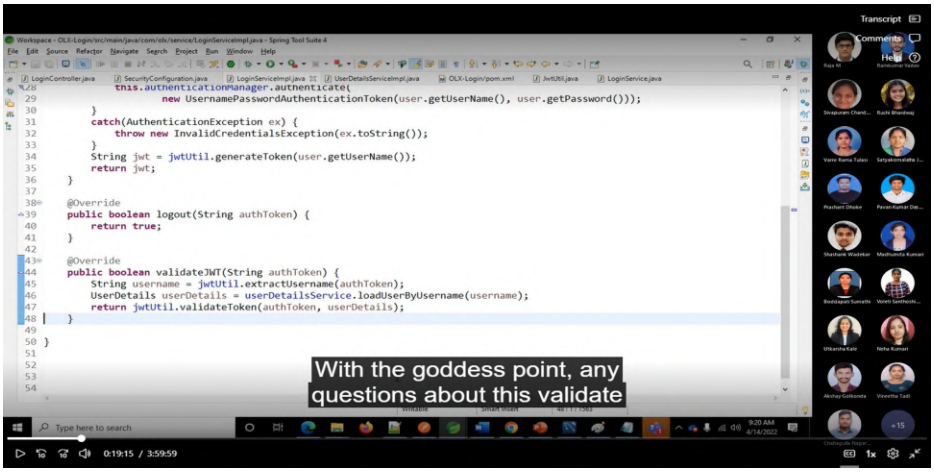




But to use JWT inside application we will use JJWT. So for Java applications if you want to create JSON web token in that case we need to add third party dependency inside pom.xml file. With this third party library we are able to generate, check expiry time etc. many operations. And this API also provide a utility class.

[JFS_14_Mar_2022/JwtUtil.java at main · anand-mentor/JFS_14_Mar_2022 (github.com)]



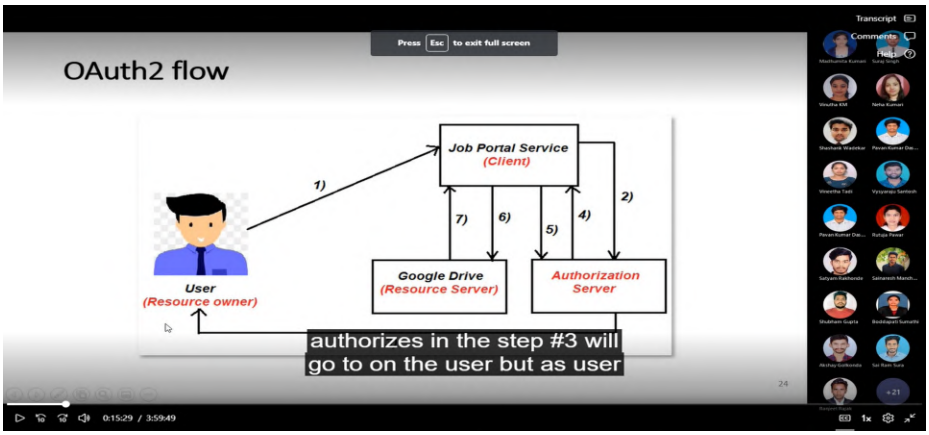


Every JSON web token must started with Bearer word. This is guide lines we must have to follow.

All Servers are stateless. It means a clients calls for 100 times every calls to the service it treats a client as a new client. The service will not recognize you because all servers for the HTTP protocols are stateless. So they will not recognize you. So that's why after login every request Auth token must validate with request to check is that a user try to access some information is a valid user or not.

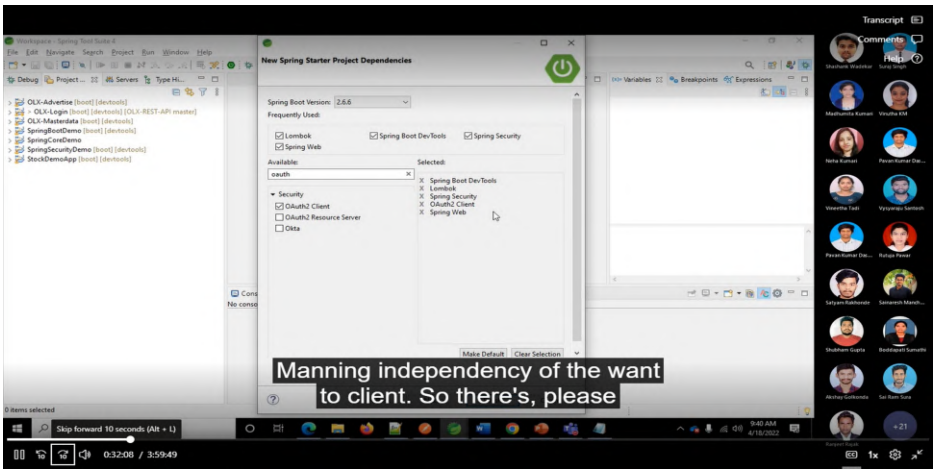
Whenever we are trying to make communication between two microservices is called inter communication. To make communication between 2 services we use interface called delegate. Inside delegate service we will use very important object over here is called as the rest template.

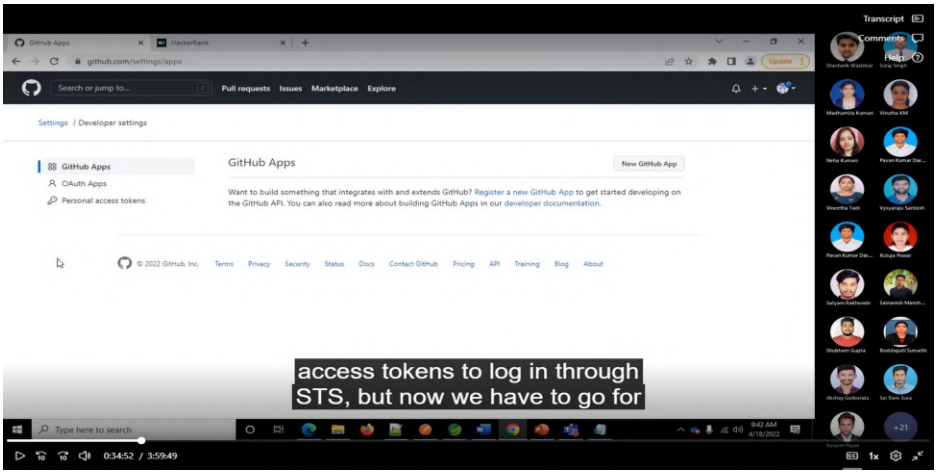
OAUTH2 is a ultimately establishing communication between two applications.

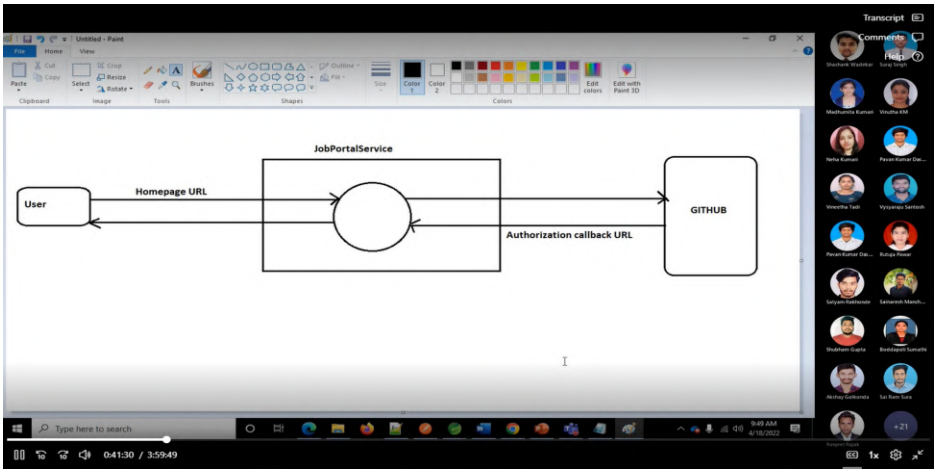


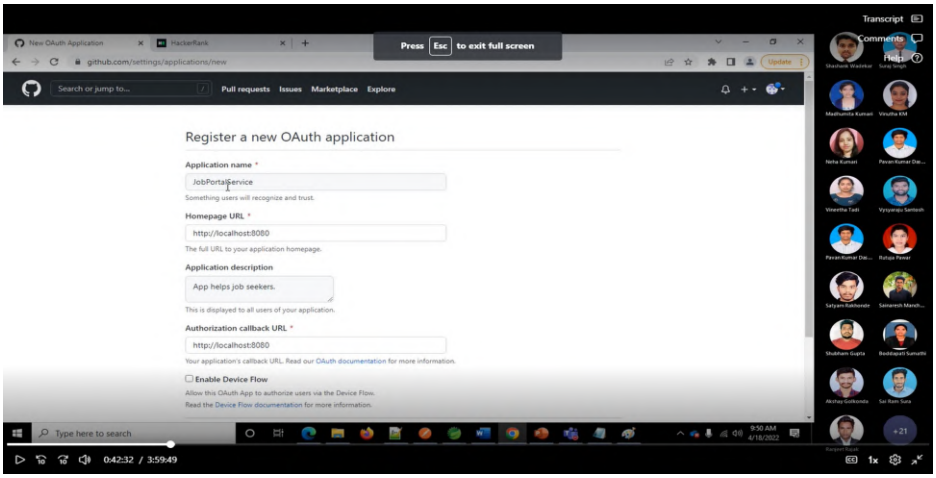
2 step Client will say to access user's data but authorization server will say that I don't know you so first I have to ask user that you authorized or not so in 3 step authorization server ask to user about client if user confirmed then In 4 step a authentication token generated and given to client. In 5 step again token send back to client come back with authentication token and authorization server generate a access token. The access token send to google drive in step 6 google drive check is that a access token is valid or not. If it is valid then Google Drive send back information of user to Client.

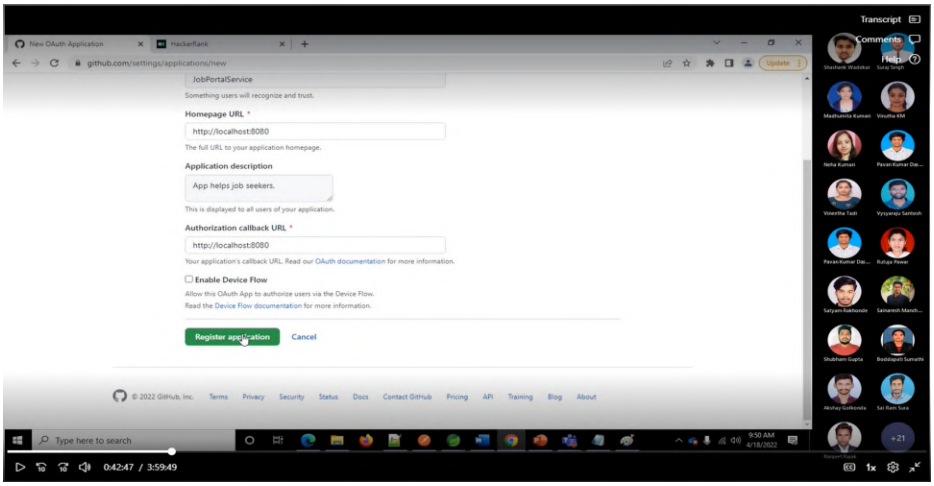
Authorization Server responsible to generate Authentication token and access token.

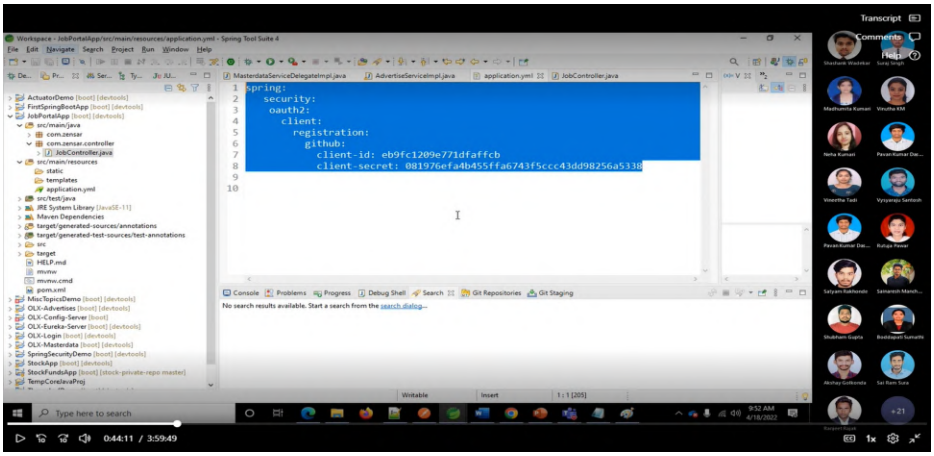


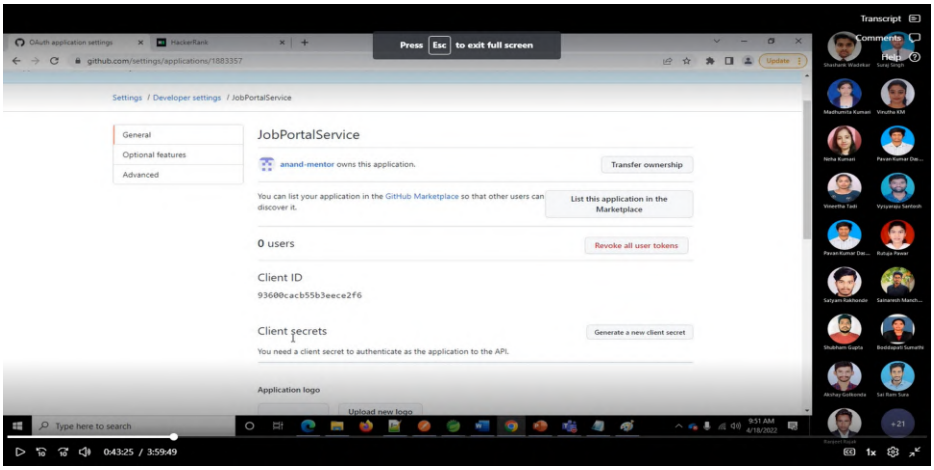


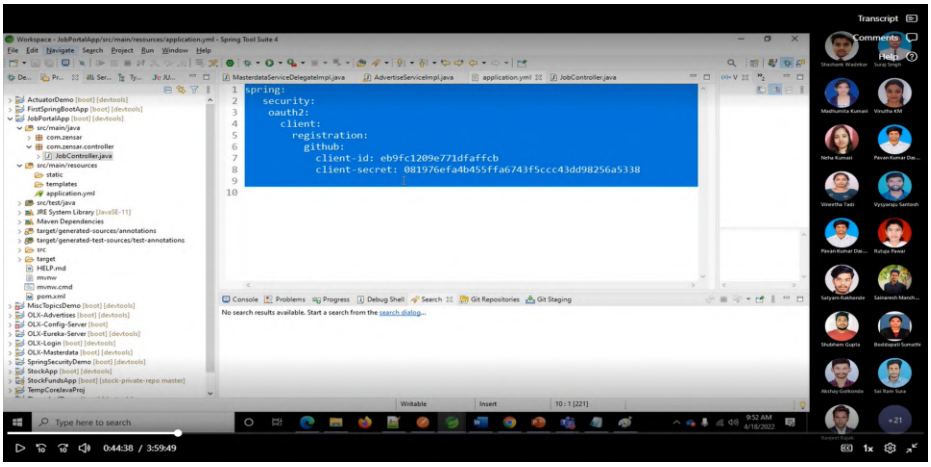

JobPortalService
User
Homepage URL
GITHUB
Authorization callback URL
Transcript
Comments
0:41:30 / 3:59:49

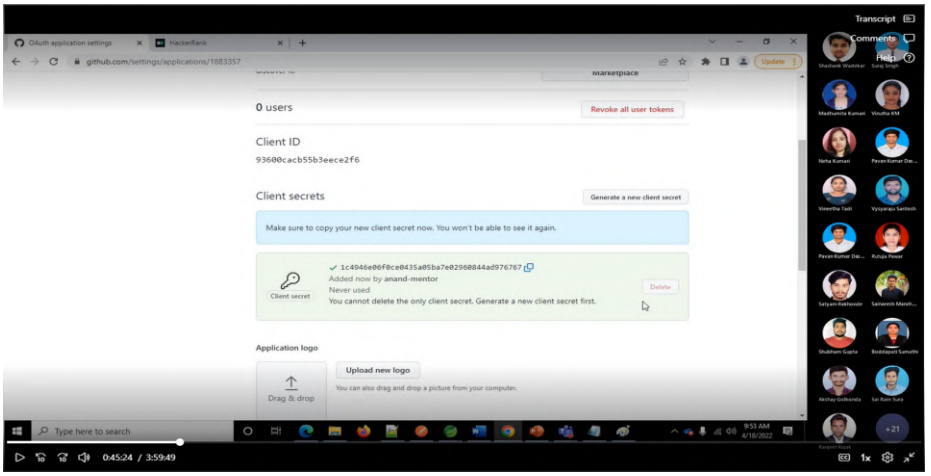

Press Esc to exit full screen
Register a new OAuth application
Application name *
JobPortalService
Something users will recognize and trust.
Homepage URL *
http://localhost:8080
The full URL to your application homepage.
Application description
App helps job seekers.
This is displayed to all users of your application.
Authorization callback URL *
http://localhost:8080
Enable Device Flow
Transcript
Comments
0:42:32 / 3:59:49

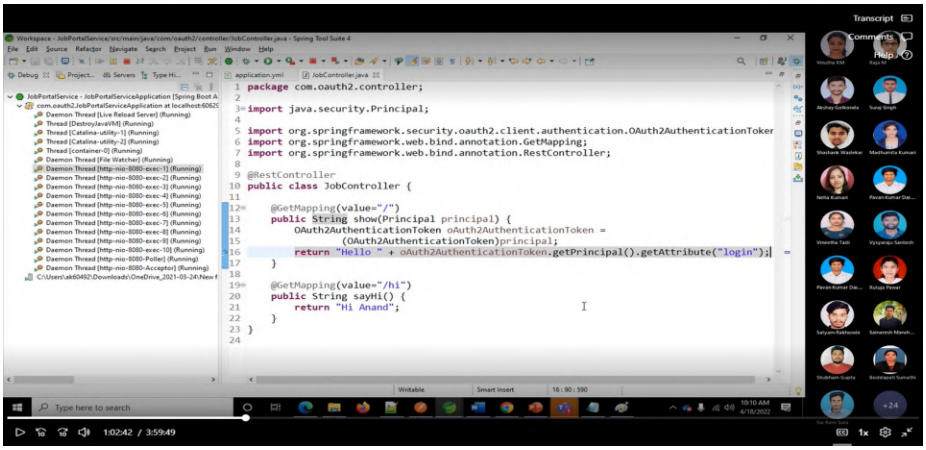

JobPortalService
Homepage URL *
http://localhost:8080
Application description
App helps job seekers.
Authorization callback URL *
http://localhost:8080
Enable Device Flow
Register application
Cancel
Transcript
Comments
0:42:47 / 3:59:49


spring:
security:
oauth2:
client:
registration:
github:
client-id: eb9fc1209e771dfaffcb
client-secret: 081976efa4b455ffa6743f5ccc43dd98256a5338
0:44:11 / 3:59:49

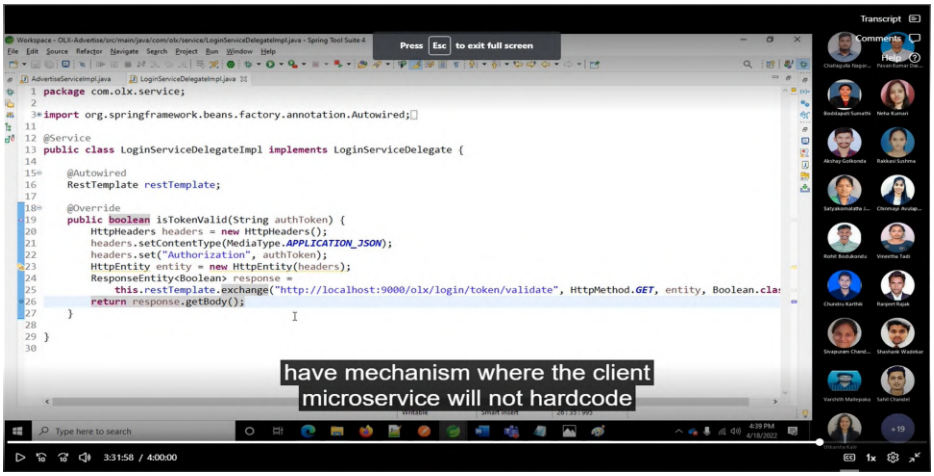

Press Esc to exit full screen
Settings / Developer settings / JobPortalService
JobPortalService
anand-mentor owns this application.
Transfer ownership
0 users
Revoke all user tokens
Client ID
93600cacb55b3eece2f6
Client secrets
Generate a new client secret
Application logo
0:43:25 / 3:59:49


spring:
security:
oauth2:
client:
registration:
github:
client-id: eb9fc1209e771dfaffcb
client-secret: 081976efa4b455ffa6743f5ccc43dd98256a5338
0:44:38 / 3:59:49


0 users
Revoke all user tokens
Client ID
93600cacb55b3eece2f6
Client secrets
Generate a new client secret
Make sure to copy your new client secret now. You won't be able to see it again.
Client secret
Added now by anand-mentor
Never used
You cannot delete the only client secret. Generate a new client secret first.
Delete
Application logo
Upload new logo
Drag & drop
0:45:24 / 3:59:49



Inter-communication if you want to create to connection between two microservices. Then we use a interface which is called delegate. And in delegate we using rest template object. Using rest template connection is possible. Foreign key concept is possible only with if we want to make communication between 2 tables in same database. Between intercommunication comes into a picture when we want to create communication between different microservices with different tables in different database.



Problems with this

1. Here we using hardcoded port number.
2. Multiple instance possible of a microservice and to allocate a specific microservice to another Required someone who's helps to find out which microservice is up and running.

Eureka Server is an application that holds the information about all client-service applications. Every Micro service will register into the Eureka server and Eureka server knows all the client applications running on each port and IP address. Eureka Server is also known as Discovery Server. Eureka Server helps to find out which microservice is up and running and provide a specific service to another service who's required on that. The use of eureka server to get request from client service and forward it to a destination microservice. That's it.

Eureka Server comes with the bundle of Spring Cloud. For this, we need to develop the Eureka server and run it on the default port 8761.

After downloading the project in main Spring Boot Application class file, we need to add @EnableEurekaServer annotation. The @EnableEurekaServer annotation is used to make your
Spring Boot application acts as a Eureka Server.

Make sure Spring cloud Eureka server dependency is added in your build configuration file. Spring cloud implements Eureka Server.

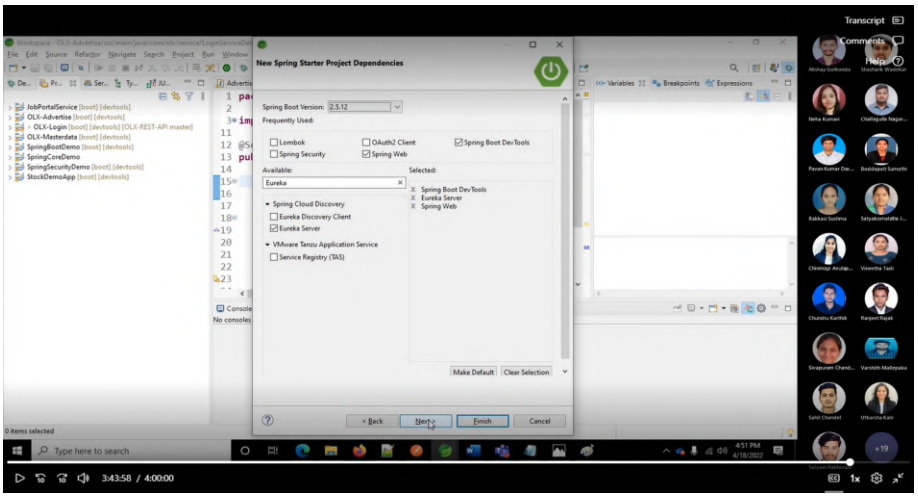
The code for Maven user dependency is shown below –

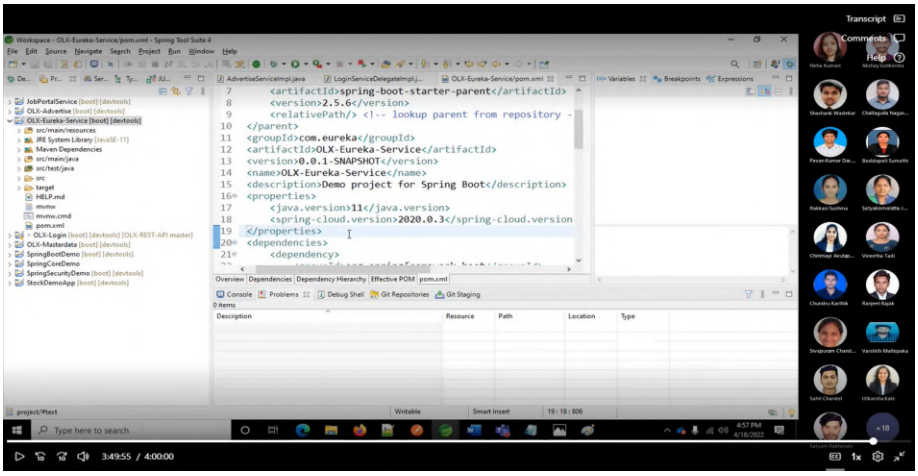
```
<dependency>
<groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-eureka-server</artifactId>
</dependency>
```

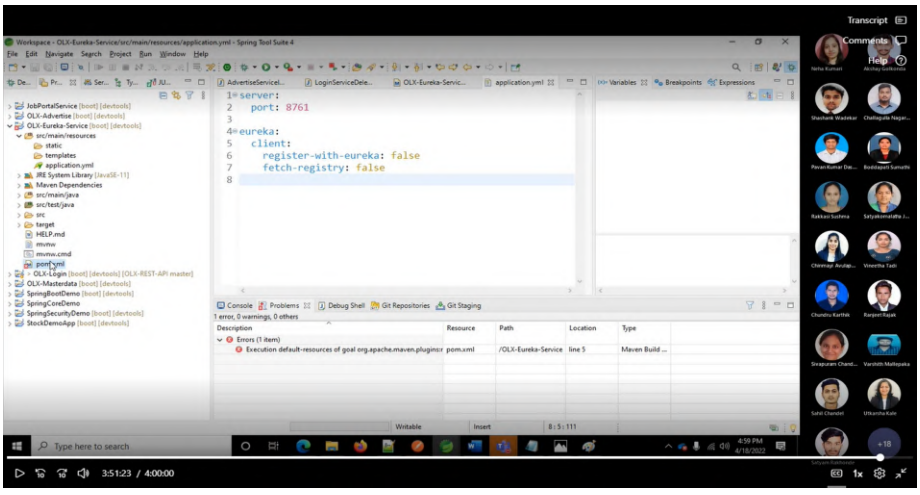
By default, the Eureka Server registers itself into the discovery. You should add the below given configuration into your application.properties file or application.yml file.

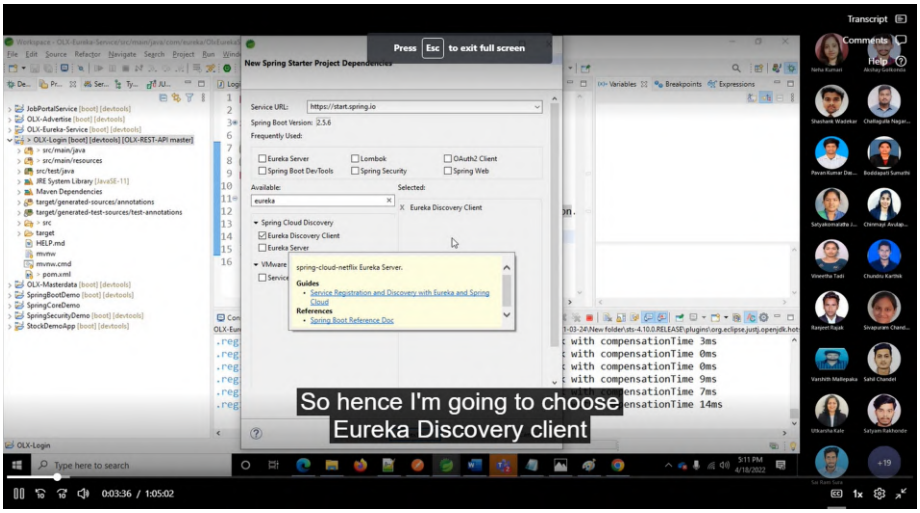
application.properties file is given below −

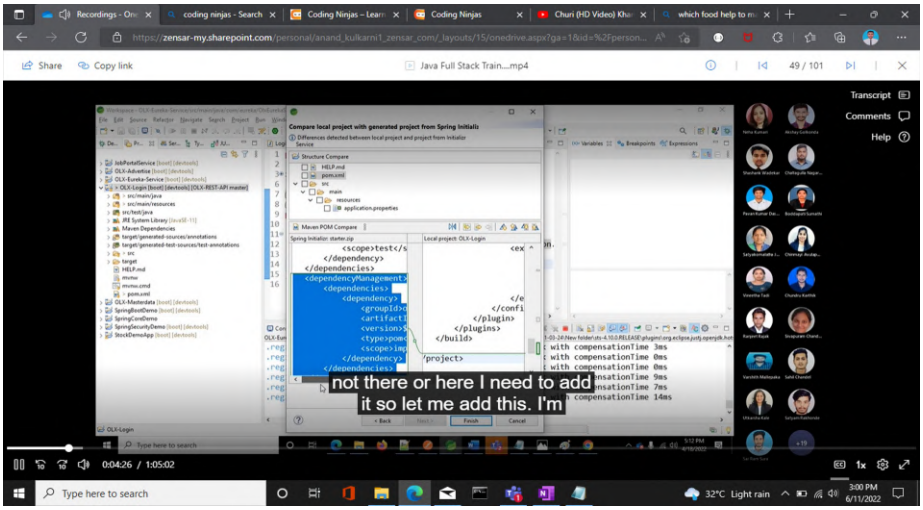
```
eureka.client.registerWithEureka = false
eureka.client.fetchRegistry = false
server.port = 8761
```

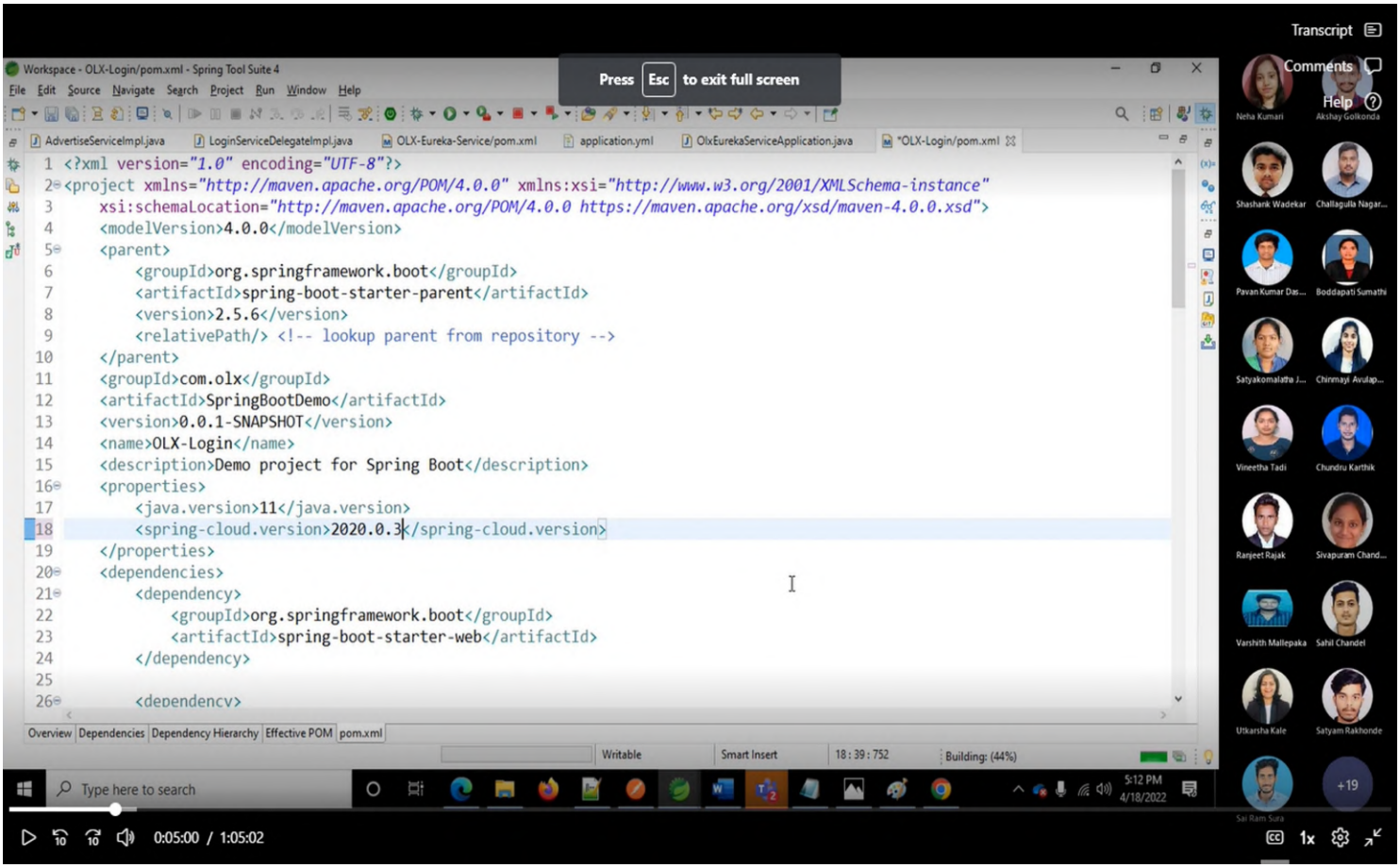


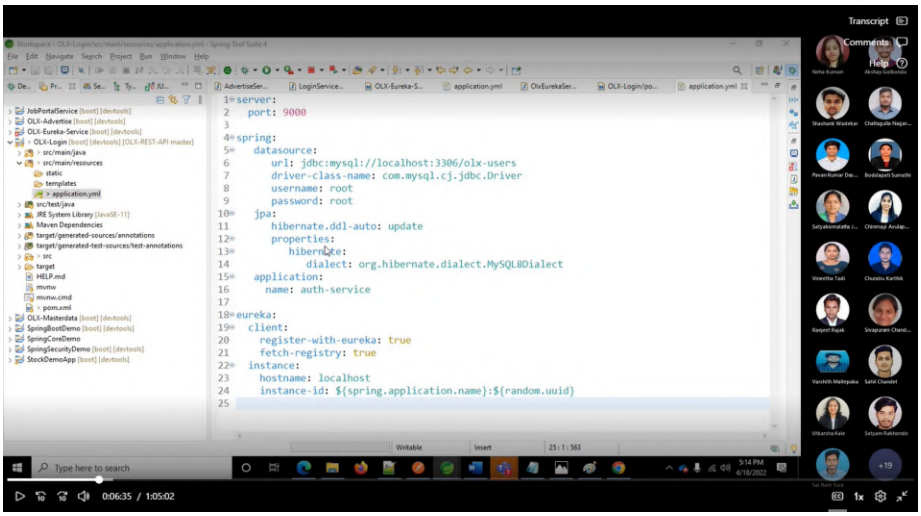




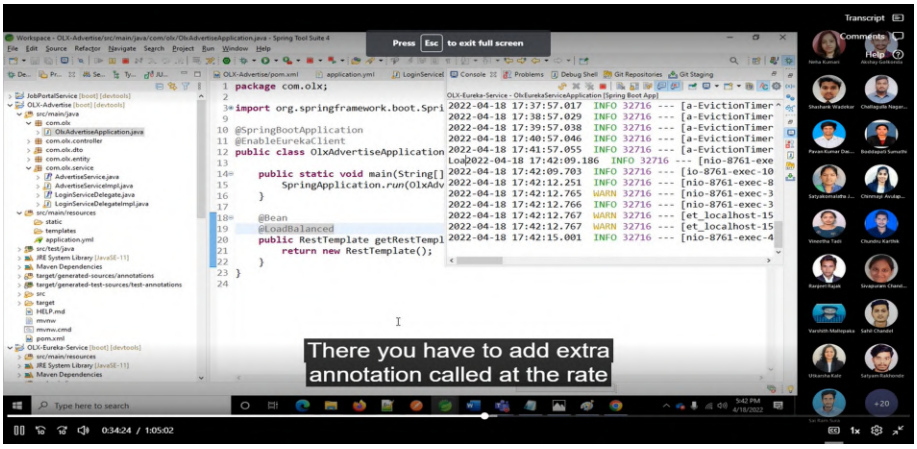

Press Esc to exit full screen
So hence I'm going to choose
Eureka Discovery client
0:03:36 / 1:05:02


not there or here I need to add
it so let me add this. I'm
0:04:26 / 1:05:02

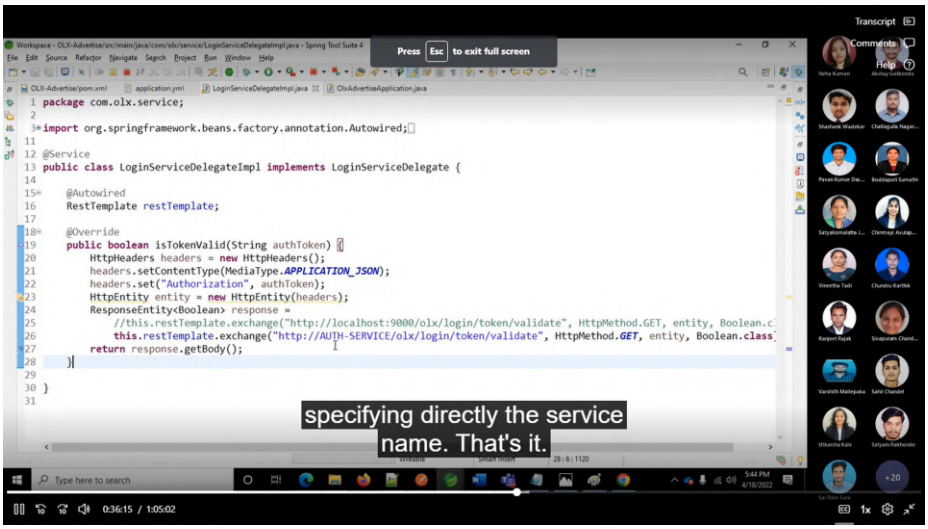




Instance information is important might be possible that I developed multiple instances of any service. This provide some more information to a specific service all the time.



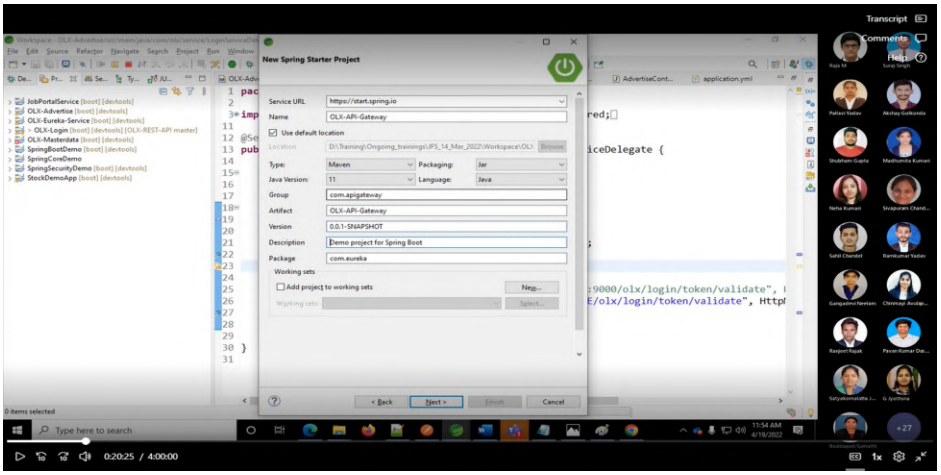
@LoadBalanced used to helps if we have multiple instance of services and if we want to locate specific service instance out of it and whenever you want rest template to call microservices through eureka service. In that case, rest template should be smarter than the traditional one. And now call will made through eureka server. We can tell using Load Balanced.

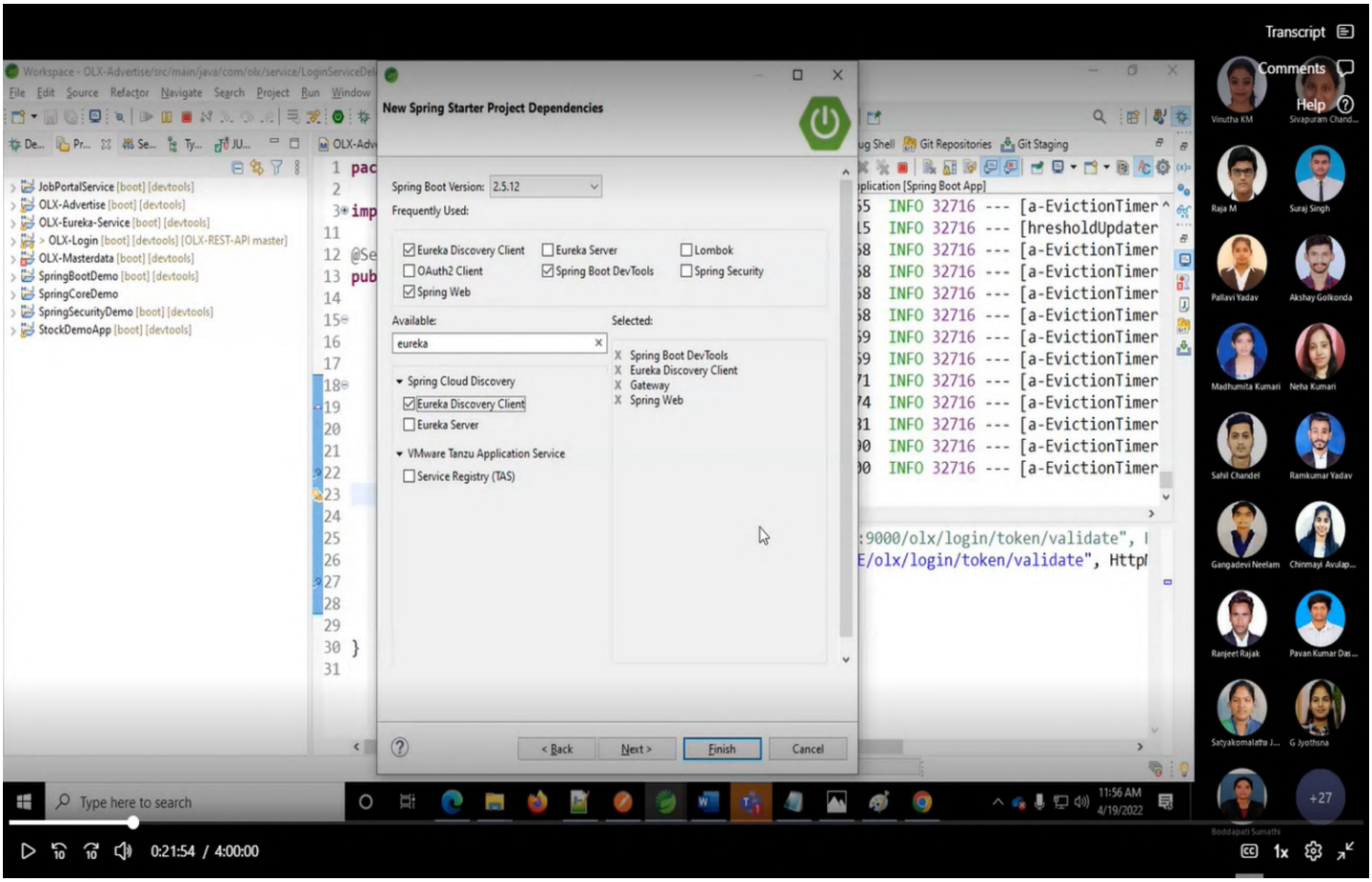


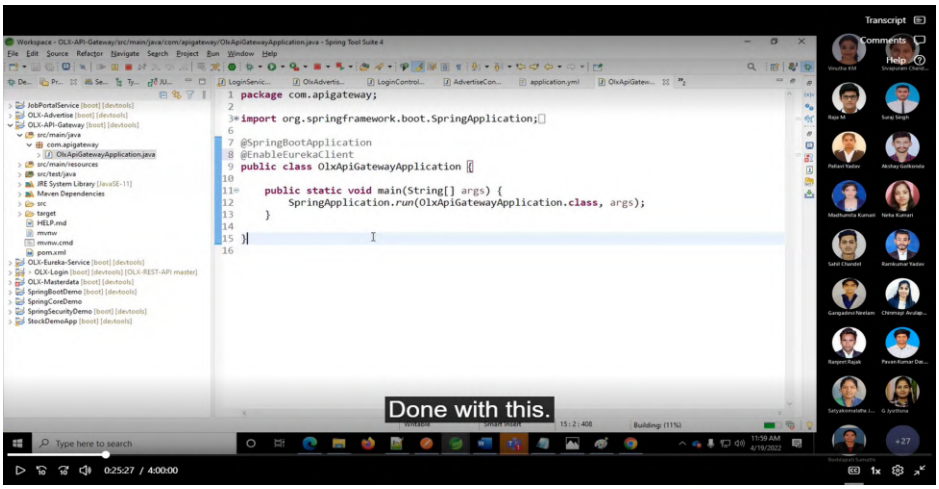
Limitations

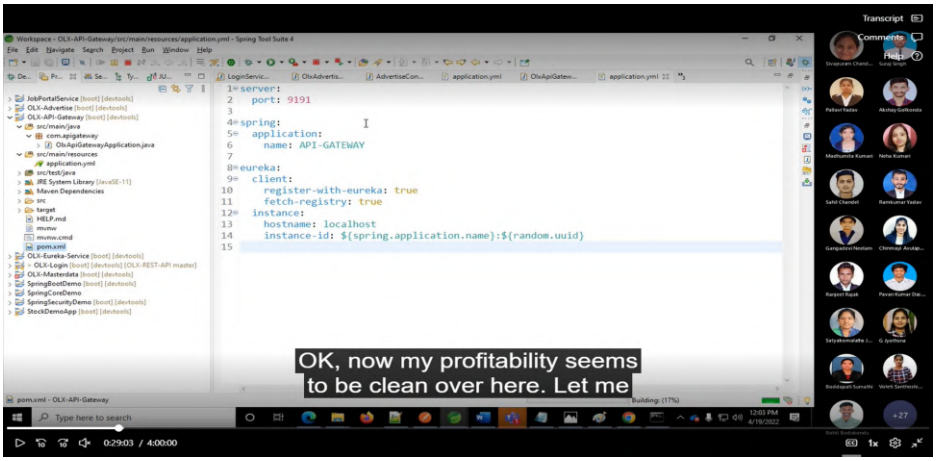
1. Hard Coded Service in future if changed application name then we have to do some changes into some microservices URL.
2. For External Client it is impossible to call Eureka because Client can't use name of service. Client will use IP Address or Port number to locate Eureka Server Clients.

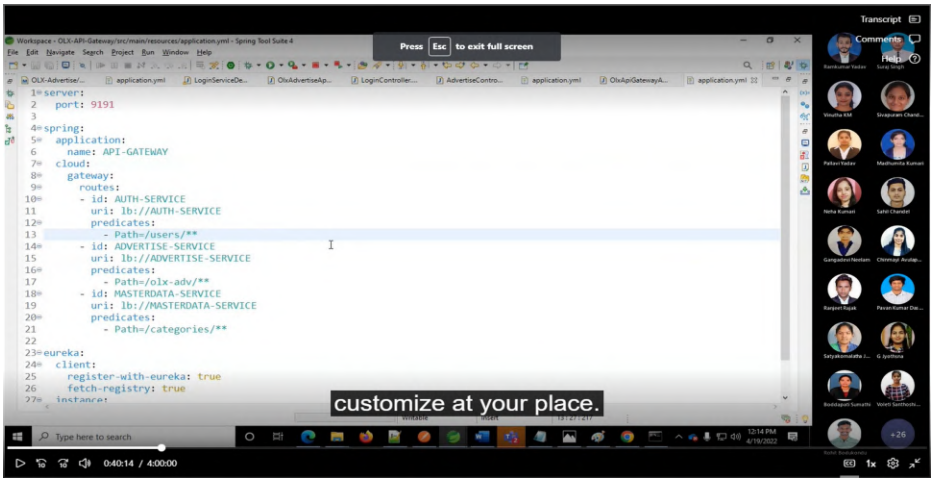
API Gateway is used to make common entry point for services call does not matter if we are calling from external or internal the microservices in the application. And resolve all the limitations of eureka server. It is also kind of microservice but act as a client of Eureka server.



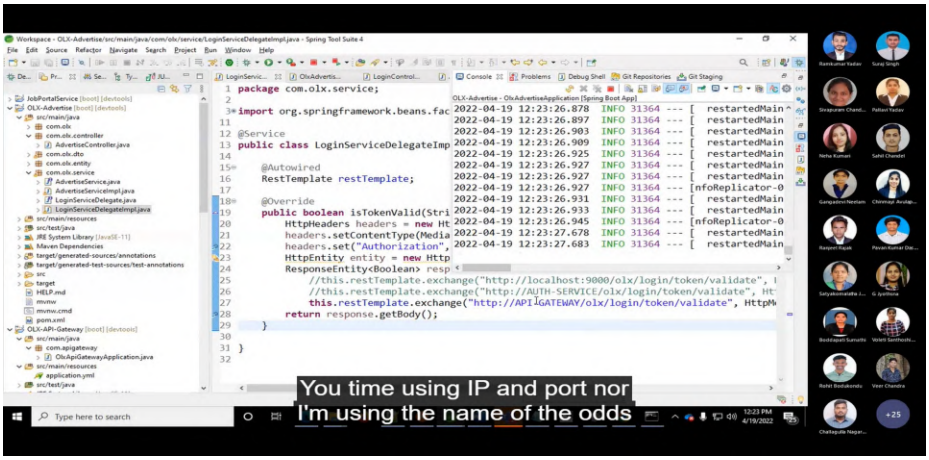

Transcript
Comments
Help
New Spring Starter Project Dependencies
Spring Boot Version: 2.5.12
Frequently Used:
Eureka Discovery Client
Eureka Server
Lombok
OAuth2 Client
Spring Boot DevTools
Spring Security
Spring Web
Available:
eureka
Spring Cloud Discovery
Eureka Discovery Client
Eureka Server
VMware Tanzu Application Service
Service Registry (TAS)
Selected:
Spring Boot DevTools
Eureka Discovery Client
Gateway
Spring Web
< Back
Next >
Finish
Cancel
0:21:54 / 4:00:00


Transcript
Done with this.
0:25:27 / 4:00:00

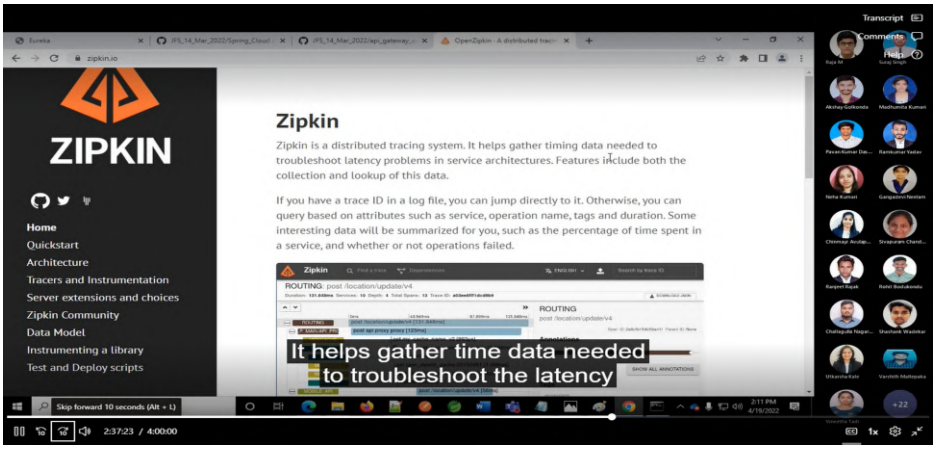

Transcript
OK, now my profitability seems
to be clean over here. Let me
0:29:03 / 4:00:00

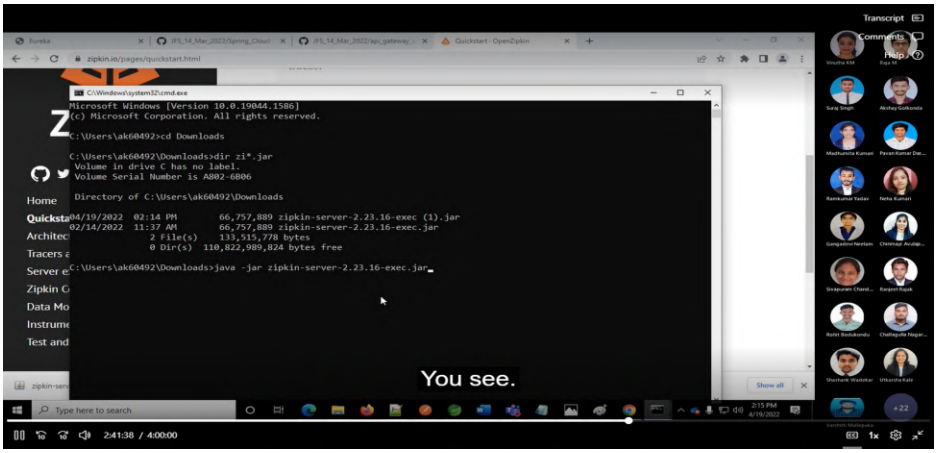


Routes have multiple id's and id's can be anything. But it must be unique. If there is any request coming from someone to the API gateway and in the URL you find /users then please route the request to AUTH-SERVICE. Lb is nothing a load balancing protocol which is used by your eureka server. lb://**AUTH-SERVICE** is a actual name of eureka server. It means with respect to path we are calling eureka server clients. Path must be base URL of application.



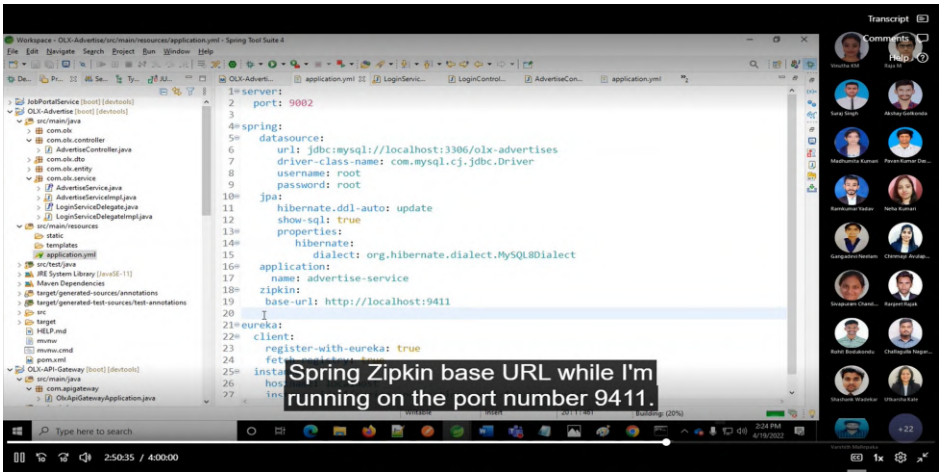
Distributed Tracing is used to do some time slicing or I would like to do some bifurcation that tells to client why is talking more time get response. This bifurcation if you are trying to make it possible somehow, then this is called as a distributed tracing because you are developing a distributed application client call is going via N number of remote call. There are N Number of microservices calls are going to happen. In such application we get some problem on to the clients side. With that we able to find out at which node we are getting more response time. Zipkin server will allow you is going to put the logs. It will show you complete picture that which microservice took how much time.





Now we add dependencies into microservices so that when any call happen then these dependencies able to add zipkin into microservices. And Zipkin give us logs.

```
<dependency>
		<groupId>org.springframework.cloud</groupId>
		<artifactId>spring-cloud-sleuth-zipkin</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.cloud</groupId>
		<artifactId>spring-cloud-starter-sleuth</artifactId>
	</dependency>
```



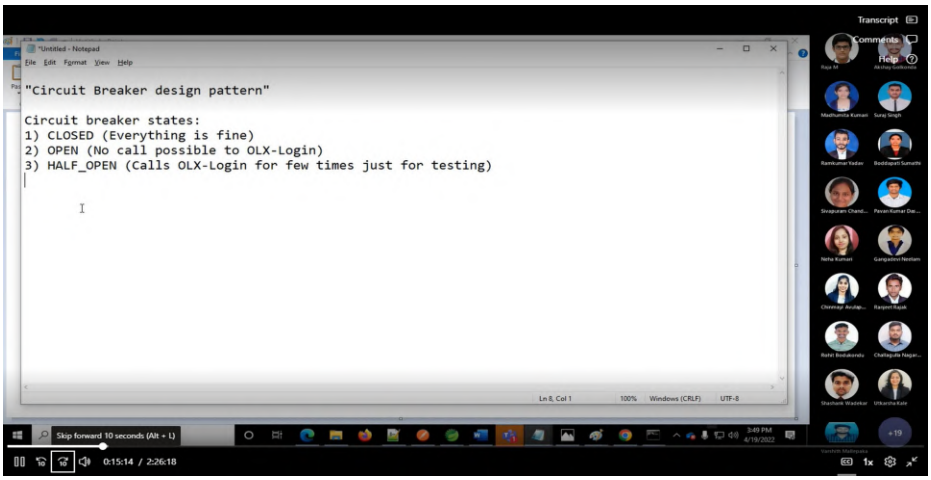
Circuit Breaker

Traditional Communication is a communication. Every time when you going to make the inter microservices client going to hit and due to this, what happens that you will find thousands of unnecessary microservices calls are happening and due to this network traffic will increase

and hence the overall performance of the whole application will go down.

How do I improve the performance of network?

To improve the performance of network you need to have little intelligence in your client microservice. We will call a API if our call fails then we will calls a API's after defining some time bound. So this intelligence we want to add into our client microservice. The intelligence is called as Circuit breaker. It will not make unnecessary calls if service is down. We will reduce the failure of calls.
Now we cannot make any remote call directly every call that we are going to make the remote microservice. It has to go via circuit breaker. This is rule now. Whenever response we will get that will first go to circuit breaker then it will be received by client microservice. Suppose clients calling API again and again then Circuit breaker maintain the history and it will check suppose first five calls if those fails then it will not call API internally no matter if Client calling API again and again continuously for one minute. Internally Circuit breaker will not call API's it will just check only history and reduce the traffic on internet and it will call API after specific time bound.

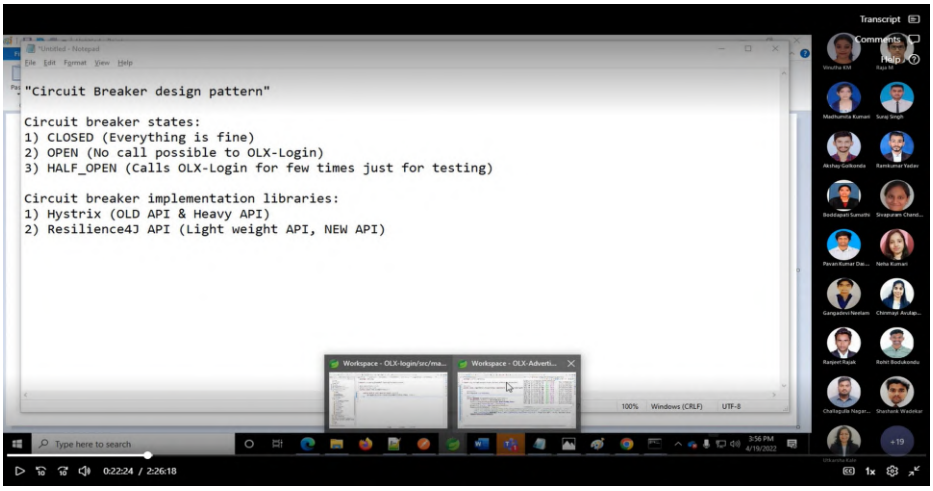


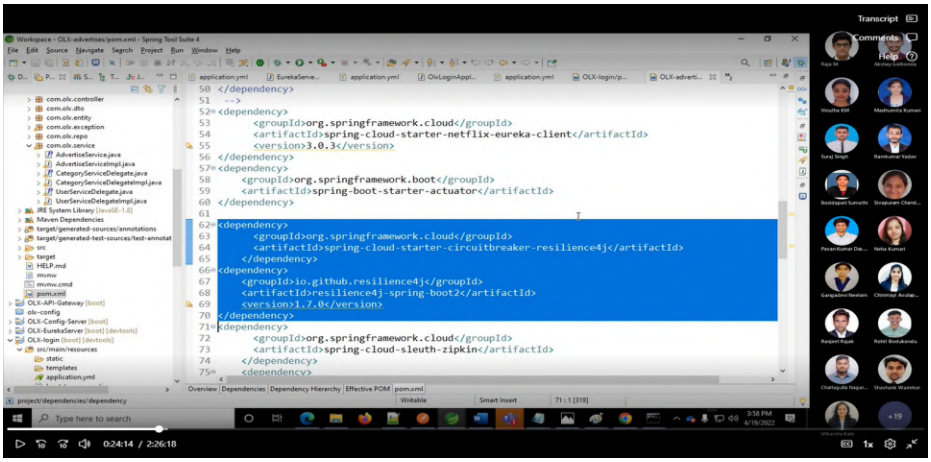
Closed means there is not any role of circuit breaker in the inter communication microservices. Because when client calls API's to another services and it is running and up then there is no role of circuit breaker.

Open state when no call possible to another service.

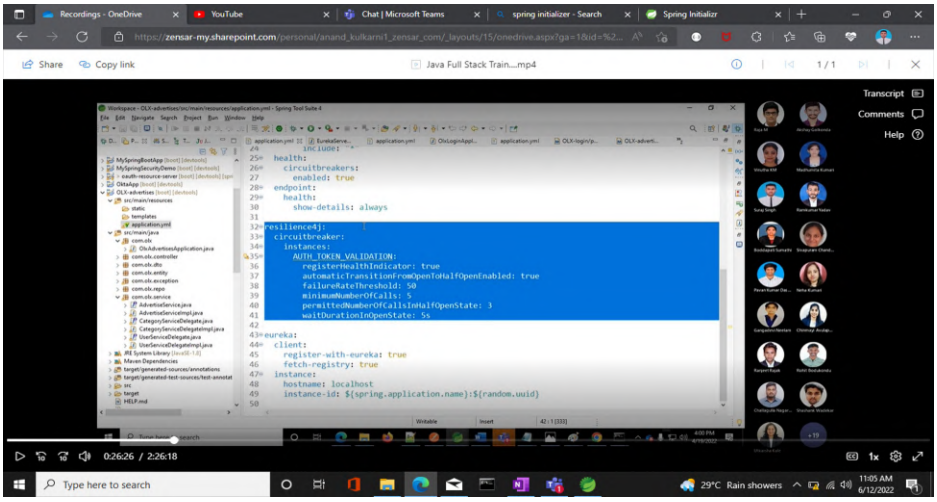
Half Open state means depends upon another service if one more calls made and service up and running then state changed to Closed state else to Open State.

Open & Half open are temporary states.

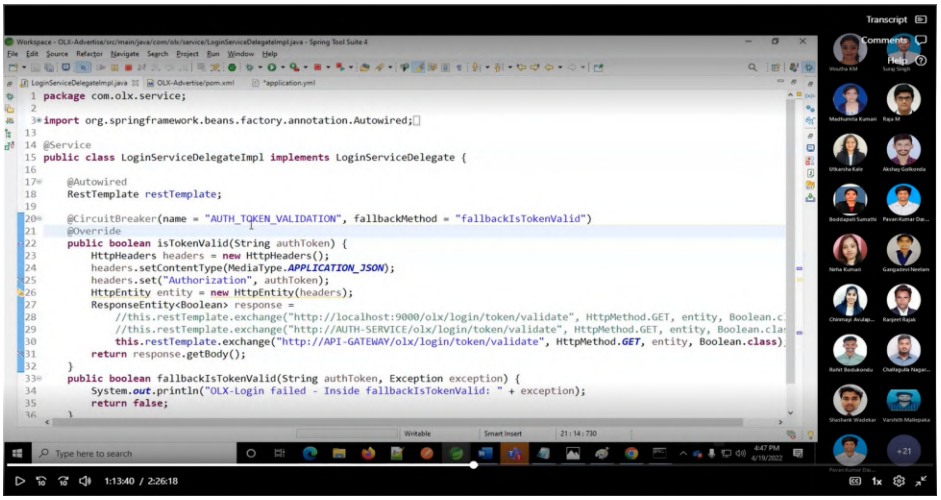


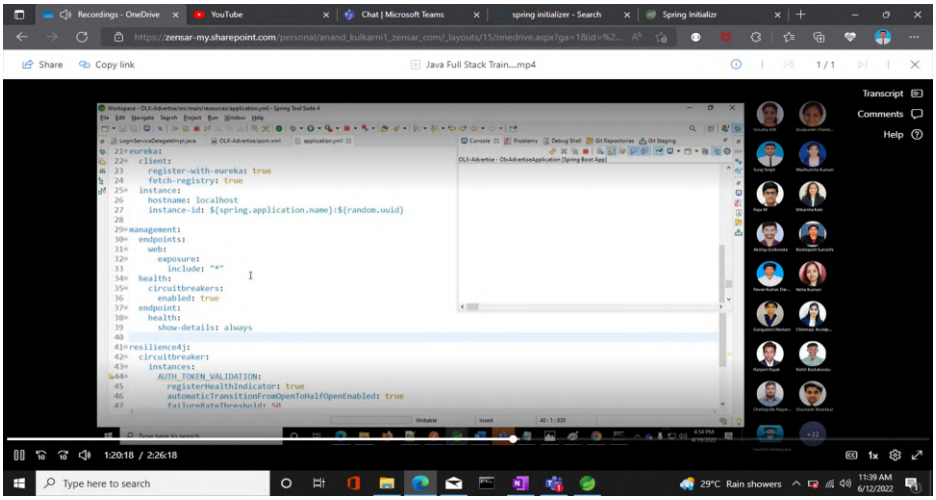


<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
</dependency>
<dependency>
<groupId>io.github.resilience4j</groupId>
<artifactId>resilience4j-spring-boot2</artifactId>
<version>1.7.0</version>
</dependency>



resilience4j:
  circuitbreaker:
    instances:
      AUTH_TOKEN_VALIDATION: (Unique Id of Instance)
        register-health-indicator: true (Enabling actuator to true)
        automatic-transition-from-open-to-half-open-enabled: true
        failure-rate-threshold: 50 (if 50% calls failed it means service is down)
        minimum-number-of-calls: 5 (Minimum calls to make to check servise down)
        permitted-number-of-calls-in-half-open-state: 3 (we will calls 3 times and failed, change half open states to open states)
        wait-duration-in-open-state: 5s (we will wait upto 5 seconds to change the state from open to half open)



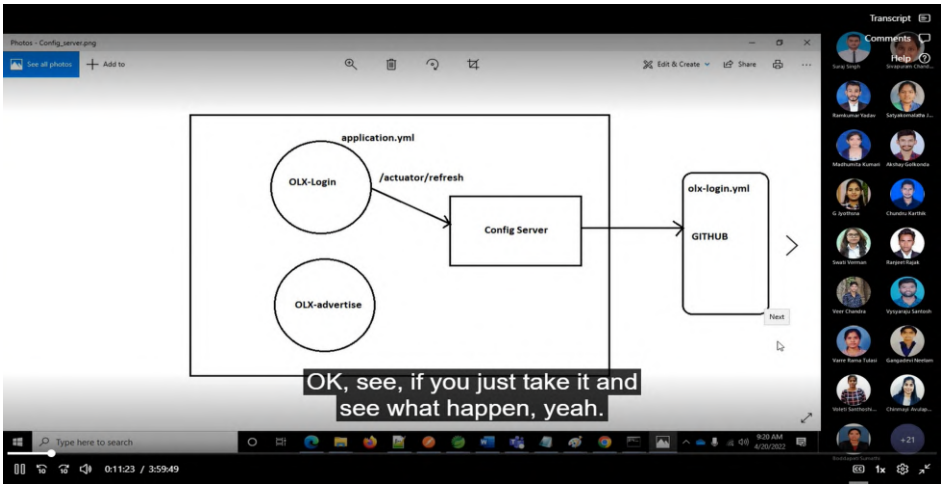


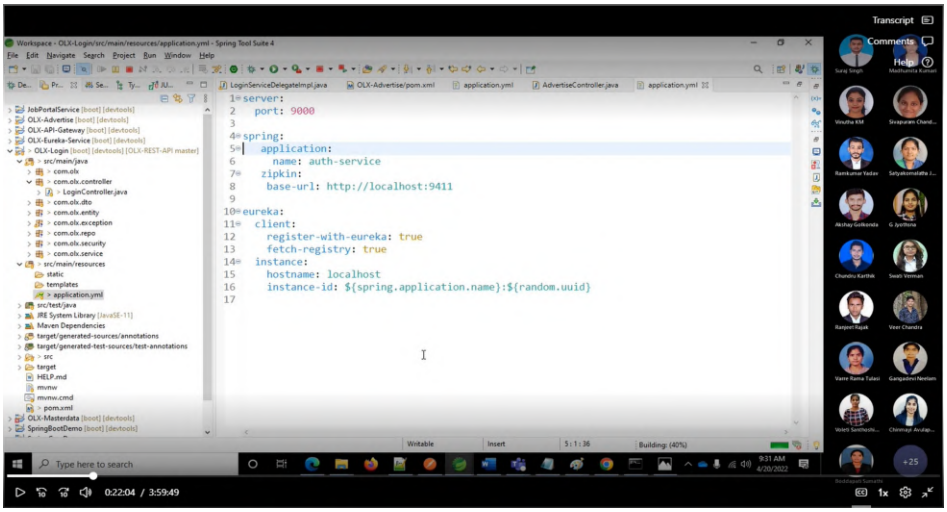
Config Server comes into picture when you think that some configurations can be changed into future. And Application on production and you want to do some changes into configurations in applications.yml file for that you have to stop the application which running on production and have to due required changes into applications.yml and redeploy the application again. It is time consuming as well as not good for user experience so better to make some changes in parameter or configurations on production without stopping the application or redeploying the microservice and make dynamic configurations.

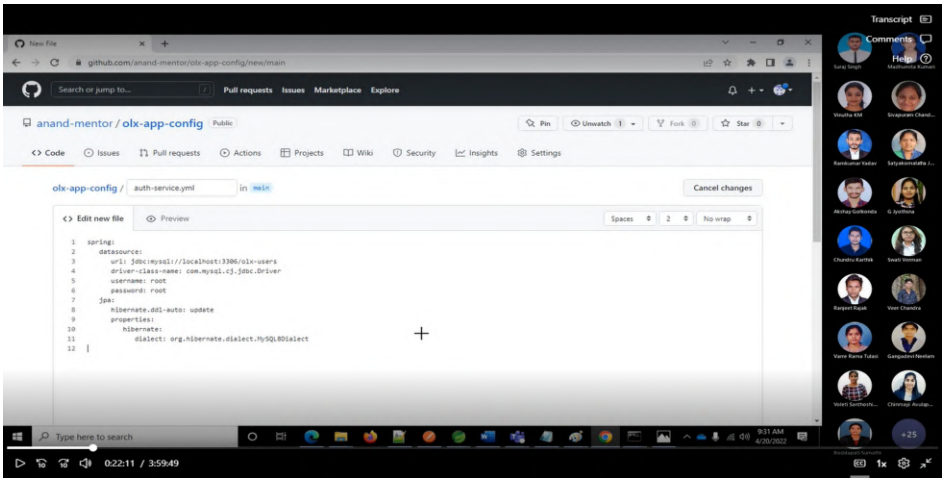
Now we will create application_name.yml on git and if we do any changes into application_name.yml which is stored on git then config server pull all the config from the git application_name.yml and store into memory. Whenever we start our application we will pass all the these config from config server.

3 Steps to make it possible.

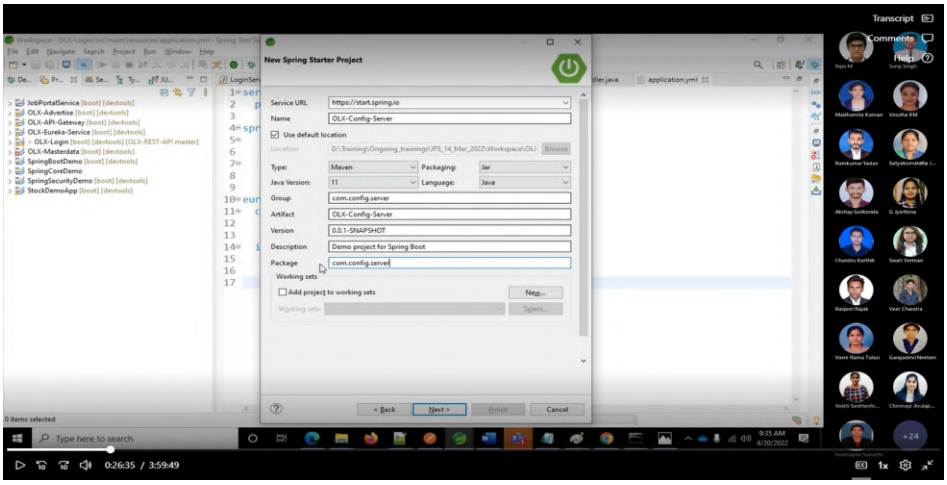
1. Create applications YML files on GITHUB with same name of application name of service. (file names depends upon application name)
2. Create Spring boot application with config server dependencies and give path or url so when it will start it automatically get data from GITHUB.
3. Add Config ser

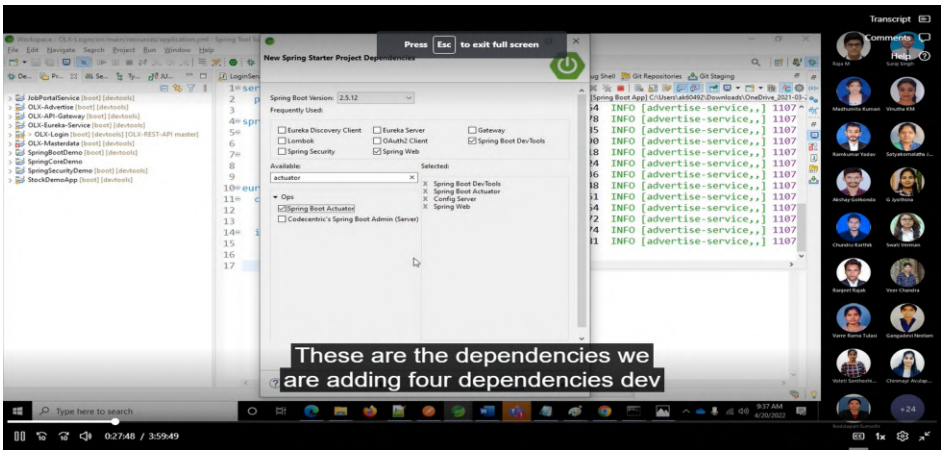

application.yml
OLX-Login
/actuator/refresh
Config Server
olx-login.yml
GITHUB
OLX-advertise
Next
OK, see, if you just take it and see what happen, yeah.
0:11:23 / 3:59:49

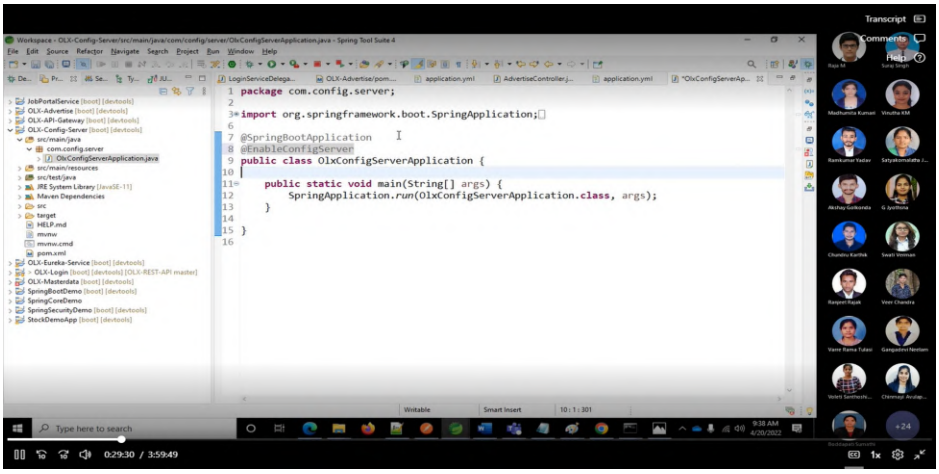

server:
port: 9000
spring:
application:
name: auth-service
zipkin:
base-url: http://localhost:9411
eureka:
client:
register-with-eureka: true
fetch-registry: true
instance:
hostname: localhost
instance-id: ${spring.application.name}:${random.uuid}
0:22:04 / 3:59:49

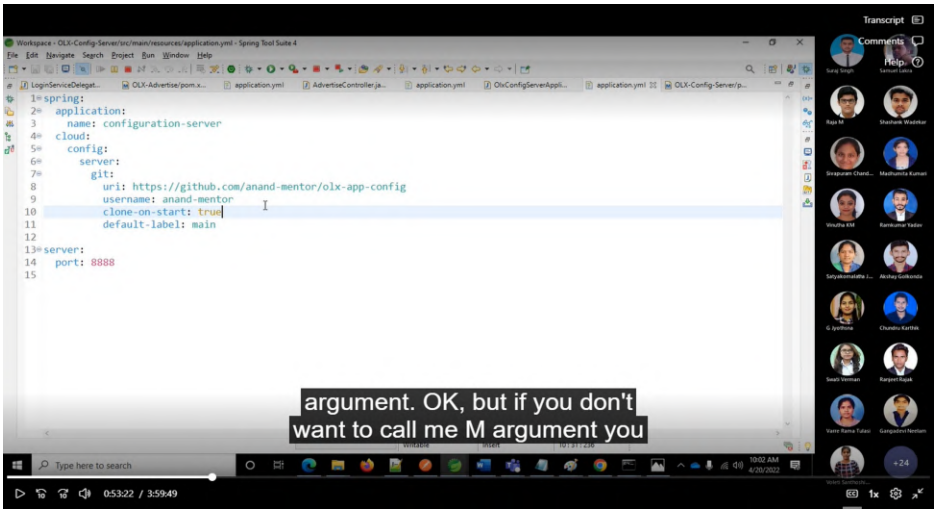

anand-mentor / olx-app-config
olx-app-config / auth-service.yml
Cancel changes
spring:
datasource:
url: jdbc:mysql://localhost:3306/olx-users
driver-class-name: com.mysql.cj.jdbc.Driver
username: root
password: root
jpa:
hibernate.ddl-auto: update
properties:
hibernate:
dialect: org.hibernate.dialect.MySQL8Dialect
0:22:11 / 3:59:49

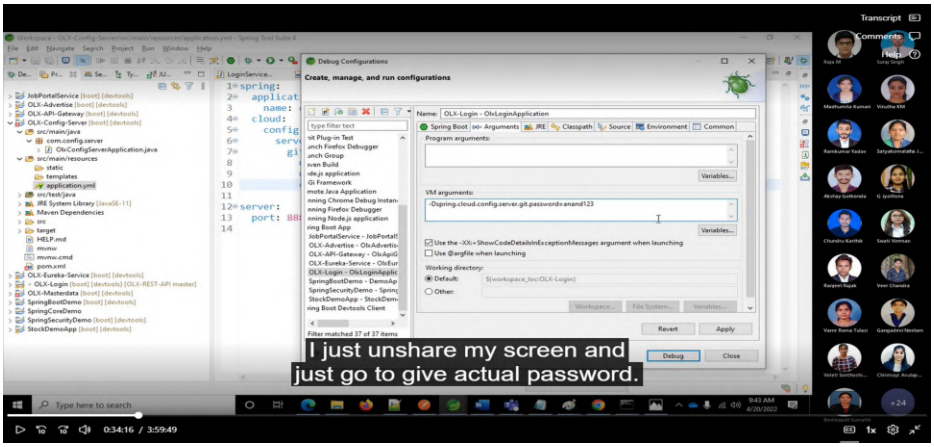
Step 2nd

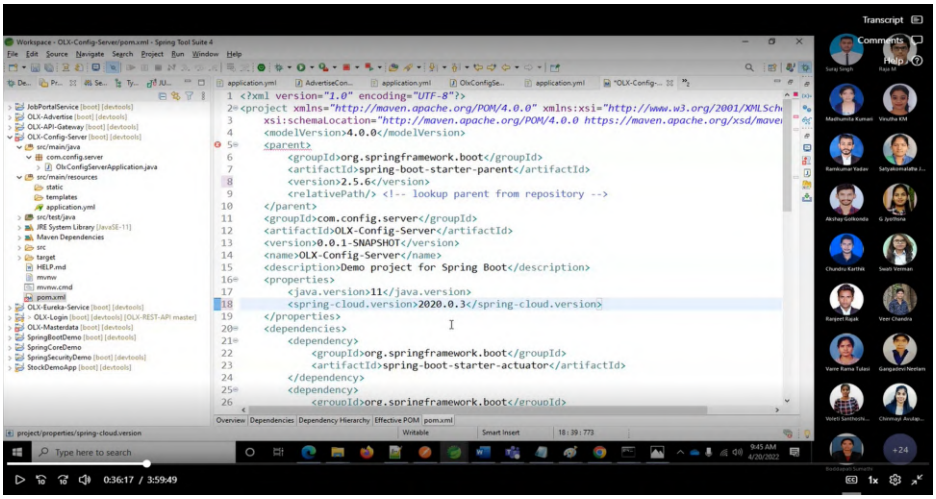

Transcript
Comments
Help
New Spring Starter Project
Service URL https://start.spring.io
Name OLX-Config-Server
Use default location
Type: Maven
Packaging: Jar
Java Version: 11
Language: Java
Group com.config.server
Artifact OLX-Config-Server
Version 0.0.1-SNAPSHOT
Description Demo project for Spring Boot
Package com.config.server
Working sets
Add project to working sets
< Back
Next >
Finish
Cancel
0:26:35 / 3:59:49

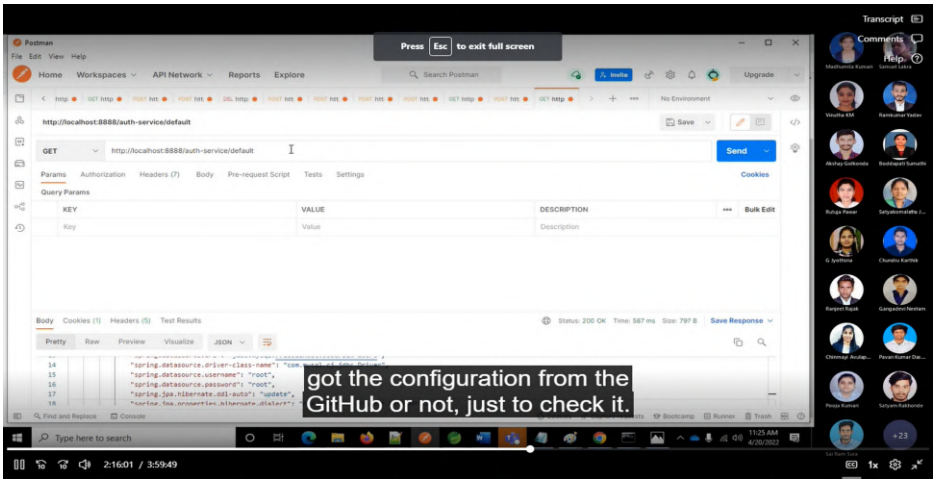

Transcript
Comments
Help
Press Esc to exit full screen
New Spring Starter Project Dependencies
Spring Boot Version: 2.5.12
Frequently Used:
Eureka Discovery Client
Eureka Server
Gateway
Lombok
OAuth2 Client
Spring Boot DevTools
Spring Security
Spring Web
Available:
actuator
Ops
Spring Boot Actuator
Codecentric's Spring Boot Admin (Server)
Selected:
Spring Boot DevTools
Spring Boot Actuator
Config Server
Spring Web
These are the dependencies we
are adding four dependencies dev
0:27:48 / 3:59:49


Transcript
Comments
Help
package com.config.server;
import org.springframework.boot.SpringApplication;
@SpringBootApplication
@EnableConfigServer
public class OlxConfigServerApplication {
public static void main(String[] args) {
SpringApplication.run(OlxConfigServerApplication.class, args);
}
}
0:29:30 / 3:59:49


Transcript
Comments
Help
spring:
application:
name: configuration-server
cloud:
config:
server:
git:
uri: https://github.com/anand-mentor/olx-app-config
username: anand-mentor
clone-on-start: true
default-label: main
server:
port: 8888
argument. OK, but if you don't
want to call me M argument you
0:53:22 / 3:59:49

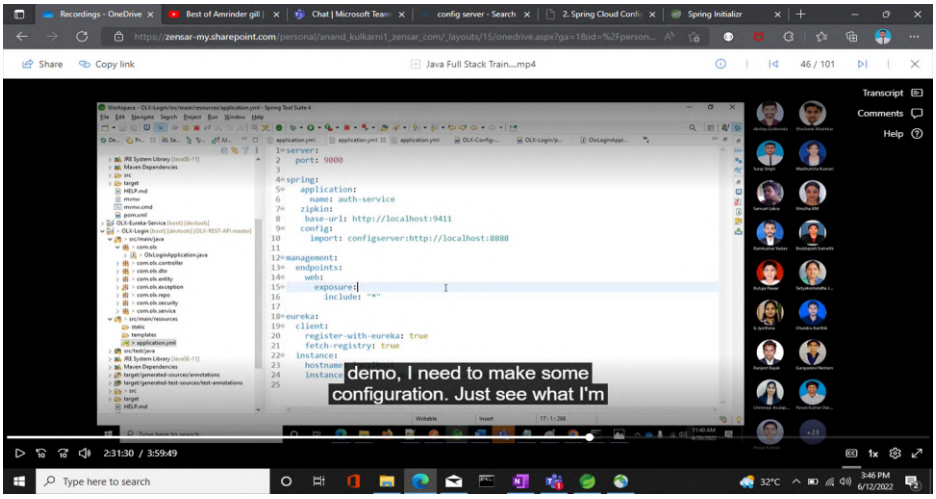

Transcript
Comments
Help
Debug Configurations
Create, manage, and run configurations
Name: OLX-Login - OlxLoginApplication
Program arguments:
VM arguments:
-Dspring.cloud.config.server.git.password=anand123
Working directory:
Default:
Other:
Revert
Apply
Debug
Close
I just unshare my screen and
just go to give actual password.
0:34:16 / 3:59:49


Transcript
Comments
Help
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSch
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/mave
<modelVersion>4.0.0</modelVersion>
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.5.6</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.config.server</groupId>
<artifactId>OLX-Config-Server</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>OLX-Config-Server</name>
<description>Demo project for Spring Boot</description>
<properties>
<java.version>11</java.version>
<spring-cloud.version>2020.0.3</spring-cloud.version>
</properties>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
0:36:17 / 3:59:49

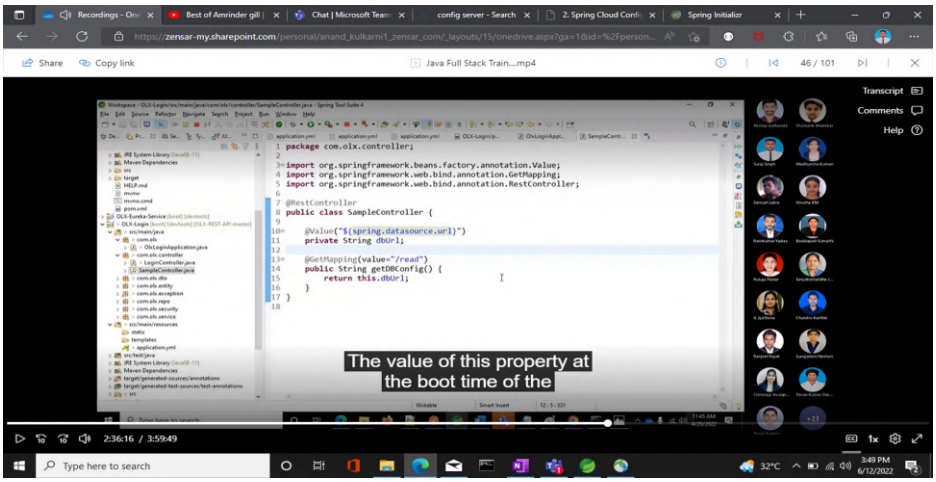


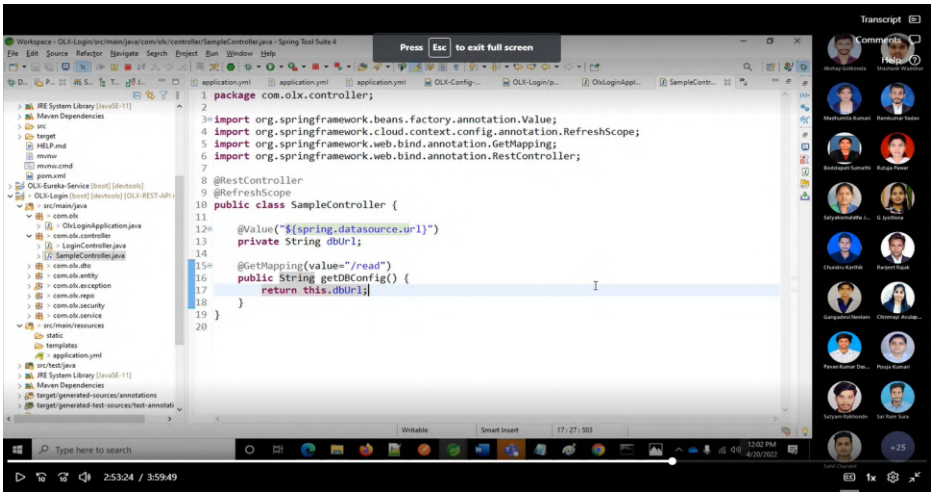
Default means root folder.

Step 3



@Value helps to read application.yml file properties.





Here is a problem that in boot time OLX-LOGIN will get properties from config server. Suppose after some time we did some changes into application YML file stored in database. Config server automatically pull changes from git and store it in the memory. OLX-LOGIN don't know about to tell or to update properties we have to refresh application manually. Url/actuator/refresh