00:00:00.000 --> 00:00:05.140  
Patil, Tatyasaheb  
Yeah, it's recording started so as I was saying about the D3 OK.

00:00:06.790 --> 00:00:36.410  
Patil, Tatyasaheb  
So for the data just to record it for the day three, like microservices security and principles like some of the authentication or like with security principle like what all you can use to secure your application like basic authentication, Oauth, Openid Digest based authentication, then a small demo on how the basic authentication you can implement out how it is implemented and then then open ID Connect also one more example on that front. So my colleagues are more and if it will focus on that now I'll be today focusing on some of the principles and some of the theoretical.

00:00:36.520 --> 00:00:37.230  
Patil, Tatyasaheb  
Concepts about.

00:00:37.970 --> 00:00:39.330  
Patil, Tatyasaheb  
Michael through security.

00:00:40.650 --> 00:00:46.650  
Patil, Tatyasaheb  
Yeah, and like just to recap, like what we learn yesterday. Like you know we saw yesterday like firstly.

00:00:47.370 --> 00:00:47.790  
Patil, Tatyasaheb  
No.

00:00:48.490 --> 00:01:07.100  
Patil, Tatyasaheb  
Make domain driven design like domain driven design like why? Why why domains segregation is important in the micro services? You know? I took an example of a national Stock Exchange and just like just to buy buy stocks and sell stocks is just top level or a holistic view of that. But you know, like what a deep domain it is.

00:01:07.910 --> 00:01:37.610  
Patil, Tatyasaheb  
And how many people are involved there? And water water such a big domain with billions or millions of business processing happens there with the transactions and also like dividing that domain and small spawn subdomains and then like working with micro service on that. So and even not like normalized and denormalized like you just not cutting into very very very small and lick it should not become overhead as well to manage them because you know they you if you divide that domain in such a way that you know it, it should not become a overhead to manage that.

00:01:38.150 --> 00:01:38.900  
Patil, Tatyasaheb  
Micro services.

00:01:39.980 --> 00:02:12.310  
Patil, Tatyasaheb  
Then like my like my friend or armor like looked at like explained some of the monolithic to microservices migration strategies like what all aspects you need to consider on the business front on the technical front on management, management front. So so those are the very important things like you know just just you. Just do not do not directly go and like you know, start cutting the application and start developing because you need to consider like the cost effects of that and like how much time it will take and like is it possible like side by side we run the application and the small part of that moment.

00:02:12.480 --> 00:02:42.410  
Patil, Tatyasaheb  
Having some converting to micro services and then start consuming that and if it starts getting to that whatever the requirement we have like you know, right now we have one client like sunlight, sunlight, both. So what he they have like we have the compass product like from last I think I don't exactly know they don't know the ears but like from last many years these guys are using the compass product which is a 30 year old product. COBOL I'd like Oracle but now these guys are migrating like there's certain type is migrating from this multi application to microservices application.

00:02:42.630 --> 00:02:50.440  
Patil, Tatyasaheb  
But you know these guys are like goalie. Slowly going with the steps like you know they asked for couple of microservices only then they integrated with their UI.

00:02:51.340 --> 00:03:22.330  
Patil, Tatyasaheb  
Couple of microservices only they access Ryan like, integrated with their UI like tested on their user. Accept things like performance testing all this and then they moved that into production production environment. Only couple of ones that got successful. Then they ask for Q more services. Few more. So that kind of a stage door step in steps steps manorhouse staging manner. They are moving towards this complete microservices migration. You know in the end at the at the same time, they're monolithic is also running out. But like you know like within one year or so they completely move to.

00:03:22.540 --> 00:03:35.570  
Patil, Tatyasaheb  
Uh, like micro services and like I think probably stop the monolithic application. Or like maybe for some years they keep on using that, but like this is how like you should be going microservices that you staged manner and it immediately you cannot replace everything.

00:03:36.700 --> 00:03:40.710  
Patil, Tatyasaheb  
That is what the some of the documentation strategy is for. The micro services implementation.

00:03:41.730 --> 00:04:11.850  
Patil, Tatyasaheb  
And then like a like, a Viper Nikhil told about like like he told about like how you can introduce or like work with spring boot applications and like some of the future like the some of the the many features provided by spring boot. Like you know like how you can quickly develop the micro small microservices based application with embedded tomcat and like whatever the starters like by default all the jobs you don't need to click download and like add in the classpath and all those things you can just add the dependencies.

00:04:12.240 --> 00:04:42.770  
Patil, Tatyasaheb  
And like the spring boot starter, parent will take care of, you know, downloading all the required dependencies for your application and like it's it's ready to eat product you know nowadays you know USC like lot of ready to eat products in packets coming at you. Just take it like make it warm and you gotta start hitting like like rated. This is kind of like spring boot is now like ready to it or like proxygen ready open united opinionated approach kind of solutions or like applications you can develop with spring boot with a lot of features you know are there we have to have a video link to download the heap dump our like memory.

00:04:42.850 --> 00:05:04.840  
Patil, Tatyasaheb  
Great comes to that kind of like that kind of few facilities directly provided by using the actuators you know actuator help you out to understand like what is down databases down or application is dialogue where the memory leaks out like where the memory like how the consumption is happening with that micro service and some of the some of the very pretty or useful information about that application easily provided by this spring boot.

00:05:06.320 --> 00:05:22.810  
Patil, Tatyasaheb  
Yeah and then then then we went on like this like explaining or like a small application what we are currently working on. Like if the claims application migrating nickel experience like how we are working on that like exclaims application like how the controllers and services and the model and those are structured.

00:05:22.860 --> 00:05:27.950  
Patil, Tatyasaheb  
So that way we explained some of the very good questions from the audience as well on that front.

00:05:29.220 --> 00:05:31.820  
Patil, Tatyasaheb  
So that is what like we did like a small.

00:05:31.870 --> 00:05:32.560  
Patil, Tatyasaheb  
No.

00:05:34.190 --> 00:05:38.880  
Patil, Tatyasaheb  
Or recap of like yesterday I did and then like maybe I think.

00:05:39.760 --> 00:05:48.630  
Patil, Tatyasaheb  
By this time, hopefully we like all the participants have joined. I'll let me share the today's today's agenda on today's sleep. Then we'll take it forward from there.

00:06:04.710 --> 00:06:06.280  
Patil, Tatyasaheb  
Is it visible at the PPT visible?

00:06:08.350 --> 00:06:08.860  
Yadav, Rameshchandra  
Yes.

00:06:08.490 --> 00:06:09.180  
Durga, Amar  
It's not, yeah.

00:06:09.790 --> 00:06:10.100  
Kennedy, Prarthana  
Yeah.

00:06:12.760 --> 00:06:40.700  
Patil, Tatyasaheb  
Yeah, OK, so any questions or like for like like yesterday or day before yesterday's session. I think couple of guys connected yesterday me like they've tried developing these small like the starter application. I think from Ramesh younger and I think we're shahrani somebody so these guys connected yesterday before like subway issues they face down we are able to successfully resolve the issues and like things are working on their end like they have successfully completed like the starter application.

00:06:41.370 --> 00:06:47.340  
Patil, Tatyasaheb  
Anybody ends like experience rain issues or like successfully completed, that or any other questions.

00:06:48.300 --> 00:06:55.220  
Patil, Tatyasaheb  
Like on that securely. Sorry the strategy is pregnant or domain driven design or the whatever the stuff we've seen on the first day.

00:06:57.660 --> 00:06:59.900  
Patil, Tatyasaheb  
We can take up that and we will continue our.

00:07:00.930 --> 00:07:01.470  
Patil, Tatyasaheb  
Day 3.

00:07:05.150 --> 00:07:08.830  
Kundaragimath, AshwiniAndanayya  
I was able to do it that we successfully I started application.

00:07:09.920 --> 00:07:11.230  
Patil, Tatyasaheb  
Yeah, great great.

00:07:12.550 --> 00:07:13.060  
Patil, Tatyasaheb  
Thank you.

00:07:23.580 --> 00:07:24.940  
Patil, Tatyasaheb  
Yeah, I think I think like.

00:07:26.250 --> 00:07:34.520  
Patil, Tatyasaheb  
Everyone is looks good on that front, so let us move on today's today's session so they three like we are, we have.

00:07:36.640 --> 00:07:37.190  
Patil, Tatyasaheb  
Uh.

00:07:38.540 --> 00:07:57.530  
Patil, Tatyasaheb  
This is the agenda for today. Like what is the importance of of like security in microservices or some of the principles in microservices security? And then we'll talk about different different mechanisms or like methodologies like basic authentication and Oauth, two Openid and Digest based on those are the few things.

00:07:59.270 --> 00:08:00.520  
Patil, Tatyasaheb  
We are going to focus on that.

00:08:01.270 --> 00:08:03.640  
Patil, Tatyasaheb  
So let us tap so like.

00:08:04.910 --> 00:08:13.680  
Patil, Tatyasaheb  
You know why it means maybe I will ask him like why security is important? Maybe or like anybody knows like some of the some of the the some of the last.

00:08:15.220 --> 00:08:20.010  
Patil, Tatyasaheb  
Not security hacks or like hacking happened on this industry. Anybody any thoughts?

00:08:22.080 --> 00:08:25.340  
Patil, Tatyasaheb  
Like you're not in 2007, I'm sorry, 2017.

00:08:26.760 --> 00:08:29.190  
Patil, Tatyasaheb  
A big big one happened everywhere.

00:08:31.780 --> 00:08:32.830  
Thirugnanam, Kumaran  
The ransom mirrors.

00:08:33.140 --> 00:08:33.600  
Patil, Tatyasaheb  
Correct?

00:08:34.480 --> 00:08:47.890  
Patil, Tatyasaheb  
Ransomware is one of the biggest, like, I think, like a hacking attack happened over over the whole world, and like the the the cost involved around that. So like if you have the examples I'm I'm trying to put in front of you guys.

00:08:48.590 --> 00:08:58.760  
Patil, Tatyasaheb  
So this is one of the like a like a hacking attack happened on like Equifax credit Bureau Quick Credit Bureau like you know where like.

00:09:00.250 --> 00:09:20.040  
Patil, Tatyasaheb  
This is one of the biggest, biggest or worst breaches. All that all times as very secure information or like a sensitive information got exposed and like for millions of people. So for millions of people there all personal information and like like the Social Security numbers like like this, this information got hacked.

00:09:21.090 --> 00:09:48.860  
Patil, Tatyasaheb  
And like that's why I like the security is important. Like why this slightly saying why security is important? Because you know, like if you the data is stolen it's it's not know, right, right? Good right. So that's why I like the security is important and like with these microservices you know we have different different different like with monolithic application. You know you have like a single layer like single application and like no attack attacks can happen on different different areas at different different places like maybe at service layer or maybe printed layer or maybe database layer.

00:09:50.140 --> 00:10:19.580  
Patil, Tatyasaheb  
So we have to secure this with, you know, with microservices, this is begin again. More challenge because you have separate unit. Like every unit you need to secure. Like for this single application, single securities chili important but you know you have single unit to control or single single unit to save off from all these attacks. But with microservices this is this became more complex because you have many units, took care of. Take care of all the units, should have the same security or like same same same kind of protocols should be following.

00:10:19.850 --> 00:10:28.560  
Patil, Tatyasaheb  
Or maybe you can segregate that as well, like some people are putting some kind of private or sensitive information in in private clouds as well.

00:10:29.900 --> 00:10:32.210  
Patil, Tatyasaheb  
Then that one of the another important.

00:10:32.310 --> 00:10:45.260  
Patil, Tatyasaheb  
Or or or liquid the next like example on the security front. Trees like Yahoo, Yahoo 3 billion accounts like sometime back like you know like yeah, I was like all the information of all almost 3 billion users.

00:10:45.750 --> 00:10:50.060  
Patil, Tatyasaheb  
Uh, got hacked with the personal information from all over the world.

00:10:51.330 --> 00:11:22.020  
Patil, Tatyasaheb  
And then like as in the first, we are talking about like the first like like in 2007. You know, like some of the people may be already aware of, like some are new joiner, but still this gram somewhere you heard of this world, this this is targeted with the Microsoft Windows machines and like you know in like all the data from these machines get hacked. I'm like you don't have a control on the machines like they ask some random payments you have to pay them and then only the motion machine get unlocked. So such a huge impact for almost 3-4 days it's everywhere it's running like.

00:11:22.070 --> 00:11:52.640  
Patil, Tatyasaheb  
People are just talking about that talking, and even like those people who are working with FIS, you know you may have looked like before that you know FIS also not much having like like we have the securities, but you know we are by that time like you know anyone can download any software and installing his machine. And like we had certainly not doing anything wrong but like for do some our daily jobs. We download some software since 12:00 AM try to do some POC and like our normal stuff we'll keep on doing for some of the like project related task and that is certainly not a like a wrong like intention.

00:11:52.690 --> 00:12:12.580  
Patil, Tatyasaheb  
Like what do I choose? Some of the tasks with downloads or painful but you know, with this attack from 2017 that everything got changed. You know if you have to install any software on any like your machine or any machine you have to go through this all approvals and then like the if the software is like this guy verifies that and then the software is installed and.

00:12:13.330 --> 00:12:30.550  
Patil, Tatyasaheb  
Those are the all things came into picture. Like before that you know I remember like I joined FIS IE download any software installed on my machine. No, no in any secret like I download any software and I installed on my machine but like after these ram somewhere like I have to like go through all the processes approvals and then only you can install the software.

00:12:31.610 --> 00:12:33.470  
Patil, Tatyasaheb  
And that is very important as well, you know.

00:12:34.430 --> 00:13:04.640  
Patil, Tatyasaheb  
I remember like I was in one of my company and like they're also one of the guys saying for example, you know like there is a free software like you can easily download and install it on but they have terms and conditions. Disclaimer for that and like in that disclaimer they have mentioned like for one year you can use it freely but after one year if you are paying you you are using this. You have to pay this money and if you are not paying this money you are you're liable for like we can file your fire against you or you have to pay this like you have to pay these penalties and this and that.

00:13:04.910 --> 00:13:15.360  
Patil, Tatyasaheb  
So this this guys are keeps kept on using that kept on using that and after like one year or so like that company who has developed that software like raises that issue.

00:13:16.040 --> 00:13:34.080  
Patil, Tatyasaheb  
Like you are using this software and you have to pay that now like on that time like it's it's a huge like escalation everywhere in this company happened. I'm like we have to pay that much money so make lot of like the employees or the people have not done any intentional things but it happened like there are some terms and conditions we just overlooked that and that causes.

00:13:34.980 --> 00:13:49.960  
Patil, Tatyasaheb  
For long term impact, so certainly we should be like taking Mexico City is one of the very important task. Like you develop the application functionality and on the similar lines or like on the very higher end its security is very important thing. We should take care of.

00:13:51.060 --> 00:14:03.430  
Patil, Tatyasaheb  
I think a proper example CI mentioned here, like the domain to one like lot of people you know, we enter all the credit card information to pay the things in credit card, debit cards or call information gets saved and like this, this gate also hacked.

00:14:05.100 --> 00:14:07.760  
Patil, Tatyasaheb  
And then like like our Aadhaar card.

00:14:08.910 --> 00:14:29.460  
Patil, Tatyasaheb  
Yeah, that that also like other calls. Also having some issues like lot of Indian Indians they terminated two other card personal information and like and like biometric database. You know all fingerprints like how much the Penguins are important. You know like if you are in like something up in Bates or accessing some of the company and if this data is stored and somebody can easily enter into that system. So so it's very important.

00:14:30.570 --> 00:14:34.250  
Patil, Tatyasaheb  
But these are the different different attacks happened. I like it has a huge cost.

00:14:34.940 --> 00:14:45.800  
Patil, Tatyasaheb  
You know, like we don't see, but like the people that like at very top end seating they don't know how how they have to, how, how much costs they have to bear for this. This particular attacks and they have to.

00:14:47.200 --> 00:14:53.950  
Patil, Tatyasaheb  
They have to pay for this. There is no option, so some of the few of the examples like further data, cost of data breaches like happened.

00:14:54.990 --> 00:15:15.920  
Patil, Tatyasaheb  
Sometime back like no and these are the few few records like it's a. It's like you all will data. This is sometime back like we have people have some source from some of the sources but like like average like average on on an average like 3-3 millions is a cost of single data breach ever is like it varies like sometimes it goes through two 200 or sometimes it's.

00:15:17.750 --> 00:15:19.980  
Patil, Tatyasaheb  
Make a little down but like average cost.

00:15:21.210 --> 00:15:51.620  
Patil, Tatyasaheb  
I would discuss with 3.62 million or every data breach and like you know, mostly focused on like the post most focused industries, financial industry, financial and healthcare industry. I think from the data or the from the data we've got like these are very important like entities wearing a lot of attacks happens because you directly case directly get access to the customer accounts and customer balances and you can directly start doing like paying the things that nowadays you know people. Like recent example I saw.

00:15:51.920 --> 00:15:55.770  
Patil, Tatyasaheb  
Now these hackers have posted their own numbers as a customer care numbers.

00:15:56.690 --> 00:16:11.090  
Patil, Tatyasaheb  
Customer care numbers as their own numbers, and people call that number. This is OK. We are from customer care only tell me what is your OTP and they did their transactions by the customers. We have. You know, even people are posting their like these hackers are posting their own numbers as a customer care numbers.

00:16:12.450 --> 00:16:24.190  
Patil, Tatyasaheb  
With that kind of stuff is happening, so we should be like, you know we are in the software industry. Like we we still make mistakes still and even then then consider about like the non like non it people or even like.

00:16:24.530 --> 00:16:43.740  
Patil, Tatyasaheb  
But maybe in India we have this. This much population educated, non educated period. This calls any very soft manner, talks with very soft manner. Access your credit card data or like ask for the credit card data and then do the transaction on your behalf. So it's very important security is very important in this as this aspect and everyone like by this time I think.

00:16:44.410 --> 00:17:02.840  
Patil, Tatyasaheb  
Like we all know this word, why security and what is? What is the importance of this? And like you know certainly there are different protocols. Different mechanisms came into picture and with this stateless stateless implementation of rest services, now we have this over based authentication or like you know you have to have the token every time you want to pass that token.

00:17:04.250 --> 00:17:21.840  
Patil, Tatyasaheb  
I don't know what you know nowadays. Not don't want to just to keep that you know earlier in earlier models happening we are just keeping a user data table. You know user data and where we put the username and password and like that kind of stuff like possibility. Certainly encrypted. But like earlier we have user data table and like where we are putting all the information related to users.

00:17:22.600 --> 00:17:30.930  
Patil, Tatyasaheb  
So so those things are now going away and people are going towards this over based override C and like whatnot different protocols coming up so moving towards that.

00:17:31.860 --> 00:17:34.150  
Patil, Tatyasaheb  
And that's why this security is important.

00:17:35.300 --> 00:17:37.970  
Patil, Tatyasaheb  
How to secure microservices like what we should be doing?

00:17:38.830 --> 00:17:42.300  
Patil, Tatyasaheb  
Upload the important factors you know and like.

00:17:43.050 --> 00:18:12.530  
Patil, Tatyasaheb  
This this securing microservices? Why important? Because you know we are like executing our like exposing the end points. We are exposing the end points to some of the developers and developers are like software developers are the bigger based hackers for this. This kind of a micro service because they know they know the code better. They know how the endpoint works, they know how to hack it. So so like the consumers of this micro service are very important. Like there's people only like that's that that thing we need to explore that thing. We need to take into consideration.

00:18:13.360 --> 00:18:16.020  
Patil, Tatyasaheb  
Like how much important that you should be like?

00:18:16.510 --> 00:18:27.440  
Patil, Tatyasaheb  
A meeting that this this guy like like this person sitting at other end like he knows like what I wrote and how he is going to have that so accordingly we have to build that that kind of a security on.

00:18:28.770 --> 00:18:59.720  
Patil, Tatyasaheb  
Uh, security on the microservices front. Like you, you for this. Microservices? Nowadays, like you know the the customer base is varying, not just like the UI, like some of the third party applications or like like I I was talking about the example like couple of days back. Like like this service warrant or micro service. You know we consume services from different different vendors and build our application on top of that. Consumers like mobile is application on tablets or laptops unlike some other third party systems.

00:18:59.940 --> 00:19:12.670  
Patil, Tatyasaheb  
What? Not like they've consumers for this microservices are nowadays in large extent or use extent, so we have to be take care of this. All. All these varying varying number of large customer base for the microservices.

00:19:15.400 --> 00:19:44.540  
Patil, Tatyasaheb  
And then like make some of their like nowadays like these people talking about, like microservices should talk across like on premises like internally you are talking between one micro service to another microservice and they should not be direct access like once in micro service directly call to other Microsoft. Now suppose you have API Gateway based authentication and you just validated there and like and you directly give allowed access to all the microservices internally. So suppose somebody tries to reach tried or try to break that API gateway level.

00:19:44.620 --> 00:20:02.630  
Patil, Tatyasaheb  
And then you directly enters in the system and then he can go directly to admin data. You know, even like that kind of work there also, we should be very carefully passing the token for accessing what what information given access to the user. So you know there are two principles like authentication and authorization.

00:20:03.270 --> 00:20:17.650  
Patil, Tatyasaheb  
You're just just not authentication, but this is another layer. Authorization is that user, even though by mistake or some by some hacking he entered into the system, does this user have access to that particular resource, or does he can access some admin related stuff? Admin related data?

00:20:18.310 --> 00:20:24.390  
Patil, Tatyasaheb  
So that kind of two layer two more like more layers of authentication Rajesh and we should be providing.

00:20:25.210 --> 00:20:46.630  
Patil, Tatyasaheb  
And with with at least privileges, you know what microservices standard says you should be providing very less privileges to people at at. At first you should. We just have access denied and then according to the role you should be like increasing the OR like providing more more and more authorization to the specific resources. That is one of the few like few principles what microservices says.

00:20:48.170 --> 00:21:13.160  
Patil, Tatyasaheb  
Then like I heard like like this, microservices says, like you know, you segregate the sensitive information like sensitive information in terms of like our our like personal data or some of the like bank information. Really out bank information relative. Suppose you have some bank related data. So say agreement that information segregate that inform your sensitive information from like like that. That should be independent of microservice which is which is you know.

00:21:14.210 --> 00:21:15.180  
Patil, Tatyasaheb  
How access to?

00:21:16.030 --> 00:21:48.510  
Patil, Tatyasaheb  
That particular that particular data only like all the micro service should not be talking to that sensitive information, but the Microsoft all like like this like pink box and the other like other boxes so they should not have direct access to that personal or sensitive information. But you should be talking to one of the microservices which only has access to that data now directly. All Microsoft should not be talking with that, but I'll specific microservices only talking to that. And then the other microservices just to consuming that and like like access to that information by through that microservices that principle system.

00:21:49.080 --> 00:21:51.540  
Patil, Tatyasaheb  
Or make you minimize as much as possible.

00:21:52.500 --> 00:21:57.470  
Patil, Tatyasaheb  
People like R2 connecting to that actual data or like private data.

00:21:59.040 --> 00:22:07.750  
Patil, Tatyasaheb  
So that that reduce the risk or like some mishaps, who are going to happen because of like that. So try to isolate that as much as possible.

00:22:09.750 --> 00:22:11.490  
Patil, Tatyasaheb  
Put our dedicated microservices.

00:22:12.350 --> 00:22:16.500  
Patil, Tatyasaheb  
So for this for this, for this personal information or sensitive information.

00:22:18.330 --> 00:22:20.240  
Patil, Tatyasaheb  
Then try to put some trust boundaries.

00:22:22.060 --> 00:22:35.560  
Patil, Tatyasaheb  
Right? Like like people coming like like this circle in the middle season. You directly talking to one of the like microservices? No there there should be like have some some kind of authentication are you came into that like First green box?

00:22:36.260 --> 00:22:58.020  
Patil, Tatyasaheb  
Hope you block. So should you should not be talking to another microservice or like some of the secure microservices over here, but put some some more authorization kind of stuff over here. They should not be directly coming and like talking to each Microsoft or any micro service talking to any micro service and like passing the information and getting the access to the all the data. So instead of that try to put some some more boundaries.

00:22:58.840 --> 00:23:16.050  
Patil, Tatyasaheb  
With some authentication and authorization. So depending on like what it seems like we should be authenticating authenticating the users and do an authorization check on that request and then only it both very patient succeeded. Then only he should be able to access the data so authentication and authorization. So what I talked about earlier?

00:23:17.970 --> 00:23:36.040  
Patil, Tatyasaheb  
That that that should be we should be taking care of that then they like depending on the isolation level, infrastructure, security risk at hand, service to service requests may be classified as a trusted and untrusted. So like depending on like the different isolation like private cloud or public clouds, you should be segregating this.

00:23:37.160 --> 00:23:38.180  
Patil, Tatyasaheb  
This microservices.

00:23:40.450 --> 00:23:58.560  
Patil, Tatyasaheb  
And then like inter service communication like Internet service communications so you should not be directly a person directly coming into the system. And like without any trust boundaries it directly starts through accessing the data. So we should be stopped that we should be defining that infrastructure in such a way that you know somebody should not directly come in and access the data.

00:24:00.180 --> 00:24:10.180  
Patil, Tatyasaheb  
I didn't like dig like Pew of the best practices. What microservices says. You know you should be having a user principal up list privileges. So what it says? Like, you know, give a minimal access.

00:24:10.930 --> 00:24:15.550  
Patil, Tatyasaheb  
Minimal exit that is required for one 1% to complete his job given minimum access.

00:24:17.380 --> 00:24:28.690  
Patil, Tatyasaheb  
Or like what they say is like you know first that we find definition should be no access. And then according to his role that he should be able to able to access only that particular that particular resource.

00:24:30.490 --> 00:24:50.040  
Patil, Tatyasaheb  
Then like moving like the other practice says, you know, while you have inter service now. Now with the microservices like inter service and inter service communication is becoming very important because you know you cannot have everything in my 1 Microsoft as we talk about the microservices himself says you know there are different micro different different services and they are talking with each other.

00:24:51.460 --> 00:25:21.190  
Patil, Tatyasaheb  
So this inter service communication like we are using like API Gateway pattern where you validate the all the everything ADS at front end or like first first time only and then all communication happen without any authentication or like the service discovery mechanism where the service is talking with each other. But every time needs to be authenticated like our application also like what we have in place every request call get authenticated even you are talking from one micro service to another bug. These all authentications stored at like the request context and every time through headers. These authentication get past.

00:25:21.610 --> 00:25:26.180  
Patil, Tatyasaheb  
Then when the other microservices started looking at it again, checks for the application.

00:25:27.210 --> 00:25:35.990  
Patil, Tatyasaheb  
Invalidates the username, password and then only gives access to that that particular resource, so so that kind of mechanism we should. We should be putting in place.

00:25:37.460 --> 00:25:47.800  
Patil, Tatyasaheb  
And then then these are the few few important points like when like when you are securing your codes like or like making your your services secure. There are few bullet points.

00:25:49.090 --> 00:26:08.370  
Patil, Tatyasaheb  
Like, say, when it would it points we have and those what I want to lick validate the input. Nowadays you like the the text boxes or like the things we enter through like Angular JS, basically wise or applicant UI based applications. User enters some some kind of like you know select star from user data in the text box.

00:26:09.690 --> 00:26:20.240  
Patil, Tatyasaheb  
I want to see any corner this SQL injections, SQL injections people do and that that that causes like a data hack so you know we should be validating the whole data.

00:26:21.300 --> 00:26:34.640  
Patil, Tatyasaheb  
People put some slashes or some some some and we are the characters in in that particular input boxes and that causes and SQL injections. So validating input is one of the important front or important tasks.

00:26:35.960 --> 00:26:37.750  
Patil, Tatyasaheb  
Then designing the security policies.

00:26:38.570 --> 00:26:55.350  
Patil, Tatyasaheb  
Period like like we will design application like you know we follow some design patterns like no there are different behavioral, structural or architectural design patterns like this. Microservices communities says like hey, you should be like architecting on the on the security front as well. You know you should have a separate architecture for your security.

00:26:56.370 --> 00:27:04.070  
Patil, Tatyasaheb  
While implementing this while implementing your microservices based application, you security should be completely or separately designed follower applications.

00:27:05.120 --> 00:27:28.280  
Patil, Tatyasaheb  
Try to keep, don't make it. Make it complex. You know small like if you make complex and complex application you don't know from where they security had can happen. So this is like try to keep it simple and small so you should be able to easily manage it or get it down and then fix it and then again make it up. So like make this make this as much as as much as small and simple with complex.

00:27:29.400 --> 00:27:37.410  
Patil, Tatyasaheb  
Complexity, it becomes like a easy to hack it because we also don't know. Like with this such so much of complexity what is going wrong?

00:27:38.240 --> 00:27:39.520  
Patil, Tatyasaheb  
So that is one of the principle.

00:27:40.710 --> 00:27:47.820  
Patil, Tatyasaheb  
Practice and then the next point stays like dinner. By default the deliverable. So by default we should not be giving any access.

00:27:48.860 --> 00:27:52.270  
Patil, Tatyasaheb  
Access is denied and then the production line scheme accordingly.

00:27:53.350 --> 00:27:54.750  
Patil, Tatyasaheb  
You you you the access.

00:27:55.380 --> 00:28:16.380  
Patil, Tatyasaheb  
I thought I was talking earlier in. By default access is denied. You are talking to any specific service or microservices. By default it should be access denied according to your roles and responsibility. You should be able to access that by default. Like this. This principle says that or the core practices says that like you should be how minimum privileges minimum access.

00:28:18.300 --> 00:28:20.130  
Patil, Tatyasaheb  
Moving right, there are few more.

00:28:22.130 --> 00:28:32.170  
Patil, Tatyasaheb  
Be more things like you know you should be preferably list privileges. Give minimum access, then like sanitize the data sent over the system. You data sanitization happens and then.

00:28:33.620 --> 00:28:35.350  
Patil, Tatyasaheb  
Practice with the difference like.

00:28:36.580 --> 00:28:47.620  
Patil, Tatyasaheb  
So we hurry up. As you know, the policy of first line of defense, second line of defense. So we should have that that kind of different different layers in the security front should be how like.

00:28:49.210 --> 00:28:52.390  
Patil, Tatyasaheb  
Different different layers on the security front as much as possible.

00:28:53.810 --> 00:29:21.790  
Patil, Tatyasaheb  
No, like nowadays you know separate security teams now in organize separate security infrastructure handling things are sitting over there. Who knows like how to implement 6 security and how to how to set up that infrastructure for that, like earlier, that was not the case. But with this increasing number of hacking's like every every year or every month or every day like how how many attacks like when it happens only with the people like the like sensitive information and all like you may hold up like this this banks every day gets.

00:29:23.240 --> 00:29:33.250  
Patil, Tatyasaheb  
The I I heard up some sometime back. You know this banks get every day, every day three lack plus hacking attacks, three lack plus hacking attacks every day and how these guys are, you know, secure dating this.

00:29:33.920 --> 00:29:44.600  
Patil, Tatyasaheb  
I had long back as we know this Microsoft CEO Bill Gates every day he gets two lack 2 left plus you know spam mails spam mails for to add 2 accesses Inbox.

00:29:45.840 --> 00:29:50.010  
Patil, Tatyasaheb  
So that that kind of information like that is why the security is very critical.

00:29:50.920 --> 00:29:59.410  
Patil, Tatyasaheb  
And that's why, like we are like little bit focusing on that. And like I was trying to cover some of the principles or some of the core practices we should follow.

00:30:00.400 --> 00:30:11.650  
Patil, Tatyasaheb  
While implementing this, and like with increasing number of like applications now earlier we have single application and like some layers we are more secure with but with microservices every single unit we need to secure.

00:30:12.570 --> 00:30:25.450  
Patil, Tatyasaheb  
And that's why I security is carrying more importance nowadays and with stateless a status implementation of this. This microservices, we are more very much better with this implementing the security.

00:30:26.840 --> 00:30:28.170  
Patil, Tatyasaheb  
And there are few like.

00:30:29.030 --> 00:30:37.390  
Patil, Tatyasaheb  
No, the group in in next slides we're going to talk about some of the some of the best practices or some of the ways how you can implement the security.

00:30:39.580 --> 00:30:44.130  
Patil, Tatyasaheb  
I think this is water like I wanted to cover today and like like anybody has any questions on this?

00:30:47.670 --> 00:30:51.060  
Patil, Tatyasaheb  
Orange will talk further on this different protocols.

00:30:52.760 --> 00:30:53.650  
Patil, Tatyasaheb  
But before that.

00:30:55.580 --> 00:30:57.030  
Patil, Tatyasaheb  
Any thoughts, any questions?

00:31:03.240 --> 00:31:06.830  
Swamy Bread, Yerri  
Oh God, this is swami.

00:31:07.580 --> 00:31:07.930  
Patil, Tatyasaheb  
Uh-huh

00:31:08.590 --> 00:31:20.670  
Swamy Bread, Yerri  
Internally complete, we maintain the private cloud and we we deployed the microservices in the private cloud. Not required. These kind of worth integrations or we need to Mustang should follow some protocol to.

00:31:21.380 --> 00:31:22.640  
Swamy Bread, Yerri  
Authentication purpose.

00:31:23.770 --> 00:31:25.640  
Patil, Tatyasaheb  
so in that case, yeah.

00:31:23.990 --> 00:31:25.270  
Swamy Bread, Yerri  
If in case private cloud.

00:31:27.260 --> 00:31:27.880  
Patil, Tatyasaheb  
Go ahead, sorry.

00:31:28.490 --> 00:31:40.890  
Swamy Bread, Yerri  
Yeah, if if in case of private cloud, if we deploy the microservices is we need the provider that security all the things because already in the private cloud we can maintaining right?

00:31:41.970 --> 00:32:09.700  
Patil, Tatyasaheb  
Yeah, so it it it depends. It depends like it's not mandate or but like I think we should be mandating that I feel like we should be mandating that and I think like Michael Nikhil have like one example on that. Like you know there's invest one product and like I think we're fires only managing the things. FIS only managing the server side all. Maybe Nikki would you like to explain like how we manage that some of the private information stored at the end or how they publicly available like some of the public services and some of the private services.

00:31:41.980 --> 00:31:42.820  
Swamy Bread, Yerri  
His mandate.

00:32:11.970 --> 00:32:39.380  
Damle, Nikhil  
Right, right and and Swami. Good question, you know because because like what you're saying is within the private cloud, you're already, you know, within secure kind of environment and within the trust boundaries and all that right? So so four inter services communication I guess right? The question is maybe 1 microservice application calling another Microsoft is application but within the same trust boundary should we even enforce any kind of you know security authentication is that?

00:32:40.300 --> 00:32:41.020  
Damle, Nikhil  
The question load.

00:32:40.470 --> 00:32:56.720  
Swamy Bread, Yerri  
Yes yes yes correct? Yeah I heard that in the microservices API gateway something entry point providing the security for API Gateway and internal microservices cannot miss need not require the securities.

00:32:41.940 --> 00:32:42.510  
Damle, Nikhil  
Right, right?

00:32:43.880 --> 00:32:44.350  
Damle, Nikhil  
Yeah.

00:32:57.440 --> 00:33:07.200  
Swamy Bread, Yerri  
So like that I heard in the microservices, so he's still true or we need to provide the basic level of authentication for each microservices, like I'm not sure with them.

00:33:08.910 --> 00:33:14.940  
Damle, Nikhil  
It's it's always good to provide some layer of security you know, even within within the.

00:33:15.930 --> 00:33:35.990  
Damle, Nikhil  
Within the private cloud environment, you know it's good to provide some basic layer of security, but you right, you know, sometimes the API gateway takes care of authentication for all the requests coming from outside world in into the private cloud. So in that case you know the outside the external calls are already taken care of.

00:33:36.750 --> 00:33:44.880  
Damle, Nikhil  
Internally, you know internally when one Microsoft is calling another microservice application. Maybe you know might want to take look. Take a look at.

00:33:46.010 --> 00:33:51.660  
Damle, Nikhil  
What kind of you know access privileges does this? You know user really have? Because although the call is.

00:33:52.280 --> 00:34:00.650  
Damle, Nikhil  
Uh, you know, initiated from an external user and you know it's getting authenticated there, but in terms of authorization, maybe you want to check whether that user.

00:34:01.520 --> 00:34:12.930  
Damle, Nikhil  
Really has privileges, so maybe the user calls one microservices application and from within that microservices application. If giving a call to another, you know.

00:34:13.750 --> 00:34:22.460  
Damle, Nikhil  
Endpoint on another application, but maybe within the same private cloud. It's still I guess you need to check for access privileges. Authorization rules.

00:34:23.230 --> 00:34:25.420  
Damle, Nikhil  
That kind of security, I guess.

00:34:25.420 --> 00:34:25.720  
Swamy Bread, Yerri  
No.

00:34:26.730 --> 00:34:29.290  
Damle, Nikhil  
Maybe authentication is already taken care of, but.

00:34:30.090 --> 00:34:30.540  
Damle, Nikhil  
Yeah.

00:34:31.490 --> 00:34:31.810  
Durga, Amar  
Yes.

00:34:32.390 --> 00:34:32.880  
Swamy Bread, Yerri  
Yeah, yeah.

00:34:32.640 --> 00:34:36.000  
Durga, Amar  
Other thing is can be taken at API gateway level.

00:34:32.760 --> 00:34:33.160  
Patil, Tatyasaheb  
Yeah.

00:34:36.390 --> 00:34:40.700  
Durga, Amar  
Printer communications still need some kind of operation.

00:34:45.520 --> 00:34:46.110  
Durga, Amar  
Correctly.

00:34:45.760 --> 00:34:47.530  
Damle, Nikhil  
So good question. So I mean, yeah, thank you.

00:34:45.880 --> 00:34:46.270  
Swamy Bread, Yerri  
P.

00:34:46.470 --> 00:34:46.810  
Patil, Tatyasaheb  
Yeah.

00:34:47.430 --> 00:34:48.260  
Swamy Bread, Yerri  
Yes, thank you.

00:34:51.770 --> 00:34:52.150  
Patil, Tatyasaheb  
Yeah.

00:34:53.140 --> 00:34:54.160  
Patil, Tatyasaheb  
Thanks Nikki for that.

00:35:00.840 --> 00:35:07.250  
Patil, Tatyasaheb  
Yeah, with this like sheep, there are no more questions. Let me hand over to like my colleague Nikhil and like he will.

00:35:02.360 --> 00:35:02.690  
Damle, Nikhil  
OK.

00:35:08.070 --> 00:35:17.680  
Patil, Tatyasaheb  
Continue on some of the security protocols and I talked like there are different different ways to implement the security and you will more elaborate on that and with some some of the examples on that.

00:35:19.710 --> 00:35:20.800  
Patil, Tatyasaheb  
Or to uniquely.

00:35:21.550 --> 00:35:22.010  
Patil, Tatyasaheb  
Thank you.

00:35:23.810 --> 00:35:26.200  
Damle, Nikhil  
Thank you, thank you Daddy. Let me share my screen.

00:35:38.410 --> 00:35:42.700  
Damle, Nikhil  
So I'm sharing the the PPT. Now I I think I hope it's visible.

00:35:43.860 --> 00:35:44.530  
Durga, Amar  
It's for me.

00:35:44.960 --> 00:35:45.210  
Damle, Nikhil  
OK.

00:35:48.060 --> 00:35:51.960  
Damle, Nikhil  
So yeah, good morning, all you know I'm on the kill so.

00:35:53.030 --> 00:36:01.680  
Damle, Nikhil  
I think like Tatya mentioned, you know the importance of security in general land on specially for microservices applications as well.

00:36:02.490 --> 00:36:03.080  
Damle, Nikhil  
Uhm?

00:36:03.950 --> 00:36:16.010  
Damle, Nikhil  
So we've looked at, you know, the the principles or the the really the importance of security. And now we can actually look at, you know how the Spring Framework you know provides a lot of inbuilt.

00:36:16.690 --> 00:36:20.170  
Damle, Nikhil  
Uh, capabilities to, you know, implement.

00:36:20.860 --> 00:36:31.330  
Damle, Nikhil  
Uh, different types of security mechanisms you know depending on what level of security or what type of design and architecture we are planning to.

00:36:32.620 --> 00:36:34.830  
Damle, Nikhil  
I'll create for this microservices ecosystem.

00:36:35.370 --> 00:36:39.770  
Damle, Nikhil  
Uh, Spring really provides a lot of inbuilt capabilities, you know so.

00:36:40.730 --> 00:37:04.740  
Damle, Nikhil  
I think the team for in the last couple of days and even today has been that you know we try to first look at the principles. You know the the design philosophy is, you know, the the importance of doing it right and then we, you know, look at how spring boot in general you know helps implement those principles and you know design philosophies right so.

00:37:05.830 --> 00:37:08.520  
Damle, Nikhil  
Continuing with that same theme, right like we've looked at.

00:37:10.170 --> 00:37:12.680  
Damle, Nikhil  
Some of the core principles in of security like.

00:37:13.210 --> 00:37:22.420  
Damle, Nikhil  
UH-10 dictation authorization. You know least privileges and and all that. And let's see how really you know spring provides.

00:37:23.060 --> 00:37:27.530  
Damle, Nikhil  
Uh, capabilities out of the box that you know you don't have to.

00:37:28.810 --> 00:37:37.380  
Damle, Nikhil  
Really implement each and every aspect of of those security mechanisms, again by hand. You know a lot of.

00:37:38.660 --> 00:37:42.590  
Damle, Nikhil  
Protocols and mechanisms have already been supported by spring.

00:37:43.120 --> 00:37:44.870  
Damle, Nikhil  
Uh, inherently.

00:37:46.240 --> 00:37:48.010  
Damle, Nikhil  
But looking at you know what are the.

00:37:48.840 --> 00:37:50.060  
Damle, Nikhil  
The min.

00:37:51.150 --> 00:38:02.740  
Damle, Nikhil  
All types of you know security protocols or methods that we normally employ. You know to secure our services and endpoints and microservices applications in general.

00:38:04.260 --> 00:38:22.910  
Damle, Nikhil  
So I I just I just listed out these four. But you know, like we normally know basic authentication right? Which is again based on the username and password. The user can actually try to access a particular resource on an application or a microservices endpoint, you know?

00:38:23.710 --> 00:38:26.360  
Damle, Nikhil  
So that's the most widely known, although you know.

00:38:27.350 --> 00:38:34.410  
Damle, Nikhil  
It sounds very simple. Of course there's a lot of you know background code that needs to be written in order to implement that so.

00:38:35.000 --> 00:38:39.570  
Damle, Nikhil  
Uh, the spring security feature, you know, really provides all of that.

00:38:40.800 --> 00:38:45.170  
Damle, Nikhil  
Mechanisms in build a variation of basic auth.

00:38:45.900 --> 00:38:46.770  
Damle, Nikhil  
Is, uh.

00:38:47.710 --> 00:38:53.800  
Damle, Nikhil  
The MD5 Digest based authentication so we know this is another way of.

00:38:54.530 --> 00:39:19.640  
Damle, Nikhil  
Accessing our services again by, you know, providing some authentication information on the request headers so when we send a HTTP request to an endpoint and we can include some authorization authentication headers really in that HTTP request and will also look at that MD5 Digest based in some detail in the in the later slides.

00:39:20.910 --> 00:39:28.110  
Damle, Nikhil  
And then you know these days you know the overt to it is very popular and I think it's it's.

00:39:28.870 --> 00:39:29.410  
Damle, Nikhil  
Uh.

00:39:30.480 --> 00:39:43.660  
Damle, Nikhil  
It's it's implemented in in a variety of ways, really. You know. Although it just says over two, there are a lot of different flavors and different flows. You know within a worth 2. So we look at.

00:39:44.900 --> 00:40:14.590  
Damle, Nikhil  
What that means and know how it works and then of course there's a variation of that which is the 4th one. It's called open ID Connect, so it's kind of a variation of over 2 minutes, mostly again used for authorization as well as authentication, you know, so the the Open ID Connect is implemented using both the worth 2 and Open ID Connect are implemented using tokens. You know there's typically called access tokens or.

00:40:15.370 --> 00:40:18.600  
Damle, Nikhil  
JWT tokens, if you heard of JWT already, I guess.

00:40:20.080 --> 00:40:25.710  
Damle, Nikhil  
So these are the main mainly these four that we can look at today.

00:40:26.390 --> 00:40:35.710  
Damle, Nikhil  
But, uh, just a general question. You know any any other security mechanisms you know, apart from this that you are aware of and you know?

00:40:36.760 --> 00:40:38.610  
Damle, Nikhil  
If you want do through some light on that.

00:40:44.800 --> 00:40:45.330  
Damle, Nikhil  
Turf.

00:40:45.770 --> 00:40:49.310  
Thirugnanam, Kumaran  
And hired a SAML being used in one of the project.

00:40:48.430 --> 00:40:48.980  
Damle, Nikhil  
Salmon.

00:40:50.140 --> 00:40:50.640  
Damle, Nikhil  
Correct?

00:40:50.380 --> 00:40:50.690  
Thirugnanam, Kumaran  
Yeah.

00:40:51.810 --> 00:40:55.650  
Thirugnanam, Kumaran  
But not much idea on that. But yeah, I've consumed it, but not much idea.

00:40:51.920 --> 00:40:52.550  
Damle, Nikhil  
So I.

00:40:54.730 --> 00:40:55.170  
Damle, Nikhil  
Correct?

00:40:57.340 --> 00:40:59.510  
Damle, Nikhil  
Correct, correct saml? I was just.

00:41:00.110 --> 00:41:09.030  
Damle, Nikhil  
Yeah, I'm going to mention Simulaid included here, specially because you know, Samuel stands for the security Assertion Markup language, so again, it's it's.

00:41:09.120 --> 00:41:31.450  
Damle, Nikhil  
So security mechanism or a security protocol, but it's normally used in, you know, single sign on kind of applications where you know you have a user session and web applications you know with the UI and typically when you use a SAML token you verify it once and then start creating an HTTP session there on so.

00:41:32.270 --> 00:41:38.880  
Damle, Nikhil  
That's why you know if I haven't added SAML here just because you know, with the microservices layer.

00:41:39.530 --> 00:41:40.540  
Damle, Nikhil  
Mostly.

00:41:41.730 --> 00:41:44.740  
Damle, Nikhil  
The authentication or authorization is, you know.

00:41:45.410 --> 00:42:01.190  
Damle, Nikhil  
Per request, so it's it's mostly stateless and will come to the. I don't know why it the the stateless point is very important, but that's why you know maybe cimal have included in here. Of course spring security does support, you know the SAML protocol as well.

00:42:04.060 --> 00:42:04.430  
Damle, Nikhil  
OK.

00:42:09.590 --> 00:42:11.720  
Damle, Nikhil  
OK, so moving on, you know, let's look at.

00:42:13.550 --> 00:42:22.400  
Damle, Nikhil  
Spring security and I I think you know many of you who are already working on the spring based you know or spring boot page.

00:42:25.630 --> 00:42:27.560  
Damle, Nikhil  
Hi, I think there's some background noise.

00:42:30.760 --> 00:42:32.250  
Damle, Nikhil  
Anything going mute P OK.

00:42:35.580 --> 00:42:36.970  
Damle, Nikhil  
OK, sorry so.

00:42:39.930 --> 00:42:50.320  
Damle, Nikhil  
You know spring security. Like we said, you know already provides a lot of inbuilt mechanisms to implement these security protocols, right so?

00:42:51.490 --> 00:43:19.310  
Damle, Nikhil  
Like you know, if you've already used spring boot or spring based applications, you might have realized that you know you don't have to really implement each and every you know protocol or each and every mechanics of the protocol by hand you know. Or you can customize definitely what is provided by spring spring security, but you don't have to really code it from scratch.

00:43:20.530 --> 00:43:38.850  
Damle, Nikhil  
So spring security is, you know, it's a powerful land in highly customizable authentication and access control framework. So it's you know for any spring based applications it's the default. You know security framework that has to be used or the most recommended right because.

00:43:40.000 --> 00:43:47.030  
Damle, Nikhil  
Don't you spring security? Then you have to write a lot of code to, you know, implement those security protocols.

00:43:48.560 --> 00:43:55.690  
Damle, Nikhil  
So it it it focuses on, you know, both authentication and authorization to you know Java based applications so.

00:43:56.280 --> 00:44:05.320  
Damle, Nikhil  
Uh, it has a support for, you know, all these different various protocols as well as you know, trend, dictation and authorization.

00:44:05.370 --> 00:44:08.020  
Damle, Nikhil  
No, for the incoming requests.

00:44:09.490 --> 00:44:13.780  
Damle, Nikhil  
And why is it so powerful that you know? Because it can be, you know.

00:44:13.840 --> 00:44:47.280  
Damle, Nikhil  
Will extend it to meet customize customized requirements. Right so although you know like going with that spring boot philosophy that you know everything is ready to use, you know like that you're mentioning it's it's all there. You know it's ready to use and we just have to start using it. But you know it comes all with default configuration, right? For example, trying to use basic authentication using spring security. It comes with all those default configurations and if you want to customize, you know we have some peculiar.

00:44:47.460 --> 00:45:04.300  
Damle, Nikhil  
Authentication requirements or you want to add maybe an additional layer or you know verify a few more attributes on those tokens or on this authorization headers so you can really customize it very easily there are, you know, a lot of hooks available.

00:45:07.020 --> 00:45:08.440  
Damle, Nikhil  
Excuse the background noise.

00:45:09.080 --> 00:45:13.790  
Damle, Nikhil  
OK, there are lot of hooks available with spring security that you know can be used to.

00:45:15.120 --> 00:45:18.300  
Damle, Nikhil  
Uh, customize these amacan isms.

00:45:19.490 --> 00:45:24.180  
Damle, Nikhil  
And still, you know, use the standard mechanism, but have some custom hooks in there, right so?

00:45:25.380 --> 00:45:31.550  
Damle, Nikhil  
So let's look at, you know, spring security in terms of, you know, a pictorial view. So you know this.

00:45:33.300 --> 00:45:41.780  
Damle, Nikhil  
This blue box you know in between inside you know is is really RFC or microservices application. You know spring boot based application.

00:45:42.850 --> 00:45:47.650  
Damle, Nikhil  
And you know this. This dotted line. Oh this this is the spring security layer so.

00:45:49.280 --> 00:45:54.930  
Damle, Nikhil  
Uh, whenever we enable, you know spring security on spring boot application.

00:45:56.560 --> 00:46:01.680  
Damle, Nikhil  
It comes into play by default. You know, by default it starts up.

00:46:03.180 --> 00:46:15.780  
Damle, Nikhil  
Or, or you know, adding authentication layer you know or security rarely for each incoming request, so you know as as soon as we add the spring boot starter dependency to spring boot application.

00:46:17.050 --> 00:46:25.980  
Damle, Nikhil  
It comes into action, you know. And of course we have to then configure it a bit and you know maybe customize it a bit as per our needs but.

00:46:27.410 --> 00:46:34.960  
Damle, Nikhil  
Pointers that you know as soon as we add the dependency in our POM dot XML for Spring Boot starter security.

00:46:35.530 --> 00:47:05.030  
Damle, Nikhil  
It comes into play, know immediately and what does it do? We know it's it acts as interceptor. You know for each incoming requests, by default you know each incoming request. It could be, you know, request coming from some third party applications or some workflow applications. Or you know programmatic access where there's a programmatic consumer trying to call our microservices or from a UI. So any any kind of requests you know coming into our Springwood application, the spring security layer.

00:47:05.890 --> 00:47:13.780  
Damle, Nikhil  
Here's the first one you know to intercept, you know, so as good as you know, if you draw an analogy there that you know maybe someone is entering our our.

00:47:14.440 --> 00:47:33.830  
Damle, Nikhil  
Uh, you know building or you know our society premises right in and there's a security guard you know at at the entrance and you know you can't bypass that guy. You know the guard will ask, you know who you are and what do you want? You know. So those two questions are really asked by the spring security layer. You know every time any request has to come in.

00:47:34.770 --> 00:47:42.490  
Damle, Nikhil  
Uh, I will try to ask. You know who who you are, you know. So you have tried to authenticate and as well as know what, what do you want to access?

00:47:44.250 --> 00:47:58.720  
Damle, Nikhil  
And of course all these things will need to be configured on the spring security layer according to our requirements, but the security layer spring security layer will essentially, you know, stop every request and check for you know.

00:48:00.180 --> 00:48:05.800  
Damle, Nikhil  
If it's a valid request or not in terms of is it authentic users trying to come in?

00:48:08.620 --> 00:48:18.310  
Damle, Nikhil  
If you, if you look at here you know all these are straight list rest requests. OK so I'll also try to talk about you know why stateless?

00:48:20.270 --> 00:48:22.030  
Damle, Nikhil  
Requests are important, you know. But

00:48:24.050 --> 00:48:29.100  
Damle, Nikhil  
what happens is you know, normally in a web application we typically.

00:48:30.260 --> 00:48:35.890  
Damle, Nikhil  
Uh, you know, authenticate once you know using the username password and then we tried creating an HTTP session.

00:48:36.850 --> 00:48:42.170  
Damle, Nikhil  
And from there on, you know each subsequent HTTP request need not, really, you know.

00:48:42.740 --> 00:48:57.930  
Damle, Nikhil  
Uh, provide its username and password every time. If it's a normal web application, knows where, maybe there is a could be a you know. Starts based back end or you know any other spring MVC braised back end but.

00:48:58.940 --> 00:49:02.340  
Damle, Nikhil  
What happens in in? In that case is that you know, let's say.

00:49:03.530 --> 00:49:13.160  
Damle, Nikhil  
This application is deployed into a cluster. Typically like we talked about, right? If it's illegal, Kubernetes managed, you know, clustered environment where.

00:49:13.930 --> 00:49:21.030  
Damle, Nikhil  
Maybe you want to auto scale, you know, let's say this microservice application you want to create five different instances of it at runtime. You know depending on the load.

00:49:21.840 --> 00:49:24.990  
Damle, Nikhil  
On on the system. So if you create five different.

00:49:26.460 --> 00:49:40.300  
Damle, Nikhil  
All instances of the same microservices application, but if you have a, you know a stateful kind of architecture where once the user logs in they create a something called as a session ID, right? If you've heard of that and then every time the.

00:49:40.710 --> 00:49:44.040  
Damle, Nikhil  
Oh, the same session is trying to request.

00:49:44.980 --> 00:49:50.450  
Damle, Nikhil  
I'll send a request to this microservice application. It's accompanied by the J session ID.

00:49:51.750 --> 00:50:11.520  
Damle, Nikhil  
Now that the J session ID is adding, identifying a session which is like an in memory, you know object or in memory context of that user inside a particular JVM right inside a single JVM. There is this session information stored. So if there are five instances you know of this microservices application.

00:50:12.530 --> 00:50:18.790  
Damle, Nikhil  
Uh, it's necessary that you know if it's so stateful request it has to hit that particular instance only.

00:50:19.530 --> 00:50:24.820  
Damle, Nikhil  
Uh, in order to, you know, respect that session because otherwise if it hits some other.

00:50:25.420 --> 00:50:56.440  
Damle, Nikhil  
Uh, or some other instance in the cluster and that instance, or have any information of this session ID that you know it becomes like a invalid invalid request and they'll show you an error back to the user. But what it means is and there's a load balancer in between and then the the load balancer has to really ensure that you know if you've heard of session affinity and sticky sessions. The load balancer as to really ensure that this request reaches that particular instance every time you know, because there is a, it's a stateful.

00:50:56.950 --> 00:51:06.550  
Damle, Nikhil  
Request, whereas if it was a stateless request, the load balancer is free to divide it. You know in terms of round Robin or based on you know.

00:51:07.550 --> 00:51:39.120  
Damle, Nikhil  
Uh, any of those load balancing algorithms it can efficiently? You know, route the request to any of the instances. If it was a stateless request, you know, and that's the you know power of actually having these microservices applications into a. You know, maybe a Cuban. It is environment which can be scaled horizontally without you know, even if you scale, you know, let's say you create 10 instances, but if you have a very stateful kind of interaction with this, you know it's very ineffective in terms of balancing the load.

00:51:39.810 --> 00:51:58.680  
Damle, Nikhil  
Because the stateful request has to always reach that particular instance which has the session information for that session ID, right? So? So that's you know, I just wanted to emphasize the importance of having these stateless requests every time you know, especially in the context of microservices applications.

00:51:59.980 --> 00:52:10.960  
Damle, Nikhil  
So that you know you can efficiently scale and still get all the, you know, benefits of of efficient load balancing algorithms that the load is evenly distributed.

00:52:12.230 --> 00:52:13.600  
Damle, Nikhil  
On the various instances.

00:52:18.560 --> 00:52:18.990  
Damle, Nikhil  
K.

00:52:21.800 --> 00:52:24.510  
Damle, Nikhil  
So let's look at basic authentication.

00:52:25.840 --> 00:52:37.150  
Damle, Nikhil  
I think, uh, I think most of us already know, you know that this is the most straightforward method where you know the user sends in the username and password into the request header.

00:52:38.460 --> 00:52:58.370  
Damle, Nikhil  
OK, so the request header you know has has a predefined standard header called as in authorization header and the username password is actually, you know, encoded and together into a base 64 encoded format and placed on that authorization header.

00:52:59.060 --> 00:53:03.510  
Damle, Nikhil  
And send with, you know, uh, there should be request to the microservices application.

00:53:04.960 --> 00:53:14.010  
Damle, Nikhil  
No, no. It it better be stateless. Again like I emphasized, if it's a stateless, you know we'll get all that advantages of clustering and efficient load balancing.

00:53:14.820 --> 00:53:20.390  
Damle, Nikhil  
Uh, so let's say on every request you know you've sending this username and password.

00:53:23.700 --> 00:53:28.670  
Damle, Nikhil  
This method of course does not require any cookies and session ID's and and all that you know, because again.

00:53:29.290 --> 00:53:41.510  
Damle, Nikhil  
There's no complex handshakes which you know there might be some handshakes in in the other security mechanisms that we will look at later today, but this is the most.

00:53:42.430 --> 00:53:50.280  
Damle, Nikhil  
A straightforward you know mechanism and like this is an example here that you know this is the authorization header and you know.

00:53:51.230 --> 00:53:52.190  
Damle, Nikhil  
It's just actually.

00:53:52.650 --> 00:53:53.130  
Damle, Nikhil  
Uh.

00:53:54.370 --> 00:54:00.380  
Damle, Nikhil  
So he's basic and this is the encoded in base 64 encoded value of the username and password.

00:54:01.440 --> 00:54:02.280  
Damle, Nikhil  
So when this.

00:54:03.040 --> 00:54:09.280  
Damle, Nikhil  
Authorization is hilarious, sent you know. Then the microservices application or the spring boot application.

00:54:11.040 --> 00:54:21.450  
Damle, Nikhil  
So if the if it's, you know, implementing basic authentication so that you know the spring security layer provides all the hooks and mechanisms needed, you know 2.

00:54:22.600 --> 00:54:25.570  
Damle, Nikhil  
Implement basic authentication in spring boot application.

00:54:26.760 --> 00:54:28.590  
Damle, Nikhil  
OK, So what what does it do, you know so?

00:54:30.050 --> 00:54:45.190  
Damle, Nikhil  
Again, this is a spring security layer, you know which comes into play. Any any request coming in it, just a diagrammatic representation that any request coming in will have to be accompanied by a username and a password, right? So in any request coming in.

00:54:45.850 --> 00:54:49.590  
Damle, Nikhil  
Uh, the security layer. The spring security layer will intercept it.

00:54:50.560 --> 00:54:59.790  
Damle, Nikhil  
Will you know a check for the validity of the username and password and if it's valid only, then let the?

00:55:00.740 --> 00:55:02.520  
Damle, Nikhil  
Only then let the request in.

00:55:04.170 --> 00:55:09.490  
Damle, Nikhil  
Right, so how does you know how does it? How does the spring security layer do all this links?

00:55:11.100 --> 00:55:16.330  
Damle, Nikhil  
So you know just a quick view of some of the you know.

00:55:17.150 --> 00:55:19.100  
Damle, Nikhil  
Classes which are provided.

00:55:20.130 --> 00:55:36.680  
Damle, Nikhil  
Uh, by spring security. So like we saw initially that you know spring security provides this implementation. You know, for these security mechanisms, like for example basic authentication, it provides this kind of a.

00:55:38.660 --> 00:56:03.570  
Damle, Nikhil  
Uh, implementation, but you can also, of course, you know. Add in your hooks because for example the username and password, right? So you you need to have your own, you know way of fetching the username and password from. Maybe you know your database, let's say and then you know, validate that against what the user has sent in, right? So although spring security provides these, you know set of classes.

00:56:04.740 --> 00:56:11.750  
Damle, Nikhil  
It's still, you know, needs some hooks in there, you know so that we can customize that bid for for our needs.

00:56:12.820 --> 00:56:16.970  
Damle, Nikhil  
So just a quick look at how spring security, really, you know, works.

00:56:18.300 --> 00:56:24.350  
Damle, Nikhil  
And this is true mainly for, you know, most of the security protocols that spring security supports.

00:56:25.940 --> 00:56:37.710  
Damle, Nikhil  
So you know the first thing that happens is you know there's an HTTP request coming in. There's a username, password, authentication token, object that is created based on, you know the authorization header.

00:56:39.150 --> 00:56:44.940  
Damle, Nikhil  
And there is something called as a filter. You know you might have heard of filters, right? These are like typical.

00:56:45.920 --> 00:56:52.430  
Damle, Nikhil  
Uh, concepts from, you know, the servlets and that kind of work right? Every web application we can define a filter.

00:56:53.630 --> 00:57:01.230  
Damle, Nikhil  
Filter will intercept each incoming request. In this case you know spring security also has something called as an authentication filter.

00:57:02.270 --> 00:57:06.160  
Damle, Nikhil  
And then the filter, really, you know, delegates.

00:57:07.620 --> 00:57:12.900  
Damle, Nikhil  
Oh the the the job of authentication to something that doesn't authentication manager.

00:57:13.980 --> 00:57:15.740  
Damle, Nikhil  
Not authentication managers or you know.

00:57:15.820 --> 00:57:32.660  
Damle, Nikhil  
Uh is again a class which has the responsibility of figuring out you know what kind of authentication provider has to be used. So the authentication provider he'll here really is an interface and it you know it. There are various implementations that spring security provides.

00:57:33.830 --> 00:57:36.740  
Damle, Nikhil  
And then it actually uses the user details service.

00:57:37.950 --> 00:57:43.820  
Damle, Nikhil  
To fetch the user details from the system and then you know the authentication provider will do the.

00:57:43.870 --> 00:57:53.840  
Damle, Nikhil  
Oh oh oh the matching or you know, the verification of the user name password sent in by the user versus the ones which you know exists in the system.

00:57:54.840 --> 00:57:58.260  
Damle, Nikhil  
And then it will return back the control back to the manager and you know the filter.

00:57:59.980 --> 00:58:08.810  
Damle, Nikhil  
So this is kind of, you know, the basic set of classes that are provided by spring security. Now you know we'll have to, of course.

00:58:10.150 --> 00:58:19.540  
Damle, Nikhil  
Add a few you know hooks in there and will also look at an example in in some time of you know how to really make use of this spring security mechanism.

00:58:20.810 --> 00:58:43.370  
Damle, Nikhil  
But definitely the user details service. You know it's user details service is something that you know we as the microservices application developers might have to implement, you know so that we fetched the relevant username and the and the password. Let's say for that particular user name and provide it back to the provider so that you know the provider will then.

00:58:44.310 --> 00:58:54.130  
Damle, Nikhil  
Compare it with the ones which are sent from the user and then complete the authentication so the user details service is something you know mainly that has to be.

00:58:54.990 --> 00:59:02.240  
Damle, Nikhil  
Implemented by the user, but that's the only thing that has to be implemented in. Everything else is really, you know, provided out of the box.

00:59:03.880 --> 00:59:05.780  
Damle, Nikhil  
The provider, you know. I've just added a few more.

00:59:05.840 --> 00:59:10.720  
Damle, Nikhil  
Uh, uh classes here that spring security this to you know.

00:59:11.500 --> 00:59:15.060  
Damle, Nikhil  
Complete the picture here, for example, the authentication provider you know can.

00:59:16.330 --> 00:59:24.540  
Damle, Nikhil  
Make use of different types of providers that spring security already has implemented, and typically you know the Dow. The authentication provider is mostly used.

00:59:25.370 --> 00:59:29.900  
Damle, Nikhil  
Uh, because you know that the authentication provider is is where it actually tries to.

00:59:30.920 --> 00:59:37.290  
Damle, Nikhil  
I'll fetch the user details from the database and you know, compare it. So that's the most widely used.

00:59:38.800 --> 00:59:55.730  
Damle, Nikhil  
The user little service as well, you know. Like I said, we can really customize that. You know to our because it could be an in memory user details service. Maybe it's maybe the user details are cashed somewhere or you know we're fetching the user details from another service. Maybe it could be a custom user details service, right so?

00:59:57.360 --> 00:59:58.230  
Damle, Nikhil  
These are all the.

00:59:59.660 --> 01:00:06.680  
Damle, Nikhil  
You know classes that spring security provides and there are a few a couple of things that you know the user will have to.

01:00:07.140 --> 01:00:16.270  
Damle, Nikhil  
Uh, kind of implemented, specially the user details service so that you know this whole mechanism comes to life and works as expected, right?

01:00:19.370 --> 01:00:27.740  
Damle, Nikhil  
So I think this is the you know, uh, concept of basic authentication so far will of course look at the other authentication mechanisms.

01:00:28.310 --> 01:00:28.930  
Damle, Nikhil  
Uh.

01:00:29.700 --> 01:00:34.090  
Damle, Nikhil  
But any questions so far or anything that anyone wants to?

01:00:35.560 --> 01:00:38.670  
Damle, Nikhil  
Oscar, you know, suggest or comment on.

01:00:57.950 --> 01:01:12.820  
Damle, Nikhil  
We can look at a quick demo of how basic authentication can be implemented. You know, with with like minimal code using the spring boot application mechanism and spring security. You know framework in awhile.

01:01:20.720 --> 01:01:22.570  
Damle, Nikhil  
And maybe we could take a quick.

01:01:22.620 --> 01:01:22.900  
Damle, Nikhil  
So.

01:01:23.790 --> 01:01:28.130  
Damle, Nikhil  
Quick break as well, you know before we you know, try to look at that example.

01:01:30.010 --> 01:01:33.620  
Damle, Nikhil  
Is it a good time? Maybe you know, take a quick break or.

01:01:32.050 --> 01:01:32.390  
Subramanian Kamatchi, Gobi Ganesh  
So.

01:01:35.780 --> 01:01:36.140  
Durga, Amar  
Yes.

01:01:36.020 --> 01:01:37.150  
Damle, Nikhil  
Yeah, any questions.

01:01:36.420 --> 01:01:36.670  
Subramanian Kamatchi, Gobi Ganesh  
Yeah.

01:01:36.620 --> 01:01:37.670  
Durga, Amar  
So let's take a break.

01:01:36.930 --> 01:01:37.460  
Seniappan, Yuvarani  
Yes.

01:01:42.190 --> 01:01:43.130  
Durga, Amar  
Yes, yes.

01:01:43.410 --> 01:01:44.560  
Durga, Amar  
Yeah, don't wait.

01:01:45.960 --> 01:01:48.370  
Durga, Amar  
They think so whether shipping 50.

01:01:47.080 --> 01:01:47.810  
Damle, Nikhil  
10:50

01:01:51.950 --> 01:01:53.340  
Durga, Amar  
Yeah now 1040 days.

01:01:54.440 --> 01:01:57.320  
Damle, Nikhil  
yeah, let's come back at 10:50 or 55.

01:01:54.630 --> 01:01:55.340  
Durga, Amar  
The deck.

01:01:57.400 --> 01:01:57.710  
Durga, Amar  
Right?

01:01:58.670 --> 01:01:59.710  
Damle, Nikhil  
55 OK.

01:02:00.500 --> 01:02:02.220  
Durga, Amar  
Let's come back at 10:55.

01:02:00.560 --> 01:02:00.990  
Patil, Tatyasaheb  
Yep.

01:02:02.750 --> 01:02:03.780  
Damle, Nikhil  
Sure, sure, OK.

01:02:04.100 --> 01:02:04.570  
Seniappan, Yuvarani  
OK.

01:02:05.680 --> 01:02:06.040  
Patil, Tatyasaheb  
Yeah.

01:02:07.110 --> 01:02:07.580  
Patil, Tatyasaheb  
Thank you.

01:02:07.360 --> 01:02:07.920  
Durga, Amar  
Thank you.

01:02:10.750 --> 01:02:13.040  
Damle, Nikhil  
OK, so let's meet back at 10:55 then.

01:17:26.540 --> 01:17:27.440  
Damle, Nikhil  
OK hello.

01:17:30.870 --> 01:17:31.660  
Damle, Nikhil  
Like so.

01:17:31.120 --> 01:17:31.590  
Patil, Tatyasaheb  
Hi.

01:17:31.920 --> 01:17:32.430  
Seniappan, Yuvarani  
K.

01:17:35.810 --> 01:17:37.360  
Damle, Nikhil  
OK, so let's so.

01:17:40.670 --> 01:17:44.660  
Damle, Nikhil  
Let's pick up from you know where we had left off before the break, so.

01:17:45.800 --> 01:17:47.000  
Damle, Nikhil  
I'll share my screen.

01:17:47.350 --> 01:17:47.820  
Damle, Nikhil  
Ah.

01:17:57.980 --> 01:17:59.330  
Damle, Nikhil  
So you know we looked at.

01:18:00.800 --> 01:18:05.660  
Damle, Nikhil  
Just to recap, you know quickly we looked at, you know, the spring security, uh?

01:18:07.590 --> 01:18:08.310  
Damle, Nikhil  
Larry no.

01:18:08.370 --> 01:18:16.080  
Damle, Nikhil  
Well, what does IT support? You know it kind of supports various security protocols in built and you know.

01:18:16.800 --> 01:18:29.080  
Damle, Nikhil  
Uh, the stateless rest requests is is the most recommended you know, specially for a microservices based ecosystem where you know you can have horizontally scalable microservices applications.

01:18:31.510 --> 01:18:35.040  
Damle, Nikhil  
I think there's some background noise. Maybe can mute OK?

01:18:37.700 --> 01:18:38.020  
Damle, Nikhil  
OK.

01:18:42.050 --> 01:18:59.920  
Damle, Nikhil  
And then you know we started looking at basic authentication, so we know like we know there's a standard HTTP header called authorization and you know when we put the basic auth information like the username, password on that header, spring security, will you know can be.

01:19:01.200 --> 01:19:04.330  
Damle, Nikhil  
Uh utilized, you know, with all its kind of classes.

01:19:05.010 --> 01:19:06.740  
Damle, Nikhil  
Uh, already provided.

01:19:07.710 --> 01:19:12.470  
Damle, Nikhil  
Uh, blimp lament basic authentication, you know before it reaches our.

01:19:13.310 --> 01:19:16.380  
Damle, Nikhil  
Actually, you know microservices endpoint, right so?

01:19:17.600 --> 01:19:24.520  
Damle, Nikhil  
So let's look at a quick example. Again, we have, you know, an example from our.

01:19:24.580 --> 01:19:26.940  
Damle, Nikhil  
There's an application here.

01:19:29.020 --> 01:19:32.950  
Damle, Nikhil  
So let's start with really no. Looking at the Palm dot XML.

01:19:34.390 --> 01:19:36.460  
Thirugnanam, Kumaran  
Uh, and he killed on a shared screen move.

01:19:35.760 --> 01:19:36.040  
Damle, Nikhil  
Yeah.

01:19:37.820 --> 01:19:39.760  
Damle, Nikhil  
Oh sorry, I didn't share my screen.

01:19:39.450 --> 01:19:40.890  
Patil, Tatyasaheb  
No, no, you already shared.

01:19:43.440 --> 01:19:45.750  
Durga, Amar  
I can see you with a STS Nikhil.

01:19:47.000 --> 01:19:49.740  
Thirugnanam, Kumaran  
OK, maybe I'll try to reconnect, yeah?

01:19:47.030 --> 01:19:47.620  
Patil, Tatyasaheb  
Farmer

01:19:50.210 --> 01:19:50.550  
Patil, Tatyasaheb  
yeah.

01:19:50.420 --> 01:19:52.680  
Damle, Nikhil  
Should I try again? Maybe I'll try again, OK?

01:19:53.710 --> 01:19:54.240  
Patil, Tatyasaheb  
Yes.

01:19:54.060 --> 01:19:55.630  
Damle, Nikhil  
I'll, I'll try sharing.

01:19:54.910 --> 01:19:57.200  
Patil, Tatyasaheb  
All of this, can you see CS?

01:19:57.860 --> 01:19:59.990  
Swamy Bread, Yerri  
Yeah, we are able to see the screen Nikhil.

01:20:00.320 --> 01:20:01.520  
Mourya, Gourav  
I can see Nikhil again.

01:20:01.740 --> 01:20:02.830  
Damle, Nikhil  
Oh OK, OK.

01:20:02.540 --> 01:20:04.030  
Thirugnanam, Kumaran  
Oh, that's fine. I'll try to reconnect.

01:20:04.350 --> 01:20:05.300  
Damle, Nikhil  
Sure, sure sure.

01:20:05.280 --> 01:20:05.630  
Patil, Tatyasaheb  
Yeah.

01:20:11.080 --> 01:20:14.050  
Damle, Nikhil  
OK, so let's start with the Palm dot XML in the like.

01:20:15.310 --> 01:20:17.220  
Damle, Nikhil  
Like we mentioned that I don't know.

01:20:20.460 --> 01:20:24.450  
Damle, Nikhil  
So what we need really is a let me open the other poem, yeah?

01:20:26.340 --> 01:20:30.170  
Damle, Nikhil  
So you know what we need really is. You know, dependency.

01:20:31.050 --> 01:20:32.090  
Damle, Nikhil  
Called.

01:20:33.280 --> 01:20:40.660  
Damle, Nikhil  
You know spring Boot starter security, so that's it. You know this is the only dependency you know if you add this to your form.

01:20:41.460 --> 01:20:47.630  
Damle, Nikhil  
Uh, this spring boot security starts, you know, coming into play, you know, coming into action and.

01:20:48.570 --> 01:20:53.210  
Damle, Nikhil  
By default you know it will try to intercept each and every incoming request.

01:20:54.390 --> 01:20:57.760  
Damle, Nikhil  
And try to add authentication layer to it, right?

01:21:01.570 --> 01:21:06.580  
Damle, Nikhil  
If you know and we'll look at, of course, awards 2 and open ID Connect later.

01:21:07.730 --> 01:21:16.600  
Damle, Nikhil  
But just because we are in that same form, you know if you notice here, there's another. You know dependency is needed, you know to enable.

01:21:17.370 --> 01:21:37.420  
Damle, Nikhil  
The capabilities for Oauth or Openid Connect. You know, for this particular microservices application to act as over 2 resource server. But again I won't go into that right now. I think I will look at it later, but you know, like I said, just add spring Boot starter security as a dependency in your POM file.

01:21:38.580 --> 01:21:40.590  
Damle, Nikhil  
And you know you're all set, right? So?

01:21:42.250 --> 01:21:46.670  
Damle, Nikhil  
Let's look at a few other files that you know we have to.

01:21:47.420 --> 01:21:51.310  
Damle, Nikhil  
Uh, configure in order to you know, enable basic authentication.

01:21:53.560 --> 01:22:00.070  
Damle, Nikhil  
So this is a, uh, you know this is a class called, let's say in our case it's called basic security configuration.

01:22:01.720 --> 01:22:09.880  
Damle, Nikhil  
OK, before we even look at, you know, basic security configuration. There are few spring boot concepts that you know I like to highlight here.

01:22:10.580 --> 01:22:34.650  
Damle, Nikhil  
And specially some of these annotations, you know. So if you look at configuration, you know this is a special annotation that you know if you annotate a class with configuration, it means that you know this is really a bootstrap kind of a class that has to be instantiated before all of the classes are instantiated as spring beans as well. As you know, it can consider it can consist of, you know.

01:22:36.800 --> 01:22:39.450  
Damle, Nikhil  
Methods which implement the, you know the AED rate beam.

01:22:40.090 --> 01:22:53.080  
Damle, Nikhil  
Annotation that red bean means that you know you trying to instantiate a spring bean using a method so you know here you can actually add some kind of. For example, in this case you know this is AED rate being, so you're trying to expose.

01:22:53.730 --> 01:23:00.430  
Damle, Nikhil  
Or custom view, or 10 dictation provider. But you know trying to customize a bit before exposing that as a spring bean.

01:23:01.390 --> 01:23:03.100  
Damle, Nikhil  
So typically you know when we.

01:23:04.300 --> 01:23:19.300  
Damle, Nikhil  
Create a spring bean. We annotate that class with the Advair at component annotation and that becomes a spring bean by default, right? But this ad rate being is another way of, you know. Defining spring beans at runtime.

01:23:21.680 --> 01:23:25.720  
Damle, Nikhil  
Just add red bean. Annotation methods. Can only you know.

01:23:26.810 --> 01:23:31.830  
Damle, Nikhil  
This side in the class, which has configuration because you know that's like a special instruction to the spring boot.

01:23:32.390 --> 01:23:43.130  
Damle, Nikhil  
That you know you should bootstrap these kind of classes first before others, and then you know you can actually create these spring beans using the aderet beam.

01:23:45.580 --> 01:23:47.630  
Damle, Nikhil  
I don't patients, there's another.

01:23:48.620 --> 01:24:00.550  
Damle, Nikhil  
Interesting, uh annotation called enable web security so you know, this is again a special instruction to spring boot and this is needed because you know, this will actually again instruct spring boot.

01:24:01.100 --> 01:24:05.370  
Damle, Nikhil  
Uh, to enable or to, you know, use this configuration class.

01:24:06.140 --> 01:24:11.330  
Damle, Nikhil  
As a class, you know which can actually configure some kind of security mechanism.

01:24:12.080 --> 01:24:13.450  
Damle, Nikhil  
OK, and the.

01:24:17.520 --> 01:24:19.970  
Damle, Nikhil  
And and the important part here is that you know.

01:24:21.030 --> 01:24:27.910  
Damle, Nikhil  
This class should always extend something called as a web security configurer adapter, so that's like.

01:24:29.450 --> 01:24:37.170  
Damle, Nikhil  
Uh, a foundational class, or, you know, a framework class provided by spring boot in order to, you know, kind of.

01:24:37.840 --> 01:24:50.280  
Damle, Nikhil  
It's kind of the part of the you know the authentication mechanism, and it has a method called configure which needs to be overwritten for our customization. Sick, right? So we'll look at how we override that.

01:24:52.090 --> 01:24:57.340  
Damle, Nikhil  
But you know, that's the significance of having an enable web security annotation there, OK?

01:24:58.890 --> 01:25:10.670  
Damle, Nikhil  
There's another interesting annotation, and of course it's not directly related to security, but it's more of a spring boot, you know, annotation that it's called as conditional on property.

01:25:11.310 --> 01:25:28.320  
Damle, Nikhil  
So you know what this annotation will will instruct. A spring boot is that you know this particular class has to be has to be, you know instantiate iddb or maybe a spring bean has to be created for this class only if certain conditions are met.

01:25:29.650 --> 01:25:32.860  
Damle, Nikhil  
Right, so maybe this configuration classes not instantiate.

01:25:34.340 --> 01:25:36.200  
Damle, Nikhil  
Sorry it's not instantiated every time.

01:25:36.930 --> 01:25:44.600  
Damle, Nikhil  
It's instantiate only if you know we've provided a property in our application YAML file or in our properties file.

01:25:45.490 --> 01:25:53.720  
Damle, Nikhil  
The property name is really crappy dot dot mode, let's say and it has the value as basic auth. So if if our microservices or Springwood application.

01:25:55.050 --> 01:25:58.900  
Damle, Nikhil  
In the properties file, though, we've said that OK. Let's enable this for basic auth.

01:25:59.810 --> 01:26:12.440  
Damle, Nikhil  
Only then you know this conditional on property becomes true and at at server startup time you know the spring application server knows that or the spring container knows that you know this particular bean hash to be instantiated.

01:26:13.670 --> 01:26:24.680  
Damle, Nikhil  
So if you know we can figure this our application for some other authentication mechanism, let's say OK, where maybe we don't put in basic auth in our properties file, put something else.

01:26:26.320 --> 01:26:47.370  
Damle, Nikhil  
So it will, you know at server startup time will check you know if this property if this condition is not met it will not even instantiate this particular configuration. OK, so that's how you would really switch between different authentication or authorization security methods, right? By controlling that via properties file and even not even having.

01:26:48.200 --> 01:26:51.390  
Damle, Nikhil  
Uh, the configuration beans to be loaded.

01:26:52.920 --> 01:26:53.320  
Damle, Nikhil  
K.

01:26:54.940 --> 01:27:20.650  
Rani, Nisha 2  
Uh, nickel. I have a question here. Sorry to interrupt you so, uh, having value is equal to basic chords. So as you told like we can provide different type of I mean authorized different types of authorization. So if I don't provide a value, so is it there like default value will be there or what I mean will it throw error or it will not instantiate the entire class itself? How does it work?

01:26:56.050 --> 01:26:56.300  
Damle, Nikhil  
Yep.

01:26:58.080 --> 01:26:58.800  
Damle, Nikhil  
Yeah, sure.

01:27:20.090 --> 01:27:25.820  
Damle, Nikhil  
It it won't. It won't instantiate. You know it will just try to match the property value. I'll try to.

01:27:27.130 --> 01:27:28.880  
Damle, Nikhil  
I think it's an application the admin.

01:27:31.940 --> 01:27:38.010  
Damle, Nikhil  
And it's very, you know, something very so. OK, sorry it's coming from another. You know, property file called.

01:27:38.060 --> 01:27:38.260  
Damle, Nikhil  
So.

01:27:39.820 --> 01:27:43.980  
Damle, Nikhil  
See here basic auth. So this is a Jason that is, you know.

01:27:44.030 --> 01:27:45.240  
Damle, Nikhil  
Oh, uh.

01:27:45.950 --> 01:27:49.730  
Damle, Nikhil  
Actually applied during runtime for the spring boot application which has.

01:27:50.640 --> 01:27:57.630  
Damle, Nikhil  
A little bit information and a few other properties as well. As you know, the CAPI Orth mode. You know let me format disappeared.

01:27:59.870 --> 01:28:04.040  
Damle, Nikhil  
So this is where you know we have specified that the author mode is basic auth.

01:28:04.880 --> 01:28:05.290  
Rani, Nisha 2  
OK.

01:28:05.650 --> 01:28:18.150  
Damle, Nikhil  
And you know, if you maybe provide something like a like a open ID, right? So if you're if you have a class in your application, you know which is meant for Openid configuration, right? So?

01:28:19.750 --> 01:28:27.610  
Damle, Nikhil  
So in that case you know that class can be annotated with open ID here on only. In that case you know this class. That particular class will get instantiated.

01:28:29.140 --> 01:28:35.030  
Rani, Nisha 2  
OK, uh, just just one more thing. So can you go back to application yml file?

01:28:29.440 --> 01:28:30.150  
Damle, Nikhil  
But again.

01:28:30.940 --> 01:28:31.300  
Damle, Nikhil  
Uh-huh

01:28:32.240 --> 01:28:32.540  
Damle, Nikhil  
yeah.

01:28:35.790 --> 01:28:36.830  
Damle, Nikhil  
Yeah this one yeah.

01:28:36.070 --> 01:28:44.990  
Rani, Nisha 2  
So if yeah, so if I don't provide the mode here, then so basically we are not doing the security check and authorization, is it?

01:28:45.660 --> 01:28:53.330  
Damle, Nikhil  
Correct, we're not. We're not customizing it for our needs, really. You know, because this. So by default, whatever spring provides will come into play.

01:28:46.840 --> 01:28:47.200  
Rani, Nisha 2  
OK.

01:28:50.470 --> 01:28:50.900  
Rani, Nisha 2  
OK.

01:28:54.100 --> 01:29:02.660  
Damle, Nikhil  
But here you know, we want to customize that a bit, you know, to our custom needs. So that's why we're trying to customize this, but yeah.

01:28:59.610 --> 01:29:00.400  
Rani, Nisha 2  
He's got it. Thank you.

01:29:05.360 --> 01:29:06.510  
Rani, Nisha 2  
OK, thank you.

01:29:07.040 --> 01:29:08.870  
Damle, Nikhil  
Yeah, good question. But yeah, thank you.

01:29:10.240 --> 01:29:29.360  
Damle, Nikhil  
And the conditional on property. You know it's nothing very specific to security related, you know it's it's more of a spring lifecycle, you know, kind of a property where you can control whether a bean has to be instantiated or not. You know, based on some external properties that are set in your properties files or application YAML files.

01:29:33.900 --> 01:29:42.290  
Damle, Nikhil  
Uh, OK, but we've used used this, you know, in the context of security here where it can be used in any other context as well and.

01:29:44.830 --> 01:29:53.940  
Subramanian Kamatchi, Gobi Ganesh  
Nickel one thing here. I'm just wondering like how how do we get to know like so you you do? We do have annotations for this like do we do we have any?

01:29:46.490 --> 01:29:46.760  
Damle, Nikhil  
Yeah.

01:29:54.630 --> 01:30:02.060  
Subramanian Kamatchi, Gobi Ganesh  
Any, uh, I'll say a man page or help page, which which is the annotations and it's.

01:30:03.170 --> 01:30:04.680  
Subramanian Kamatchi, Gobi Ganesh  
Uh, purpose?

01:30:07.240 --> 01:30:20.340  
Damle, Nikhil  
Yeah, there's so many annotations looks so far from some spring spring, you know? Yeah, spring dot IO actually has a lot of documentation. Maybe I'll send you some of the links later, but.

01:30:10.520 --> 01:30:11.030  
Subramanian Kamatchi, Gobi Ganesh  
Yes.

01:30:19.980 --> 01:30:20.310  
Subramanian Kamatchi, Gobi Ganesh  
OK.

01:30:20.410 --> 01:30:29.120  
Damle, Nikhil  
Uh, and specially I think you know all these annotations were not as widely used with with traditional spring based applications right? But specially with.

01:30:29.870 --> 01:30:35.220  
Damle, Nikhil  
Spring boot, they've added so many of such validations that you know you can control.

01:30:36.260 --> 01:30:41.290  
Damle, Nikhil  
The life cycle of of these beans. Depending on you know external properties and all that.

01:30:43.140 --> 01:30:44.850  
Damle, Nikhil  
But but yeah, I'll send you some links.

01:30:46.240 --> 01:30:47.670  
Damle, Nikhil  
Maybe on the common chat later.

01:30:50.370 --> 01:30:50.630  
Damle, Nikhil  
K.

01:30:54.500 --> 01:31:22.150  
Damle, Nikhil  
So you know, I, I think it's it's going by spring boots principle that you know less code and you know more usage of the framework capabilities. You know, for example, you know if you had two really implement something like this, you know by hand you know. So we'll have to, you know, have a handle to the application context of spring container and then you know would love to then manually check for the value of this property. If it's not having this value, you know you don't. Instead you know.

01:31:22.260 --> 01:31:28.730  
Damle, Nikhil  
Try to block the the the black particular being and all that. So all that is really you know provided in like this.

01:31:29.490 --> 01:31:44.220  
Damle, Nikhil  
Kind of 1 annotation there, so I think there is so many other such annotations and click spring boots. Kind of philosophy is, you know code less, you know, so you have less boilerplate code and used maximum you know.

01:31:44.710 --> 01:31:52.400  
Damle, Nikhil  
All of the support provided by spring boot. Inherently. You know why annotations or other mechanisms.

01:31:54.090 --> 01:31:54.340  
Damle, Nikhil  
K.

01:31:58.590 --> 01:32:11.030  
Damle, Nikhil  
So coming back to the basic security configuration, right so we have a parent class here, but I know I will go. I will go to this class. Really this is the important class. It's called web security configurer adapter.

01:32:11.930 --> 01:32:14.900  
Damle, Nikhil  
And that's that really. Has you know the configure method?

01:32:16.210 --> 01:32:21.710  
Damle, Nikhil  
So we come back to basics, security configuration and you know we can customize, you know.

01:32:22.450 --> 01:32:31.120  
Damle, Nikhil  
Uh, the security layer bit here. OK, so we'll see configure. You know this is a method which we've overridden from the base class.

01:32:32.180 --> 01:32:38.730  
Damle, Nikhil  
The base class is the Web security configurer adapter provided by spring boot or spring security rather.

01:32:40.120 --> 01:32:44.620  
Damle, Nikhil  
Not if you look at here. You know we have a handle to this object called.

01:32:44.670 --> 01:32:47.020  
Damle, Nikhil  
So now it should be security.

01:32:47.730 --> 01:32:59.290  
Damle, Nikhil  
And you can do a few things you know using. It's actually using a builder pattern here if you realize, but you can just call you know or instruct this HTTP security object.

01:32:59.990 --> 01:33:04.390  
Damle, Nikhil  
Uh, to configure a few things, like for example you're seeing there in authorize all requests.

01:33:05.780 --> 01:33:15.650  
Damle, Nikhil  
And you know any requests that is coming in should be authenticated and it should use you know HTTP basic. So this is what you know enables the HTTP basic authentication.

01:33:16.630 --> 01:33:20.140  
Damle, Nikhil  
The dream is is OK. I mean, it's just customize dream here.

01:33:20.770 --> 01:33:21.530  
Damle, Nikhil  
And.

01:33:22.410 --> 01:33:26.540  
Damle, Nikhil  
In providing authentication entry point. So this is I will touch upon this a bit later.

01:33:27.560 --> 01:33:36.450  
Damle, Nikhil  
And see here in this session management you know you've said that it's stateless. So what it means is that you know every incoming request should really have a basic auth header.

01:33:39.080 --> 01:33:40.750  
Damle, Nikhil  
So this is the most important part, you know.

01:33:40.800 --> 01:33:44.380  
Damle, Nikhil  
Uh, and this is the only thing needed you know to kind of.

01:33:45.590 --> 01:33:47.420  
Damle, Nikhil  
Really enable basic authentication.

01:33:47.470 --> 01:33:50.960  
Damle, Nikhil  
No, and customize it a bit for your needs, right?

01:33:55.500 --> 01:34:00.930  
Damle, Nikhil  
So that's that's the configure part of it, you know, so this this will come into play when the server starts up.

01:34:03.720 --> 01:34:06.730  
Damle, Nikhil  
And you know, like we looked at, you know the.

01:34:07.590 --> 01:34:10.690  
Damle, Nikhil  
The slides in the story capability or see there was this.

01:34:11.660 --> 01:34:29.070  
Damle, Nikhil  
DL authentication provider. Right so it's provided by default. You know the authentication provider, but still if you want to customize it a bit, you know you can still do it, you know. So you create a new Dior indication provider. Now this if we look at the DA or 10 dictation provider.

01:34:29.900 --> 01:34:33.950  
Damle, Nikhil  
It's really a you know, a Spring Framework, spring security class.

01:34:37.360 --> 01:34:39.730  
Damle, Nikhil  
Right, so that is the day authentication provider.

01:34:41.350 --> 01:34:43.040  
Damle, Nikhil  
That we looked at here, right?

01:34:44.200 --> 01:34:52.520  
Damle, Nikhil  
So it has a bunch of methods, you know, kind of a life cycle of its own that you know it will authenticate. It will retrieve the user and you know it will.

01:34:53.610 --> 01:34:54.760  
Damle, Nikhil  
Authenticate

01:34:56.350 --> 01:34:57.980  
Damle, Nikhil  
the username and password.

01:35:01.280 --> 01:35:11.330  
Damle, Nikhil  
So that's that's already there. We customized it a bit, and that's why you know we added that as an add the rate being annotation so that at runtime this becomes are the authentication provider.

01:35:13.900 --> 01:35:14.250  
Damle, Nikhil  
No.

01:35:15.390 --> 01:35:18.050  
Damle, Nikhil  
There's one more, uh, you know, class that needs to be.

01:35:20.060 --> 01:35:32.390  
Damle, Nikhil  
Implemented rather and you know, again, going back to the slide, and this is the user details service here, right? So the user details service is something that has to be implemented. You know the user details.

01:35:33.920 --> 01:35:35.270  
Damle, Nikhil  
So this is an interface.

01:35:36.200 --> 01:35:54.130  
Damle, Nikhil  
And it it has just one method load user by username so the user name which is coming in on the basic auth header. You know that's passed here and then based on the username you know we load the information about the user. Typically the users username and password from the system.

01:35:56.040 --> 01:36:06.890  
Damle, Nikhil  
Right, so because it's an interface and you know we as application developers, know how to fetch that user information from where to fetch it rather than for example in this case.

01:36:07.930 --> 01:36:17.510  
Damle, Nikhil  
We fetched it from the database. You know, I think this is a repository. You know. If if you remember yesterday it's a repository which extends correct repository.

01:36:18.960 --> 01:36:23.060  
Damle, Nikhil  
And you know that so spring data JPA provided kind of a framework class.

01:36:24.970 --> 01:36:32.230  
Damle, Nikhil  
But let's not go into that right now. So you know, this is how we really load the username, and we create a, you know user details object.

01:36:34.060 --> 01:36:49.350  
Damle, Nikhil  
The user details object is returned back in as part of this method. So this load user by username is actually a method which gets called by spring security. You know during its chain of you know authentication flow.

01:36:50.560 --> 01:37:07.790  
Damle, Nikhil  
So in that you know one of the calls is to the user details service. Here the authentication provider will call in our case, in which the authentication provider right? So the authentication provider will call the user details service to load the user details and then it will try to match.

01:37:08.720 --> 01:37:10.430  
Damle, Nikhil  
If we look at the user details.

01:37:11.640 --> 01:37:18.910  
Damle, Nikhil  
This is again an interface and it has these two methods like get username and get password so the user details you know has to be.

01:37:20.420 --> 01:37:31.170  
Damle, Nikhil  
Again implemented by us and know if to provide implementation for these two methods. Specially get user name and user password. So these two will be really used by spring.

01:37:32.200 --> 01:37:32.890  
Damle, Nikhil  
Security.

01:37:33.720 --> 01:37:35.520  
Damle, Nikhil  
To do the validation really.

01:37:39.490 --> 01:37:47.630  
Damle, Nikhil  
So you know just to recap, you know we have to 1st. Of course customize. You know what has to be here. You can, actually, you know, because we've said any request.

01:37:48.670 --> 01:37:53.530  
Damle, Nikhil  
You can actually even give some patterns here that you know. Maybe you want to skip some of those.

01:37:53.580 --> 01:37:57.130  
Damle, Nikhil  
No, uh, URLs in some different context, but only.

01:37:58.240 --> 01:37:59.870  
Damle, Nikhil  
Have water integration. You know 4.

01:38:00.930 --> 01:38:09.270  
Damle, Nikhil  
See all the admin kind of things you know, so slash admin slash star will only block the ones which have that slash admin in their URL kind of thing.

01:38:09.950 --> 01:38:13.440  
Damle, Nikhil  
Right now you said that you know authenticate every request.

01:38:15.290 --> 01:38:15.560  
Damle, Nikhil  
K.

01:38:17.430 --> 01:38:23.380  
Damle, Nikhil  
So that's the configuration part of it, and then you know, like we said, the user details service right so?

01:38:25.370 --> 01:38:27.440  
Damle, Nikhil  
We have to implement that, you know.

01:38:31.720 --> 01:38:38.040  
Damle, Nikhil  
User details service to fetch the username from database or from our persistent store or from.

01:38:38.900 --> 01:38:42.600  
Damle, Nikhil  
You know anywhere which is relevant to our system.

01:38:45.600 --> 01:38:57.070  
Damle, Nikhil  
So if you you know maybe this is the only thing needed, you know. So once we let's say we had looked at this kind of a claims application yesterday, right? So on Swagger you know what it provides is?

01:38:58.290 --> 01:39:06.210  
Damle, Nikhil  
So, so this is the swagger interface you know, it's it's listing down all our services which are available. It also provides authorize button here.

01:39:07.080 --> 01:39:12.380  
Damle, Nikhil  
Case on authorized button you know you can actually put in your username and password that you want to use.

01:39:13.140 --> 01:39:15.320  
Damle, Nikhil  
If it's, you know, a basic security authentication.

01:39:16.870 --> 01:39:19.820  
Damle, Nikhil  
OK, and this username and password will be used.

01:39:21.830 --> 01:39:23.420  
Damle, Nikhil  
For you know.

01:39:24.530 --> 01:39:26.940  
Damle, Nikhil  
Calling any of the rest request from here.

01:39:28.260 --> 01:39:32.680  
Damle, Nikhil  
So it will pass. You know it will be passed as a authorization header along with the request.

01:39:34.950 --> 01:39:38.700  
Damle, Nikhil  
So let me you know, maybe just quickly, let's see at let's look at.

01:39:38.800 --> 01:39:43.300  
Damle, Nikhil  
So some of the hooks how they you know come into play at runtime so.

01:39:44.350 --> 01:39:49.280  
Damle, Nikhil  
From here you know I'm trying to get a claim by using the TR REF number.

01:39:50.950 --> 01:39:52.480  
Damle, Nikhil  
I'll try to execute this, you know.

01:39:56.310 --> 01:39:58.880  
Damle, Nikhil  
See here the curl command.

01:39:59.500 --> 01:40:01.580  
Damle, Nikhil  
You know it has I added the authorization header.

01:40:02.390 --> 01:40:05.970  
Damle, Nikhil  
Basic with our username, password. As I know base 64 encoded value.

01:40:09.030 --> 01:40:20.030  
Damle, Nikhil  
Of course it comes here, you know. Like I said, there is this hook, right? So it it has come here to our basic auth user details service to fetch, you know by username it will so the username is.

01:40:20.080 --> 01:40:23.870  
Damle, Nikhil  
Is this one which I had given up on Swagger?

01:40:25.860 --> 01:40:30.130  
Damle, Nikhil  
So it will try to find the user details from the database, right?

01:40:38.320 --> 01:40:39.810  
Damle, Nikhil  
So let us, it's actually.

01:40:42.130 --> 01:40:54.330  
Damle, Nikhil  
It's found the user details object, you know, and then it's trying to create that and return it back. OK, so that was the hook I wanted to really show that. You know whenever you call it, you know this load user by username will really be called every time.

01:40:57.540 --> 01:40:57.880  
Damle, Nikhil  
K.

01:41:01.330 --> 01:41:04.400  
Damle, Nikhil  
And then you know if I let it go ahead.

01:41:08.730 --> 01:41:15.640  
Damle, Nikhil  
It will do the basic authentication then and then it will call the actual service and then it will in return the response here.

01:41:18.830 --> 01:41:21.880  
Damle, Nikhil  
Served authentication was not successful. It would have returned back.

01:41:23.320 --> 01:41:24.650  
Damle, Nikhil  
A 404, you know?

01:41:26.760 --> 01:41:29.850  
Damle, Nikhil  
But 441 rather unauthorized.

01:41:36.250 --> 01:41:42.680  
Damle, Nikhil  
OK, any any questions or you know anything you want to see anything in more detail or.

01:41:45.110 --> 01:42:02.200  
Damle, Nikhil  
I think this is very easy to try out as well. You know. So maybe like tomorrow when we have our hands on session right here, we can easily try this out as well and you know, you can just mock mock this user details you know. So just put in some default values there for username and password and try to validate that.

01:42:12.490 --> 01:42:12.770  
Damle, Nikhil  
K.

01:42:23.580 --> 01:42:24.330  
Patro, Patanuru Santosh K  
So Nikhil.

01:42:23.680 --> 01:42:24.160  
Damle, Nikhil  
So.

01:42:25.050 --> 01:42:32.890  
Patro, Patanuru Santosh K  
So currently we're just discuss only the authentication part. So do you have any separate slide for the authorize and how you manage the operation in the spring?

01:42:25.230 --> 01:42:25.510  
Damle, Nikhil  
Yep.

01:42:34.490 --> 01:42:42.350  
Damle, Nikhil  
No, we don't have that example here, but you know, with the open ID Connect will talk a bit about authorization. We have a.

01:42:43.320 --> 01:42:46.550  
Damle, Nikhil  
We have an example about on JWT as well. Later in the day.

01:42:47.760 --> 01:42:51.380  
Patro, Patanuru Santosh K  
OK, maybe I will put that poster in that time, yeah?

01:42:49.970 --> 01:42:50.370  
Damle, Nikhil  
Filter.

01:42:51.640 --> 01:42:52.780  
Damle, Nikhil  
Sure, sure, yeah.

01:42:58.860 --> 01:43:00.820  
Damle, Nikhil  
OK, any other questions will.

01:43:10.710 --> 01:43:12.940  
Damle, Nikhil  
OK, there's another you know flavor.

01:43:14.560 --> 01:43:18.380  
Damle, Nikhil  
Two basic authentication. It's called digest based authentication, so you know.

01:43:19.550 --> 01:43:31.210  
Damle, Nikhil  
What happens is in basic or really the username and password has to be sent every time you know, although its base 64 encoded it's, it's rather plain text only right? Because can be easily decoded by anyone.

01:43:32.260 --> 01:43:34.550  
Damle, Nikhil  
Uh, sniffing the network so.

01:43:36.470 --> 01:43:39.150  
Damle, Nikhil  
This is slightly, you know, maybe a kind of a.

01:43:40.180 --> 01:43:50.630  
Damle, Nikhil  
A variation I would say you know if basic authentication and it's called digest based authentication. So spring security also has support for digest based authentication.

01:43:51.740 --> 01:43:52.200  
Damle, Nikhil  
And.

01:43:52.250 --> 01:43:52.520  
Damle, Nikhil  
So.

01:43:56.190 --> 01:43:58.190  
Damle, Nikhil  
What it means is you know, again, digest.

01:43:58.240 --> 01:44:03.610  
Damle, Nikhil  
UH is another. You know, authentication type for HTTP requests.

01:44:04.440 --> 01:44:13.430  
Damle, Nikhil  
Uh, but unlike basic oh, you know you don't need to pass the username and password in plain text, you know. Instead, you know what we do is.

01:44:14.750 --> 01:44:16.030  
Damle, Nikhil  
Like a quick.

01:44:16.980 --> 01:44:23.760  
Damle, Nikhil  
Colonel for you know algorithm is applied so from the consumer. Let's say you know there is a consumer.

01:44:24.270 --> 01:44:35.200  
Damle, Nikhil  
Oh was trying to call our service so the consumer would really create, you know something and this is a standard algorithm for MDM Defy Digest based authentication.

01:44:35.970 --> 01:44:42.520  
Damle, Nikhil  
And you know you have HTTP clients who which actually support this algorithm.

01:44:43.430 --> 01:44:44.110  
Damle, Nikhil  
Inherently.

01:44:45.270 --> 01:44:48.270  
Damle, Nikhil  
But if you look at what this algorithm does is, you know.

01:44:49.400 --> 01:44:55.910  
Damle, Nikhil  
It takes the username. The rim rim is in predefined pre agreed upon value and the password.

01:44:56.670 --> 01:45:03.960  
Damle, Nikhil  
And creates a one way hash. You know. So MD5 is actually a one way hash, and you know that is the hash one.

01:45:05.470 --> 01:45:14.420  
Damle, Nikhil  
It also takes some kind of a digest URA, again, which is a pre agreed upon value between the server and the consumer and the method which you know it's again.

01:45:15.260 --> 01:45:15.850  
Damle, Nikhil  
Oh

01:45:16.540 --> 01:45:17.290  
Damle, Nikhil  
algorithm.

01:45:18.110 --> 01:45:22.360  
Damle, Nikhil  
Uh name and do it again. Creates a MD 5 hash of that.

01:45:24.280 --> 01:45:33.070  
Damle, Nikhil  
And then you again doesn't one move. I'm device of the HK one and the HK 2 using a \*\*\*\*\* value. So the \*\*\*\*\* value is, you know something which.

01:45:33.970 --> 01:45:51.940  
Damle, Nikhil  
Ah, the consumer actually gets from the server during an initial handshake with the server, so you know of course we'll look at that bit on the next slide, but then on the on sval use something which mostly is is very short lived and changes very frequently, and that's how.

01:45:53.030 --> 01:45:55.620  
Damle, Nikhil  
That one is really used in and to create a.

01:45:56.500 --> 01:46:01.310  
Damle, Nikhil  
Complete hash and this is it's a one way hash, you know so it can cannot be.

01:46:01.360 --> 01:46:01.560  
Damle, Nikhil  
Well.

01:46:02.640 --> 01:46:05.100  
Damle, Nikhil  
Decoded really and this response.

01:46:06.020 --> 01:46:20.610  
Damle, Nikhil  
Is something which you know is sent on the authorization header with each request, although it's called response, you know it's actually sent on the HTTP request header. The authorization header with each request.

01:46:21.620 --> 01:46:27.150  
Damle, Nikhil  
Any far microservices application is only is configured four digest based authentication.

01:46:28.990 --> 01:46:48.530  
Damle, Nikhil  
On our side, you know the the username of courses available in plain text to the user name is used to fetch the username and password from the database and the server also creates a similar kind of hash on the server side and then the two hash values are matched, you know?

01:46:48.990 --> 01:46:55.490  
Damle, Nikhil  
So if the if the two hash values really match, then you know the authentication is successful. That kind of thing.

01:46:58.280 --> 01:47:10.940  
Damle, Nikhil  
So in this case you know what happens is on on each request. You know you don't really send the password in plain text, but it's actually a hashed value, and you know that's kind of slightly more secure than sending a plaintext.

01:47:11.750 --> 01:47:18.410  
Damle, Nikhil  
Username, password it with every request right? Especially you know another thing is from a UI perspective as well. You know once.

01:47:19.400 --> 01:47:49.100  
Damle, Nikhil  
So the user logs in. You know instead of storing the username password really on the UI for each of the rest calls, you could really also store the hash values in your Reno browsers, local stories or session stories. So you're not storing the users password in your browser session storage, but rather use storing just the MD 5 hash value there you know, so that's also more secure from a UI standpoint, right? When the UI is calling all these rest calls.

01:47:49.720 --> 01:47:52.350  
Damle, Nikhil  
And they can send in the you know MD5 header.

01:47:53.670 --> 01:47:54.350  
Damle, Nikhil  
Ah.

01:47:54.980 --> 01:47:56.940  
Damle, Nikhil  
Instead of sending the username password.

01:48:00.140 --> 01:48:01.470  
Damle, Nikhil  
Can you know like we said?

01:48:02.780 --> 01:48:07.070  
Damle, Nikhil  
Each of the request you know will contain this digest header and the spring security layer.

01:48:07.770 --> 01:48:10.150  
Damle, Nikhil  
Here will intercept that header.

01:48:12.380 --> 01:48:21.290  
Damle, Nikhil  
And then try to match. Try to construct a similar header similar MD 5 hash value and try to match the two hash values, right?

01:48:23.140 --> 01:48:26.980  
Damle, Nikhil  
This is just a you know, a flow really of.

01:48:28.430 --> 01:48:35.190  
Damle, Nikhil  
Will tightening you know? So like I said, you know there's this initial handshake right? So this is the client and this is a server.

01:48:36.570 --> 01:48:49.940  
Damle, Nikhil  
We try to, you know, call a service which is protected by digest authentication spring security layer. You know the first response is a 401 unauthorized, but along with the 401 authorized.

01:48:50.790 --> 01:48:53.260  
Damle, Nikhil  
It also indicates that you know.

01:48:54.120 --> 01:48:57.980  
Damle, Nikhil  
You need to send a digest you know along with the request header.

01:48:59.210 --> 01:49:05.080  
Damle, Nikhil  
It sends all the information which is relevant, like the \*\*\*\*\* information. So the \*\*\*\*\* is used in order to create the hash.

01:49:07.050 --> 01:49:13.900  
Damle, Nikhil  
Uh, once it gets a 401 on authorize, you know with WW DOT train ticket digest, you know. So the HTTP client.

01:49:14.580 --> 01:49:22.630  
Damle, Nikhil  
Uh, knows that you know this is actually digest based authentication mechanism implemented, so it needs to create.

01:49:24.210 --> 01:49:26.840  
Damle, Nikhil  
Or MD5 digest. Or you know header.

01:49:27.690 --> 01:49:38.380  
Damle, Nikhil  
So it uses this non send all this information that is passed back on that 401 response uses all that you know to end of course uses the username and password to create.

01:49:39.460 --> 01:49:44.610  
Damle, Nikhil  
So you know that MD 5 hash. If you look at here in this uh response equal to.

01:49:46.040 --> 01:50:01.150  
Damle, Nikhil  
So all this is part of the authorization header in all this big string is part of the authorization header. You know the type is digest the username, the ream the \*\*\*\*\* is passed back again. The URL and the response in the response is really that hash value.

01:50:02.850 --> 01:50:06.600  
Damle, Nikhil  
But now you know know where here the password is. You know in plain text.

01:50:08.930 --> 01:50:12.430  
Damle, Nikhil  
Once this request is made in with a valid digest header.

01:50:13.200 --> 01:50:20.290  
Damle, Nikhil  
The spring security tries to match that, and you know then it can either allow it or not allow it depending on if the.

01:50:21.180 --> 01:50:22.960  
Damle, Nikhil  
Turn dictation was successful or not.

01:50:26.620 --> 01:50:26.890  
Damle, Nikhil  
OK.

01:50:31.970 --> 01:50:52.780  
Yadav, Rameshchandra  
Nikhil, this is a one time. Like if user logged in the client side this is happening for her one time or or each time this flow will run in background for one request or how is it worked. Suppose I have raising one request from a client to server so that flow which you have explained it's one pane or based on different request.

01:50:32.100 --> 01:50:33.080  
Damle, Nikhil  
So Yep.

01:50:52.750 --> 01:50:53.080  
Damle, Nikhil  
Right?

01:50:53.370 --> 01:51:07.460  
Damle, Nikhil  
Good good question. You know, because it's it's a very heavy operation. If they you know every time you get a 401 and then you figure out now. So I think what happens normally you know let's say there is a let's say the UI. You know it's trying to call.

01:51:07.770 --> 01:51:10.160  
Damle, Nikhil  
Uh, so microservices applications?

01:51:11.190 --> 01:51:23.620  
Damle, Nikhil  
So once it gets that 401 and it's it gets handled that \*\*\*\*\* value, it creates the MD 5 hash value and stores in the browsers you know in the local session storage. So with the subsequent request.

01:51:24.240 --> 01:51:26.680  
Damle, Nikhil  
You know it by default, sends the digest header.

01:51:27.770 --> 01:51:31.450  
Damle, Nikhil  
So shall violate digest header. Spring security will let you in.

01:51:33.340 --> 01:51:37.380  
Damle, Nikhil  
But each request should have the digest header, you know, because it's like a stateless request.

01:51:38.560 --> 01:51:47.800  
Damle, Nikhil  
The \*\*\*\*\* value you know expires once in a while, depending on what you've configured on on spring security. Let's say the \*\*\*\*\* value expires every 10 minutes.

01:51:49.220 --> 01:52:00.050  
Damle, Nikhil  
So if the \*\*\*\*\* value was expired and you know we've created a digest using that \*\*\*\*\* of course the digest won't match with what the server comes up with because the server on the server, the \*\*\*\*\* value is changed.

01:52:00.780 --> 01:52:04.250  
Damle, Nikhil  
So then in that case, again, it will send a 401 with the new \*\*\*\*\* value.

01:52:05.560 --> 01:52:15.290  
Damle, Nikhil  
And then you know the the UI can create that new hash and store that new hash you know temporarily on its side and use the new MD 5 hash value for all the subsequent.

01:52:16.390 --> 01:52:17.210  
Damle, Nikhil  
Rest calls.

01:52:20.230 --> 01:52:24.340  
Yadav, Rameshchandra  
So it's depend upon the configuration part right? And the expiry part.

01:52:24.700 --> 01:52:33.270  
Damle, Nikhil  
Yeah, the \*\*\*\*\* value can be configured to expired like every few minutes. Or you know maybe everyone or or that kind of so there is a.

01:52:32.870 --> 01:52:33.360  
Yadav, Rameshchandra  
And this.

01:52:33.960 --> 01:52:34.230  
Damle, Nikhil  
Yep.

01:52:34.730 --> 01:52:39.290  
Yadav, Rameshchandra  
But these things are happening in in the code label spring.

01:52:40.500 --> 01:52:47.890  
Damle, Nikhil  
Spring security layer you know, so again, you don't have to code anything. Spring security will take care of all this magic really.

01:52:42.530 --> 01:52:42.900  
Yadav, Rameshchandra  
OK.

01:52:48.840 --> 01:52:49.170  
Yadav, Rameshchandra  
OK.

01:52:49.630 --> 01:52:58.550  
Damle, Nikhil  
So it will create all the hash you know we will try to compare the incoming digest header with the hash value that it has. You know computed if the hash values matches then.

01:53:00.450 --> 01:53:02.770  
Damle, Nikhil  
It will, you know, allow the request to come in.

01:53:05.910 --> 01:53:12.110  
Damle, Nikhil  
It's all configuration like this again, there is no code needed because the spring security inherently supports this mechanism.

01:53:16.130 --> 01:53:34.690  
Thirugnanam, Kumaran  
Uh, Nikola, one more adult say this \*\*\*\*\* value will love our timed duration. Say so. User One asked for this. Then the non values return. Then when the user to ask will the same \*\*\*\*\* value returned within the same time period since the server doesn't know user one and two?

01:53:23.180 --> 01:53:23.540  
Damle, Nikhil  
Yeah.

01:53:24.130 --> 01:53:24.470  
Damle, Nikhil  
Yeah.

01:53:32.830 --> 01:53:33.280  
Damle, Nikhil  
Yeah.

01:53:35.550 --> 01:53:37.470  
Thirugnanam, Kumaran  
Uh, it is that the case.

01:53:36.770 --> 01:53:37.050  
Damle, Nikhil  
Yeah.

01:53:38.200 --> 01:53:39.600  
Damle, Nikhil  
True true, yeah, that's true.

01:53:39.070 --> 01:53:47.930  
Thirugnanam, Kumaran  
Yeah, so after that say user name one is guest one and the response is sent, and I mean the subsequent.

01:53:47.980 --> 01:54:06.150  
Thirugnanam, Kumaran  
Uh, authorization request and User 2 then response? Then how does the response is decrypted or deciphered based on the combination? Like will there be username, password, combination somewhere to judge the response code to decrypt it?

01:54:06.370 --> 01:54:16.360  
Damle, Nikhil  
Yeah, so here you know when when the MD5H1 is creating, so this is the algorithm really to create that hash so the HK one comprises of the username and the password.

01:54:17.870 --> 01:54:20.480  
Damle, Nikhil  
So the consumer will create this.

01:54:21.920 --> 01:54:27.240  
Damle, Nikhil  
MD5 hash and with H1 N H2. You know it will combine that to create that digest header.

01:54:28.730 --> 01:54:33.870  
Damle, Nikhil  
And that will be sent on the HTTP request. So the password is actually part of that hash.

01:54:35.000 --> 01:54:36.700  
Damle, Nikhil  
Value SG-1.

01:54:37.540 --> 01:54:48.500  
Damle, Nikhil  
And on the server side, you know based on the username we will fetch the password from the database, let's say and spring security again will apply the same algorithm on its side.

01:54:49.190 --> 01:55:00.330  
Damle, Nikhil  
To create a chat 1H2 and the digest header at the hash value and then it will try to match the incoming hash value on the HTTP header with the hash value that it has computed.

01:55:03.200 --> 01:55:20.270  
Thirugnanam, Kumaran  
Yeah, so, uh whatever. Yeah, I understood that. What I'm trying to figure out is, will there be a store passwords stored somewhere? I know username as you said it will be looked up from a database maybe, but will the password be stored somewhere like basic earth?

01:55:19.580 --> 01:55:20.160  
Damle, Nikhil  
Password.

01:55:21.240 --> 01:55:24.370  
Damle, Nikhil  
Yeah, password is on the database, let's say like basic auth.

01:55:23.590 --> 01:55:24.080  
Thirugnanam, Kumaran  
OK.

01:55:24.850 --> 01:55:25.180  
Thirugnanam, Kumaran  
OK.

01:55:25.070 --> 01:55:34.320  
Damle, Nikhil  
So you know, based on the username, you will fetch the username and if ya'll validated the username is valid in in your database and for that username you will fetch the password.

01:55:35.880 --> 01:55:40.590  
Thirugnanam, Kumaran  
OK, so it's kind of extension to basic control. Yeah, so password is still there in system.

01:55:38.930 --> 01:55:39.390  
Damle, Nikhil  
Correct?

01:55:41.140 --> 01:55:42.440  
Damle, Nikhil  
Correct, correct, correct?

01:55:43.830 --> 01:55:52.810  
Damle, Nikhil  
Correct, correct? It's you know it's it's a EXT or you know flavor of basic auth. But instead of sending the password in plain text on the authorization header.

01:55:53.530 --> 01:56:00.210  
Damle, Nikhil  
Uh, and you know, even having to save or in a store that password on your, let's say if it's a UI application.

01:56:01.000 --> 01:56:11.740  
Damle, Nikhil  
Sort of having to save that users password in the UI browser, session stories, or somewhere you know? Yeah, that's kind of a security risk, right? So instead of storing that password in plain text?

01:56:12.650 --> 01:56:15.090  
Damle, Nikhil  
You can actually store that achieve and value you know.

01:56:16.070 --> 01:56:22.760  
Damle, Nikhil  
Or in your browser, zoom session storage somewhere so that with each subsequent request.

01:56:23.750 --> 01:56:30.580  
Damle, Nikhil  
You can use that H1 along with H2O2 and with the \*\*\*\*\* with the current \*\*\*\*\* value to create the digest header.

01:56:32.420 --> 01:56:35.170  
Damle, Nikhil  
And put that digest header on the HTTP request.

01:56:38.360 --> 01:57:05.440  
Yadav, Rameshchandra  
Nikhil same thing, same concept. We can apply in the base basic authentication also right? So what's the difference there? See in the what is the logic here is basically what we are doing. We are reading the username from request and passwords from a DB. Then we are converting to hash with this particular algorithm and then sending matching logic. Then say why not we can same concepts. We are not applied for a basic authentication.

01:57:07.240 --> 01:57:12.950  
Damle, Nikhil  
Yeah, it can be, you know. But then the basic authentication follows a particular you know.

01:57:13.850 --> 01:57:15.190  
Damle, Nikhil  
Pretty cool, right? I mean.

01:57:16.580 --> 01:57:23.930  
Damle, Nikhil  
Because it it's actually the way you know that author is constructed with the username, password and the base 64 encoded value.

01:57:24.630 --> 01:57:38.010  
Damle, Nikhil  
And then you know the server side also understands that if it's a basic auth, you know this is how I'll have to read the author and tried to validate it. So it's mostly about unit defining the protocol. So basic auth protocol is defined in that way.

01:57:38.780 --> 01:57:45.720  
Damle, Nikhil  
But really, you know if you want a extension to that could use basic auth along with digest based authentication.

01:57:47.530 --> 01:57:48.670  
Damle, Nikhil  
Because spring security.

01:57:47.620 --> 01:57:49.830  
Yadav, Rameshchandra  
Customization we can do that, right?

01:57:49.970 --> 01:57:50.870  
Damle, Nikhil  
Yes yes yes.

01:57:52.530 --> 01:58:00.250  
Damle, Nikhil  
Yeah, you could have done that same, but you know again, spring security provides a few things out of the box now. For example, the \*\*\*\*\* value which tries to you know?

01:58:00.920 --> 01:58:26.610  
Damle, Nikhil  
Alex patterns value every once in awhile and then the MD 5 hash. You know there is a predefined algorithms for MD5 hash, so the client side you know also should comply to the same algorithms because if they use some oil, different algorithms and create the hash in a different value, the hashes won't match in the hash. Values won't match any time, so it's basically defining a protocol. You know doing these things in in a particular way so that.

01:58:27.360 --> 01:58:31.190  
Damle, Nikhil  
The server or the consumer and the server both are, you know.

01:58:31.890 --> 01:58:33.610  
Damle, Nikhil  
Trying to use the same mechanisms.

01:58:39.450 --> 01:58:39.780  
Damle, Nikhil  
K.

01:58:48.620 --> 01:58:50.520  
Damle, Nikhil  
Yeah, good questions though. Yeah.

01:58:49.140 --> 01:58:56.210  
Patro, Patanuru Santosh K  
Awesome so Nikhil so during the handshake we will only gather Elum from the server to the client.

01:58:56.960 --> 01:59:01.400  
Damle, Nikhil  
Yeah, the dream, the domain, the \*\*\*\*\* value, so the \*\*\*\*\* value is also very important here.

01:59:02.280 --> 01:59:13.410  
Patro, Patanuru Santosh K  
Welcome to Knoxville will be will be sent by the server when the client is requesting right but the Prada initial handshake be happening only based upon the ramble right?

01:59:14.090 --> 01:59:15.050  
Patro, Patanuru Santosh K  
Real and that you love.

01:59:14.400 --> 01:59:14.770  
Damle, Nikhil  
Right?

01:59:15.980 --> 01:59:22.260  
Damle, Nikhil  
Correct Susie initial handshake. You know if you see here I send the request. You know, without any authorization header.

01:59:23.090 --> 01:59:55.220  
Damle, Nikhil  
So spring security realizes that you know there is no authorization header, but this, you know application is configured for digest based authentication, so it will send back a 401 response. You know 401 is unauthorized along with the 401 response. It will also send all these values. The digest, the domain, the \*\*\*\*\*. So once the consumer receives the 401 response with this kind of a header West Authenticate Digest the HTTP client or you know the client side.

01:59:37.560 --> 01:59:37.900  
Patro, Patanuru Santosh K  
OK.

01:59:55.830 --> 02:00:00.840  
Damle, Nikhil  
Uh, is it smart enough to figure out that? OK, we need to, you know, create a digest header.

02:00:02.120 --> 02:00:07.430  
Damle, Nikhil  
And use that digest header on the next request. You know so that the request is a valid request.

02:00:05.510 --> 02:00:05.890  
Patro, Patanuru Santosh K  
OK.

02:00:10.760 --> 02:00:21.300  
Patro, Patanuru Santosh K  
So the handshake is happening in memory then and there is not like OK before the client start communicating to the server. There should be any kind of. Hence second it's not like that OK?

02:00:19.470 --> 02:00:19.970  
Damle, Nikhil  
No.

02:00:21.360 --> 02:00:23.650  
Damle, Nikhil  
Right no no. It's purple request in order.

02:00:23.540 --> 02:00:24.370  
Patro, Patanuru Santosh K  
But request right?

02:00:24.350 --> 02:00:25.160  
Damle, Nikhil  
Like this, yeah?

02:00:26.240 --> 02:00:27.050  
Patro, Patanuru Santosh K  
The game.

02:00:26.360 --> 02:00:34.420  
Damle, Nikhil  
Plus you know, every time you don't need to wait for a 401, you know, because once you've got that 401, you've got handled the \*\*\*\*\* value.

02:00:35.020 --> 02:00:35.330  
Patro, Patanuru Santosh K  
OK.

02:00:35.280 --> 02:00:46.600  
Damle, Nikhil  
And once you've created the digested or you can store the digest header on your consumer side for for some time. You know until the \*\*\*\*\* expires. Until that happens, you know the digester is still valid.

02:00:47.770 --> 02:00:49.390  
Damle, Nikhil  
But all subsequent requests.

02:00:50.580 --> 02:00:55.330  
Patro, Patanuru Santosh K  
Again, that \*\*\*\*\*\* posed a VM, right? So technically I think we OK.

02:00:53.870 --> 02:00:54.250  
Damle, Nikhil  
Got it.

02:00:55.810 --> 02:00:56.320  
Damle, Nikhil  
Correct?

02:00:57.340 --> 02:01:07.350  
Damle, Nikhil  
But that can you know that that can be synchronized really? So there is a way to say we can cash those values and have a common cache and all that.

02:01:07.980 --> 02:01:08.310  
Patro, Patanuru Santosh K  
OK.

02:01:10.570 --> 02:01:11.070  
Patro, Patanuru Santosh K  
Thank you.

02:01:11.920 --> 02:01:12.330  
Damle, Nikhil  
K.

02:01:19.490 --> 02:01:21.920  
Damle, Nikhil  
OK, good questions, I think I'll.

02:01:23.280 --> 02:01:30.330  
Damle, Nikhil  
Then we can move to the next topic. Really, you know that's like what two? You know. Some of the concepts of over 2 and.

02:01:31.020 --> 02:01:37.970  
Damle, Nikhil  
Open ID, you know, connect and then I think so I'll I'll let Amber take over from here.

02:01:38.350 --> 02:01:49.060  
Patil, Tatyasaheb  
When you kill it, kill you that there is one question in the chat, like from like Ramanathan, like what happens when this service want to hit another service.

02:01:38.630 --> 02:01:39.900  
Damle, Nikhil  
Uh, yeah.

02:01:43.940 --> 02:01:44.290  
Damle, Nikhil  
OK.

02:01:49.850 --> 02:01:51.450  
Patil, Tatyasaheb  
Maybe in case of digest?

02:01:55.220 --> 02:01:56.280  
Damle, Nikhil  
Proper nouns or wash?

02:02:00.280 --> 02:02:12.300  
Damle, Nikhil  
Right, I mean the same digest header can be used clearly, you know. So the same digest header can be used to call because in that case you know what happens is. Sorry, actually I'm to some other slides.

02:02:15.030 --> 02:02:22.440  
Damle, Nikhil  
So in that case you know this client really is is your micro service application, which is calling trying to call another microservices application right so?

02:02:23.620 --> 02:02:25.890  
Damle, Nikhil  
I think the same protocol applies, you know here.

02:02:26.650 --> 02:02:50.930  
Damle, Nikhil  
In that case as well, so that the micro service application will actually try to call another microservice application, you know. And if it's configured for digester indication, it'll get a 401. You know that's kind of the handshake, which is, you know, transparent to the application developer. Really, you know the application developer need not worry about handling the 401 taking the \*\*\*\*\* and although it's part of that, you know protocol so.

02:02:51.690 --> 02:03:04.120  
Damle, Nikhil  
Uh, the clients and the servers which support digest authentication will take care of that. You know, under the hoods will will take care of this handshake and creating the header and all that will be done by the.

02:03:04.780 --> 02:03:08.650  
Damle, Nikhil  
It should be client, or in this case the spring security layer.

02:03:12.520 --> 02:03:12.800  
Damle, Nikhil  
K.

02:03:20.710 --> 02:03:24.070  
Damle, Nikhil  
Is that is that OK? Yeah yeah. OK sure sure.

02:03:22.820 --> 02:03:23.080  
Patil, Tatyasaheb  
Yeah.

02:03:25.830 --> 02:03:27.060  
Patil, Tatyasaheb  
That answers the question.

02:03:27.160 --> 02:03:27.500  
Damle, Nikhil  
Yeah.

02:03:28.700 --> 02:03:30.080  
Patil, Tatyasaheb  
He's like OK, thanks.

02:03:33.840 --> 02:03:40.260  
Durga, Amar  
OK Nikhil, then I think yeah yeah thank you. Actually that is a detailed explanation.

02:03:34.660 --> 02:03:36.190  
Damle, Nikhil  
OK, AMER yeah will do you.

02:03:37.570 --> 02:03:38.020  
Damle, Nikhil  
Thank you.

02:03:40.560 --> 02:03:41.040  
Durga, Amar  
Uh.

02:03:42.220 --> 02:03:43.610  
Durga, Amar  
Yeah, moving to the next topic.

02:03:43.660 --> 02:03:46.340  
Durga, Amar  
Gonna let me share my screen.

02:03:54.430 --> 02:03:56.180  
Durga, Amar  
I hope you can see my screen right.

02:03:59.080 --> 02:03:59.490  
Ayyanan, Ramya  
Yeah, yeah.

02:03:59.250 --> 02:03:59.570  
Damle, Nikhil  
Yeah.

02:03:59.620 --> 02:04:00.070  
Patil, Tatyasaheb  
Yes, I'm.

02:04:00.520 --> 02:04:00.780  
Durga, Amar  
OK.

02:04:02.120 --> 02:04:11.130  
Durga, Amar  
So yeah, it's Nikhil. Explain about basic auth and digest or so. Moving to the next one is on the signal line. Things as being tracked.

02:04:11.180 --> 02:04:13.510  
Durga, Amar  
Pretty, uh, there is a boat.

02:04:14.390 --> 02:04:18.970  
Durga, Amar  
So anyone in general, can you explain about if you know anything about what?

02:04:20.050 --> 02:04:21.750  
Durga, Amar  
Can you tell me what is what?

02:04:31.800 --> 02:04:35.300  
Yadav, Rameshchandra  
Water is mentioning is token based authentication.

02:04:36.640 --> 02:04:38.230  
Durga, Amar  
OK, so.

02:04:40.830 --> 02:04:46.120  
Durga, Amar  
Only on token base authentication means what is the definition actually? Oh means what?

02:04:48.840 --> 02:04:49.520  
Yadav, Rameshchandra  
Open.

02:04:50.510 --> 02:04:51.170  
Yadav, Rameshchandra  
The dictation.

02:04:50.960 --> 02:04:51.410  
Durga, Amar  
OK.

02:04:52.220 --> 02:04:55.220  
Durga, Amar  
Optimists organization or event occasion.

02:04:59.030 --> 02:04:59.550  
Durga, Amar  
Perfect.

02:05:00.940 --> 02:05:04.090  
Durga, Amar  
Yeah, So what? Actually it is a protocol.

02:05:04.920 --> 02:05:12.460  
Durga, Amar  
Again, it's also a protocol and old means open, and the author means authorization, not authentication.

02:05:13.330 --> 02:05:29.080  
Durga, Amar  
There are two flavors actually. There is a first version and then later on they came up with the next version in industry. Most of the times we use 2.0, so whenever we discuss about what it is by default we are talking about both at 2.0.

02:05:30.130 --> 02:05:50.200  
Durga, Amar  
OK, and uh, what actually it'll do? So it is alos accessing your resource. OK, offer resource owner by enabling the client application on HTTPS services such as third party providers actually provides those things like Facebook, Gmail or GitHub etc.

02:05:51.850 --> 02:06:14.200  
Durga, Amar  
So what it also tells you it allows the sharing sharing of resources stored in one side and with another site. We thought sharing their credentials because in basic auth what we do is we do username and password right? So instead of giving all this credential to all the third party sites, what we decided is we rely on some Pacific.

02:06:14.860 --> 02:06:18.700  
Durga, Amar  
Uh, vendors where they are giving on this authorization.

02:06:20.510 --> 02:06:40.960  
Durga, Amar  
Are they insane credential that we already registered with them so that the third party providers they already have our credentials? So instead of sharing to all other vendors, we use those functionality and takes that indirectly. We take the token and based on the token we go and do the functionality.

02:06:41.890 --> 02:07:13.100  
Durga, Amar  
So here what will happen is if you see any shopping sites so they always see that. OK, uh, if you want to log in, or if you want to purchase me order so they say that OK you need to log in with our credentials. You need to keep on registering the things right so there what will happen is they also broke up with the third party provider like Facebook and Gmail, so that's where you see that. Log in with Facebook, login with Gmail, things will be there. So if you do with that other indication so then you don't provide again.

02:07:13.380 --> 02:07:33.570  
Durga, Amar  
Use your personal details so you're not sharing your personal details to the third party sites. So instead of that you're using one of the Gmail's provider and on your behalf, so Gmail will tell their OK, I have this person credentials so you can organize it. You can, uh, then, ticket the user and proceed for the order.

02:07:34.690 --> 02:07:52.790  
Durga, Amar  
So that's where the word comes in picture, so we'll discuss about much in terms of what are the rules are there, and what are the flows. Are there in war the 2.0? So before that I will also explain about what are the advantage which we get from both.

02:07:53.650 --> 02:08:01.230  
Durga, Amar  
So in this slide I will see, uh, there are different advantages with workflow. Why we should go with oh, author?

02:08:02.540 --> 02:08:28.870  
Durga, Amar  
One is the first advantage with this client, since overflows or simplified. So instead of in basic auth, you see once we get a capture the user name and password. Then you validate that person right? So you are calling the user service there and your other indicating was awful. You're storing the value on top of that, your other indicating that value. So here the client server flows are simplified.

02:08:29.740 --> 02:08:48.670  
Durga, Amar  
And also the next one is whenever you are working in a social media apps. There are so many thousands of apps nowadays. Works right? So yeah, whenever you're working on these apps whenever you're registering for those things, so every time every application instead of registering it you.

02:08:49.320 --> 02:09:03.670  
Durga, Amar  
Ah, you always rely on the third party so that once you register those vendors most popular one then automatically in other sites, no need to register completely, so the integration becomes simplicity because of the social media app space.

02:09:04.510 --> 02:09:14.310  
Durga, Amar  
And here we are also using third party logins. So that is what nothing but you were providing like a Gmail or Facebook. So those things will be used in this place.

02:09:15.110 --> 02:09:43.850  
Durga, Amar  
And the complex is the code complexity is more into the server and the client side which we are implementing, which will be easy for us. So we should not worry about what is the complexity on the server side. So the client side code will be simpler enough and these tokens also actually can be. It can be maintained the timely. So if you want the token can be 30 minutes if you want the token for the 30 day we can do that. So there were token can be.

02:09:45.290 --> 02:09:59.200  
Durga, Amar  
Depend upon your configuration, the tokens are validated. Validated can be increased, so that is also. The flexibility is also provided, so that's the the last one says that access and Long live tokens are possible.

02:10:00.640 --> 02:10:06.540  
Durga, Amar  
So these are the five different advantages which we see, but again.

02:10:07.480 --> 02:10:25.110  
Durga, Amar  
Overall, the concept you understand, right? So what is basically how this is other authorization? Basically it is used for authorization and these are the things based on considering all these factors. Now where this all the services are are using with this approach as well.

02:10:26.100 --> 02:10:27.870  
Durga, Amar  
Claims are preferring with this approach.

02:10:29.420 --> 02:10:52.770  
Durga, Amar  
And the next one we talk about the rules. What are the different roles are there in both? So there are four simple rules OK and and also I'm requesting if you have any questions please stop me and ask me if you need any clarification because these roles in the flows are really important in order to understand overflow.

02:10:53.630 --> 02:10:53.950  
Durga, Amar  
OK.

02:10:55.110 --> 02:11:02.290  
Durga, Amar  
But, uh, if you talked about the roles, there are four simple 4 rolls are there. First one is a client.

02:11:03.070 --> 02:11:04.020  
Durga, Amar  
Who is the client?

02:11:05.440 --> 02:11:08.510  
Durga, Amar  
OK client, here is a third party application.

02:11:10.020 --> 02:11:15.370  
Durga, Amar  
OK, the third party application where is attempting to get some users account data.

02:11:16.470 --> 02:11:22.370  
Durga, Amar  
And before you do any action, you need to get permission, right? That is a line.

02:11:24.600 --> 02:11:41.370  
Durga, Amar  
OK, later on we'll talk about some examples also, so I'll give a wizard client. Also in that case, but in terms of definition, line teaser like third party application where we need to have permission to access the power to perform the action.

02:11:42.750 --> 02:11:53.420  
Durga, Amar  
OK, and the resource owner who is the resource 10. This is nothing but a API server is also called as a PS over and where you will get the actual information.

02:11:55.490 --> 02:11:55.940  
Durga, Amar  
OK.

02:11:58.210 --> 02:12:20.800  
Durga, Amar  
And the next release authorization server. So this authorization server what it'll do is. It'll based on the interface whatever the request it came. It will waiver, approve or denies the request. OK, this will take care in terms of whether I should approve it or I should. I should provide the token or I should deny the token.

02:12:22.200 --> 02:12:24.610  
Durga, Amar  
OK, and the resource owner.

02:12:25.540 --> 02:12:32.180  
Durga, Amar  
Actually this is a user action. The resource owner is actually a physical person who is giving access to the.

02:12:32.930 --> 02:12:48.850  
Durga, Amar  
Some limited portion of their account to the client client where you want to perform some act, and right so these the owner resource owner is the user where he gives some portion of the access to the client so that client can perform the task.

02:12:51.260 --> 02:13:07.070  
Durga, Amar  
So the control is also in the control, will be there with the resource owner, so whether it to give the access or not. So based on that authorization server internally will decide OK based on resource owner, organization server will approve or reject.

02:13:11.750 --> 02:13:33.550  
Durga, Amar  
And there are so many flows. Are there, uh, and we will discuss at high level, uh flows, but the standard flow is this is the this is the flow we can capture at this point of time. And but again depends upon your project and we depend upon your scenario. These flows can vary.

02:13:34.450 --> 02:13:44.170  
Durga, Amar  
But the standard of most of the industry followed this is the standard flow. As I said, there are four roles. One is a client OK.

02:13:44.980 --> 02:13:56.040  
Durga, Amar  
Client is nothing, client is 1 where is sending the request for support and the resource owner. We decided we said right so he's a person or user.

02:13:57.430 --> 02:14:26.620  
Durga, Amar  
So when clients send the request, so this is the standard flow. But I will explain you in terms of when we talk about some example then will understand in detail about what this authorization requests, how this authorization requests can. If you talk about any personal one. If you talk about any real time example this order before this authorization request, internally there are three more steps will be there. I will provide those things but.

02:14:26.670 --> 02:14:35.700  
Durga, Amar  
In gentle clients, in the request to the resource owner, then what user will say is OK based on the request you will provide the grant.

02:14:36.520 --> 02:14:43.790  
Durga, Amar  
This is nothing but authorization, grant or authorization token. Some places will call as in other addition token as well.

02:14:45.440 --> 02:14:53.130  
Durga, Amar  
Taking that talk authorization grant so the client will send a send to authorization server.

02:14:54.280 --> 02:15:07.230  
Durga, Amar  
An authorization server. Oh OK, uh, I go to this authorization token or grant based on that he will give access token even internally. Validate the auth.

02:15:07.830 --> 02:15:30.350  
Durga, Amar  
Grant token and you access token year couple of exchanges had happened. Exchanges that happened they need user access token. Once access token is there with the client client will send this access token to resource server. Resource server is nothing but API. Application resource error is basically our micro service application.

02:15:31.740 --> 02:15:38.920  
Durga, Amar  
So the resource server taking this access token we will validate and decode this token and then.

02:15:39.570 --> 02:15:44.200  
Durga, Amar  
And then you will provide the actual resource to the client.

02:15:45.300 --> 02:15:51.820  
Durga, Amar  
Internally a. See if you talk about the flow. This is a standard flow.

02:15:52.590 --> 02:16:00.390  
Durga, Amar  
Uh, but let's talk about a one with example. There is a standard example. I think most of you or some of you already know.

02:16:01.080 --> 02:16:07.730  
Durga, Amar  
Uh, so let's take I'm a user and I I sending a request. I'm sending a request to the.

02:16:08.490 --> 02:16:18.770  
Durga, Amar  
Claim saying that I want to take a picture so there is a printing photo printing services there and he's my client and I requested to take the.

02:16:20.130 --> 02:16:41.510  
Durga, Amar  
Pictures of this. So instead of uploading the picture I requested him to take OK, I have my pictures in my Google Drive so you send the you take from the Google Drive instead of me uploading the pictures to him at the same time I can't provide my username and pass sort of Google Google account details to the resource owner.

02:16:42.990 --> 02:16:49.760  
Durga, Amar  
So I can't provide these details to the client right? So client can store my username password so it is not.

02:16:49.890 --> 02:17:03.360  
Durga, Amar  
About how say, as a security it is not a recommended. So what you'll see is then that is the pair what comes into the picture. So whenever user send the request for the photo printing service.

02:17:04.100 --> 02:17:06.380  
Durga, Amar  
What photo printing will service will do is?

02:17:07.070 --> 02:17:10.150  
Durga, Amar  
You will send a request to the authorization server.

02:17:11.210 --> 02:17:12.360  
Durga, Amar  
OK, first.

02:17:13.580 --> 02:17:38.970  
Durga, Amar  
Authorization server authorization server. Here is what Google and Google actually Google is. The one is the resource server is a responsible to implement authorization server as well. So internally if you see both resource server and authorization server rise on the Google side. So in this our example, Google is responsible for building the resource server as well as authorization server.

02:17:40.110 --> 02:17:50.050  
Durga, Amar  
So when the client send the first based on the user's request, when he sent to the authorization server, what authorization server will do is?

02:17:51.100 --> 02:18:06.170  
Durga, Amar  
OK, I don't know about this photos printing service. What internally will do? Is this authorizations are reversed. Sending request to the client Elastic client. OK, I got one request. So you tell me whether I should approve or not.

02:18:07.030 --> 02:18:37.580  
Durga, Amar  
So client will get in a alert message saying that OK, there is a one request on photos, photo printing service request came and whether with all the access means here, the authorization server is not giving complete details. It say that OK I want to print two photos means that is the portion of the request that is a limited request will will show all the details to the user and user says OK. I'm providing this limited access I'm granting it.

02:18:38.120 --> 02:18:39.230  
Durga, Amar  
Based on my.

02:18:40.240 --> 02:18:48.530  
Durga, Amar  
Approval or request denial this authorization server ultimately finally decide OK, I got approval request from the.

02:18:49.220 --> 02:18:57.470  
Durga, Amar  
User, so that is the time it'll give it token that is called authorization token to the photo printing service.

02:18:58.420 --> 02:19:02.710  
Durga, Amar  
And this photo printing service OK, I got this talk here now.

02:19:03.460 --> 02:19:08.920  
Durga, Amar  
With this Oauth grant token, or grand or token, he will take that.

02:19:10.780 --> 02:19:16.550  
Durga, Amar  
Token and again send it to the authorization server for the Access token.

02:19:17.530 --> 02:19:48.470  
Durga, Amar  
So then yeah, the couple of exchanges are happening between the client and authorization server. Finally, authorization server will say that OK, because here there are so many other services. Are there right? Or printing services so authorization servers finally need to take that grant and understand OK? Which from where this it cost again came from then. Finally the authorization server from the Google will say that, OK, this this authorization grant is valid and finally you will.

02:19:48.780 --> 02:20:00.850  
Durga, Amar  
Correct token and the token will have a, uh, details about the user details and what is the request and what type of photo it is based on that he will give a access token.

02:20:01.780 --> 02:20:20.590  
Durga, Amar  
So with this access token, client will get the access token and take to the resource server. El Resource Server is our Google Drive. So finally once the token is there then the printing service will take that token and call the Google Drive and Google Drive will authenticate the token.

02:20:21.880 --> 02:20:46.130  
Durga, Amar  
Uh, it validate the token and then it provides that those pictures to the printing press service. So then printing service will print it and send it to the resource owner as part of the request. This is the complete actually real time example. Actually when we study about what right? So this is a standard complete flow that is called authorization code flow.

02:20:50.740 --> 02:20:51.850  
Durga, Amar  
Any questions so far?

02:20:56.510 --> 02:21:06.080  
Subramanian Kamatchi, Gobi Ganesh  
OK, so this authorization grant is applicable only if we are dealing with a third party logins or or part. From that scenario we will not be requiring that.

02:21:06.670 --> 02:21:07.150  
Durga, Amar  
Correct?

02:21:13.670 --> 02:21:14.180  
Durga, Amar  
And.

02:21:14.930 --> 02:21:26.300  
Durga, Amar  
Apart from this authorization code flow, there are a few more flows also, there one is implicitly flow. Actually, this simplicity flow is not.

02:21:27.500 --> 02:21:56.770  
Durga, Amar  
Recommended because of the security concern, but again whenever we are building with any UI with the JavaScript base, the implicit flow they use this one because of the stored. They because of that flow that will simplify some of the steps. So implicit is flow is one place you lot of places used by the JavaScript application. That's where the browser takes more priority because of nowadays. As I said, browsers place some of the.

02:21:57.160 --> 02:22:08.010  
Durga, Amar  
More functionality with this browsers. JavaScript applications can do implicitly flow. You can reduce some of the steps there. Apart from that there is another flow called password flow.

02:22:08.770 --> 02:22:36.960  
Durga, Amar  
Even this one is also not much popular with microservices, and there is a fourth flow called client credential flow. Actually, this is the client credential flow when you're talking from micro service standpoint. So whenever you want to interact with one micro service to another microservices, then most of the time people flow flow users, client credential flow because that is because of the trust and boundaries.

02:22:38.270 --> 02:22:53.320  
Durga, Amar  
So how would that will work is if you see this standard workflow in this, uh, let's take a there is a micro service application one and let's take a Microsoft is 2 application and microservices 2 is calling with their.

02:22:54.280 --> 02:22:56.740  
Durga, Amar  
It is integrated with another database, right?

02:22:57.510 --> 02:23:02.870  
Durga, Amar  
So when when Microsoft is want to interact with Microsoft, is 2 in order to.

02:23:03.780 --> 02:23:18.620  
Durga, Amar  
Not perform some action, so let's take it. I want to take some paid financial application layer. There are multiple application so where I want to check my balance so balances with another microservices application first. What it will do is?

02:23:19.470 --> 02:23:38.390  
Durga, Amar  
First micro service requests will send, uh, based on the user's request, it'll go as a authorization server. OK directly it will go for authorization server and authorization server provides the keys OK, there is auth keys and with that.

02:23:40.000 --> 02:23:56.300  
Durga, Amar  
Odki it'll our organization server, uh, will build, will what it'll do is see microservices one in order to go to microservice, it needs some endpoints and it needs some details, right? So you need to provide the key actually.

02:23:57.300 --> 02:24:19.680  
Durga, Amar  
So let me stick my step back. I told or wrong. So microservices one is the one who is responsible to provide some keys to authorization server then authorization server what it will do is based on the key it Linda said OK, I need to check the balance of functionality so that is within the trusted boundary it is game. So authorization server will give it access token.

02:24:21.040 --> 02:24:37.720  
Durga, Amar  
Here the second step is self. How we got access token end from that access token microservice one takes that access token and calls Microsoft is 2 then Microsoft what it will do is it will validate the token and based on that.

02:24:38.810 --> 02:24:52.090  
Durga, Amar  
Italy interact with the database and it'll perform the resource to the micro service one, so that's how they that's the only year. Only Microsoft 1/2 and authorization server comes into the client credential flow.

02:24:53.480 --> 02:25:02.980  
Durga, Amar  
So this is the flow actually use in in terms of microservice intercommunication. Whenever you want to do it. And this is a flow. Actually they follow it in the industry.

02:25:10.310 --> 02:25:12.040  
Durga, Amar  
So these are the four types of flow.

02:25:10.690 --> 02:25:11.370  
Thirugnanam, Kumaran  
Are these?

02:25:11.420 --> 02:25:11.660  
Thirugnanam, Kumaran  
Yes.

02:25:12.650 --> 02:25:13.290  
Durga, Amar  
Yeah, go ahead.

02:25:12.740 --> 02:25:22.990  
Thirugnanam, Kumaran  
A certain term, or are these flows are more related to the two lead flow three link flow which they refer? Or is it something entirely different?

02:25:25.760 --> 02:25:30.060  
Durga, Amar  
Uh, I don't know about the two layers or three layer flow, uh?

02:25:29.610 --> 02:25:32.660  
Thirugnanam, Kumaran  
Not two licked two legged or three electric, yeah.

02:25:31.220 --> 02:25:31.590  
Durga, Amar  
Cool.

02:25:33.910 --> 02:25:37.180  
Durga, Amar  
Sorry I didn't heard about it, but Nikhil, do you have any idea?

02:25:38.480 --> 02:25:39.110  
Durga, Amar  
I thought yeah.

02:25:39.480 --> 02:25:47.050  
Damle, Nikhil  
No, I don't. I don't exactly know if it's you know exactly the same as these different flows, so maybe we'll try to look up.

02:25:47.850 --> 02:25:48.660  
Damle, Nikhil  
That information

02:25:50.130 --> 02:25:55.890  
Durga, Amar  
Yeah, so we'll check once again after this session. Will take a look and we'll explain tomorrow.

02:25:56.730 --> 02:25:58.710  
Thirugnanam, Kumaran  
Yes, yeah, I'll just put it in the chat, yeah?

02:25:58.750 --> 02:25:59.760  
Durga, Amar  
Cool, thank you.

02:26:02.950 --> 02:26:31.160  
Durga, Amar  
So basically these are the flows comes and the 4th one. As I said, client credential flow that is the one in terms of microservice application architecture. If you're concerned that is a flowy concern, but we didn't capture all kind all the different types of flows here we took as part of the training program we kept as a standard one and also there are some other industries. We also see that they as per the product to product and this flows are varying. Also there is no.

02:26:31.660 --> 02:26:42.250  
Durga, Amar  
Uh, you can say that. OK, this is, uh, slowly we need to follow because of the protocol, so every vendor is implementing their own flow, OK?

02:26:48.640 --> 02:26:50.280  
Durga, Amar  
So moving to the next topic.

02:26:51.930 --> 02:26:53.180  
Durga, Amar  
What is access token?

02:26:54.440 --> 02:26:58.630  
Durga, Amar  
And why we need this access token plays a major role in auth flow.

02:27:00.680 --> 02:27:06.890  
Durga, Amar  
If you see a traditionally the ID and password is used to authenticate and authorize the user.

02:27:08.330 --> 02:27:17.050  
Durga, Amar  
And because of you, you're passing the username and password as we already discuss about this, it is not advisable to share your information.

02:27:17.660 --> 02:27:31.930  
Durga, Amar  
Uh, with third party authentication? Uh, as per the security reasons, we share that everyone knows by this time ID and password is not at recommender way to share the parties. These third party websites.

02:27:32.850 --> 02:27:36.960  
Durga, Amar  
One way to overcome this authentication is 2 generated token.

02:27:37.740 --> 02:27:42.420  
Durga, Amar  
Well, they have the server, you just this token and identify it.

02:27:45.300 --> 02:27:49.750  
Durga, Amar  
And this token is stored in other application rather than your ID and password details.

02:27:51.630 --> 02:28:01.120  
Durga, Amar  
So based, uh, instead of like, uh, like digest authorization and they just flow. If you see there you also you see that on the ash.

02:28:01.250 --> 02:28:30.340  
Durga, Amar  
Uh, forward, which we store right on the USA here. Also the tokens are stored rather than you were ID and password and during the providing the final resource these tokens are validated. Actually the invoker ID will see later on, not on this one at later topic. Also there are few additional layers on this token as well. It is there on uh if we talk about JWT token the JWT token actually can validate also.

02:28:31.890 --> 02:28:37.530  
Durga, Amar  
This is adicional layer there, but in terms of O auth, two these tokens are.

02:28:38.770 --> 02:28:42.350  
Durga, Amar  
Use two identique to the user ID and password. Yeah.

02:28:46.610 --> 02:28:59.080  
Durga, Amar  
Ender, as we explain about as I explain about the standard flow, how this token works. This is the same steps what another picture, which was thrilled representation.

02:29:00.130 --> 02:29:18.900  
Durga, Amar  
So there is nothing different than what actual flow, but in terms of token, what are the how the flow is works. So again as I explain about the previous example, we can really correlate first application request authorization to access service resources, OK?

02:29:19.600 --> 02:29:24.040  
Durga, Amar  
Yeah, the service resources. Nothing but you were, uh.

02:29:25.580 --> 02:29:26.420  
Durga, Amar  
User actually.

02:29:28.000 --> 02:29:50.110  
Durga, Amar  
OK then on the applications is user authorization grant. If the request is organized internally the order, then the client will send to authorization server and authorization server calls to the user and from users it based on that it'll it'll granted. So that's where called as a authorization grant comes into picture, then application requests an access token.

02:29:50.920 --> 02:29:54.680  
Durga, Amar  
From the authorization server, by presenting the authentication.

02:29:55.300 --> 02:30:03.160  
Durga, Amar  
OK, then this authorization server issues access token to up to the application upon validation.

02:30:04.120 --> 02:30:19.520  
Durga, Amar  
The application request, then the fifth step is the application requests the resource from the resource server and presence the access token for authentication as well. If access token is valid, the resource server serves the resource to the application.

02:30:21.640 --> 02:30:31.150  
Durga, Amar  
Whatever we discussed in the previous example and whatever we show the standard flow in terms of token, this is how the flow actually works.

02:30:36.170 --> 02:30:43.350  
Durga, Amar  
Let's come to the spring boot application. When you want to implement what two in spring?

02:30:43.990 --> 02:31:02.930  
Durga, Amar  
Then we need only two dependencies. One is we need only one dependence equals spring security, as in in the previous example, nickel already shown about what is the dependency. We we added the same dependency applicable for spring sports two Awards 2 as well.

02:31:05.220 --> 02:31:17.590  
Durga, Amar  
So we need only to two things. One is spring boot and spring boot. What dependency worth is also part of spring security. So along with the basic auth worth also comes there.

02:31:18.780 --> 02:31:35.490  
Durga, Amar  
So that is advantage of that dependency. What spring date is all the whatever the other indications are there, all the captain to 11 dependency called as spring security. So as soon as you add this spring security dependency then you are enabling over 2 as well.

02:31:40.440 --> 02:31:43.780  
Durga, Amar  
That is about what to any questions on what.

02:32:01.760 --> 02:32:04.970  
Durga, Amar  
OK, so let's move to the next, uh.

02:32:05.890 --> 02:32:07.550  
Durga, Amar  
Topical Open edit connect.

02:32:08.830 --> 02:32:10.280  
Durga, Amar  
OK well yeah.

02:32:08.990 --> 02:32:09.440  
Damle, Nikhil  
Amar.

02:32:10.150 --> 02:32:17.110  
Damle, Nikhil  
So maybe we won't take a 10 minute break or like moving before moving the next topic can come back at 12:20.

02:32:15.790 --> 02:32:16.340  
Durga, Amar  
Oh

02:32:19.090 --> 02:32:31.700  
Durga, Amar  
sure, so I thought to you this open ID as well. Then I thought to give a break before giving the demo. Either wish, but if your team is looking for a break then let's take a break in.

02:32:25.980 --> 02:32:26.460  
Damle, Nikhil  
OK.

02:32:36.270 --> 02:32:36.530  
Damle, Nikhil  
In

02:32:36.680 --> 02:32:38.790  
Damle, Nikhil  
I'm back at 12:20. May be a short break.

02:32:36.830 --> 02:32:37.080  
Durga, Amar  
Well.

02:32:38.370 --> 02:32:38.690  
Durga, Amar  
Sure.

02:32:39.330 --> 02:32:39.750  
Durga, Amar  
Sure.

02:32:41.890 --> 02:32:45.590  
Durga, Amar  
Let's take a break and will be back at 12:20.

02:32:47.590 --> 02:32:47.920  
Seniappan, Yuvarani  
OK.

02:32:48.740 --> 02:32:49.790  
Durga, Amar  
OK, thank you.

02:41:04.410 --> 02:41:05.730  
Durga, Amar  
Let's wait for one more minute.

02:42:17.580 --> 02:42:19.550  
Durga, Amar  
OK, I hope everyone back.

02:42:22.650 --> 02:42:31.200  
Durga, Amar  
So as we discuss about what flow and what what 2.0 do, let's talk about the next topic.

02:42:32.370 --> 02:42:33.300  
Durga, Amar  
Which is a.

02:42:34.890 --> 02:42:36.450  
Durga, Amar  
Related to open ID connect.

02:42:40.240 --> 02:42:41.360  
Durga, Amar  
End up with.

02:42:42.870 --> 02:42:54.460  
Durga, Amar  
Enable process what happened is the tokens actually and doesn't hold much information but with open ID actually internally uses JWT token.

02:42:55.340 --> 02:43:02.470  
Durga, Amar  
And by this time, if you're if you're aware about JWT token means it like a Jason Web token.

02:43:05.460 --> 02:43:20.010  
Durga, Amar  
Uh, so if you're not familiar and will will explain what is JWT also stand for at a later slide, but typically open ID internally uses JWT token and based on that users.

02:43:21.070 --> 02:43:28.090  
Durga, Amar  
Users information, it'll validate it, and it'll provide. The service is actually. If you see the actual flow.

02:43:28.950 --> 02:43:40.960  
Durga, Amar  
In this, uh, uh, open ID connect. Yeah, there are basically two Part 2 rolls or their three rules are there. One is a client, another is authorization server and the resource server.

02:43:42.180 --> 02:43:47.550  
Durga, Amar  
And we successfully implemented this open ID connect by using IP.

02:43:47.600 --> 02:44:08.110  
Durga, Amar  
Uhm IDP provider so IDP is a we have one more product from FIS site so they IDP QS like a authorization server for us and with based on that IDP we get the JWT token actually. So taking that token and we use our services for validation.

02:44:08.930 --> 02:44:19.720  
Durga, Amar  
So how it actually works as client send the request OK, client send the request, then the client. What it will basic with access token, use then and.

02:44:20.460 --> 02:44:29.160  
Durga, Amar  
Once the authorization server violated and based on the whatever the details you provide it internally gives JWT token.

02:44:30.170 --> 02:45:01.650  
Durga, Amar  
Once we get this jet update token line send the resource server OK. Resource server is nothing but you were API server and the resource server. What it internally? What it Louise at the time of application startup. Based on that it will get the public keys. This is a one time activity. In the third step where it will be right, so it'll get the public keys which are appended to the JWT token and it will respond back to the resource server and resource server. Internally it will validate those signatures and.

02:45:02.020 --> 02:45:02.710  
Durga, Amar  
Permissions.

02:45:03.520 --> 02:45:22.420  
Durga, Amar  
Entity you the response as a resource body. So that's how open ID connect flow been slow. Flow will go on and if you see authorization flow there are few unlimited steps here. But ultimately the concept both are right lies in the similar direction.

02:45:38.560 --> 02:45:48.430  
Durga, Amar  
How does this JWT access token looks like? It had three sections. OK, there is a header. There is a payload and the signature.

02:45:50.420 --> 02:46:20.210  
Durga, Amar  
Header contains information on how the JWT signature should be computed. OK, your signature will have. Basically that's how your signature should be defined. So this Jason Web token header will tell us about it, and payload tells about the data that is stored inside the JWT token. So here you can provide your your own data to the token so that the personal data will be there in the payload, then the signature the year.

02:46:20.280 --> 02:46:38.080  
Durga, Amar  
The data is ash. Actually there is a ashing technology. They use it. There are multiple hashing techniques, are there and with the secret key. Using this algorithm in the general area in the using JWT header, it will use the get the signature and that signature will be validated.

02:46:39.250 --> 02:46:41.350  
Durga, Amar  
That's how the three sections look like.

02:46:42.540 --> 02:47:00.990  
Durga, Amar  
In the JWT and I will show sample JWT token also at let us lights and also I'll show this or three sections like header, payload and signature by using one of the editor called JWT header. Also we can go and see how it looks now how these three partition looks like?

02:47:02.500 --> 02:47:10.930  
Damle, Nikhil  
Amit, sorry to interrupt but I don't know if it's me. You know I I still see your open ID connect slide in this slide 32 actually.

02:47:11.780 --> 02:47:12.250  
Durga, Amar  
Why is it?

02:47:11.980 --> 02:47:13.820  
Damle, Nikhil  
Are you on next slide or she lives?

02:47:12.980 --> 02:47:15.130  
Patro, Patanuru Santosh K  
Is yeah for same for me as well.

02:47:13.500 --> 02:47:14.940  
Durga, Amar  
No, I was in the next.

02:47:17.430 --> 02:47:18.830  
Durga, Amar  
Would you like me to share it?

02:47:33.900 --> 02:47:38.930  
Durga, Amar  
Can you see now JT token? Because now I knew the full access. I don't know what happened earlier.

02:47:33.960 --> 02:47:34.340  
Damle, Nikhil  
Yeah.

02:47:40.170 --> 02:47:43.640  
Durga, Amar  
Thanks Nikhil actually. I was planning to go to the next slide as well.

02:47:45.370 --> 02:47:48.200  
Damle, Nikhil  
Yeah, I can see the JWT slide now.

02:47:47.660 --> 02:47:47.990  
Durga, Amar  
OK.

02:47:50.810 --> 02:48:20.890  
Durga, Amar  
Yeah, let me repeat once again, then a JWT access token looks in at three different sections. One is header payload and signature header is the one where it contained information on how we were signature should be computed and the payload is the one where the data will be stored. Like users information, whatever the username and password. In my case I give username, password. Those personal information will be there in the payload and the signature tells her about this.

02:48:21.180 --> 02:48:38.690  
Durga, Amar  
Well, it uses action technology, actually, uh, I think algorithm here and with the data is ash with the technology. With that algorithm and it will store it in by using JWT header. It'll it'll get the signature. It will be generated that signature.

02:48:39.540 --> 02:48:48.590  
Durga, Amar  
But we can decompose this and will also show one of the editor JWT editor and we can see the three sections. How internally looks like.

02:48:50.520 --> 02:48:58.400  
Durga, Amar  
And basically this is how you are a token looks like left hand side. If you see this is the actual token.

02:48:59.470 --> 02:49:10.370  
Durga, Amar  
And you see here and you don't understand anything about it. But if you use any of the editors details, or if you then if you see on the right hand side.

02:49:11.360 --> 02:49:38.610  
Durga, Amar  
There is a header payload, middle partners payload, and the last session is a signature in the header. It'll tells us what algorithm I need to use. OK, here it is using HS 256, so this is 256 is used by the signature and jewelry. Using that algorithm, whatever the data you provided in the payload, that payload will be encrypted.

02:49:41.120 --> 02:49:51.390  
Durga, Amar  
OK, so here, uh, in that and the 256 users or R2256 bit secret code and it'll generate the signature accordingly.

02:49:52.600 --> 02:49:57.670  
Durga, Amar  
So composing of all three, uh, evil JWT token will be generated to you.

02:49:58.750 --> 02:50:11.120  
Durga, Amar  
So that is how open ID, uh, connect users jaded token with this JWT token as we saw the flow one earlier. So with this token, once again we give one example yet.

02:50:12.070 --> 02:50:27.060  
Durga, Amar  
So we have application. We have a microservices applications are there so as we see in the last 222 sessions we showed about entities, policies, claims, domains and.

02:50:27.990 --> 02:50:46.640  
Durga, Amar  
We also have a variance sum up microservices, right? So that is nothing but the application server. We use an authentication server as a uh, like users IDP other indication and whenever the user is like a physical client where a person will send the request. So that's how.

02:50:46.690 --> 02:50:54.180  
Durga, Amar  
Uh, let's take these three examples 33 rules. With that, the first one is whenever we send the request, OK?

02:50:54.890 --> 02:51:03.510  
Durga, Amar  
To authentication server, whether it's a high DPI or a Facebook or a Google. Based on that you were identifications are will give a JWT token.

02:51:04.590 --> 02:51:25.410  
Durga, Amar  
So user will get that 100 token and take the token and call your APS application servers like whatever the microservices which you are calling based on your configuration orbit, the application servers will verify the token and even it will process the request. Here it is a open ID. There very simple process.

02:51:27.070 --> 02:51:31.430  
Durga, Amar  
Game there are only four steps. I hope you are clear with this flow.

02:51:35.800 --> 02:51:45.020  
Subramanian Kamatchi, Gobi Ganesh  
Yeah, so uh, one thing which, uh, I have like very here is like so for for this to be established or do we need to have.

02:51:45.840 --> 02:51:54.930  
Subramanian Kamatchi, Gobi Ganesh  
But the the handshake with the other public keys has to be exchanged between the user and the authentication server in Pryor.

02:51:57.150 --> 02:52:03.660  
Durga, Amar  
Ah, I see you're saying about on the authentication server, right? So your fear telling that if you want to use public one?

02:52:01.240 --> 02:52:01.700  
Subramanian Kamatchi, Gobi Ganesh  
Gather.

02:52:05.350 --> 02:52:14.350  
Subramanian Kamatchi, Gobi Ganesh  
No, uh, OK. I'm asking like where, whether whether the user has to be aware of the public key of the authentication server.

02:52:15.950 --> 02:52:18.900  
Damle, Nikhil  
The actually the application server really new.

02:52:20.700 --> 02:52:23.890  
Damle, Nikhil  
The resource server actually needs to be aware of that public key.

02:52:25.070 --> 02:52:31.790  
Damle, Nikhil  
So that you know when the the end user need not be but the end user. In this case and if the application server.

02:52:34.540 --> 02:52:36.350  
Damle, Nikhil  
If you know has to have the.

02:52:37.920 --> 02:52:41.950  
Damle, Nikhil  
Public key certificate so that when the JWT token is presented.

02:52:42.770 --> 02:52:46.700  
Damle, Nikhil  
On the request header you know it is able to validate the JWT token.

02:52:47.620 --> 02:52:49.620  
Damle, Nikhil  
Using the the public key certificate.

02:52:51.170 --> 02:53:04.050  
Subramanian Kamatchi, Gobi Ganesh  
OK, OK, so you mean to say that I say if I am logging into Zomato via my Gmail ID. So Zomato and Google like we need to have that key exchange already and have that set up.

02:53:00.310 --> 02:53:00.790  
Damle, Nikhil  
Correct?

02:53:02.700 --> 02:53:03.270  
Damle, Nikhil  
Exchange.

02:53:04.480 --> 02:53:06.590  
Damle, Nikhil  
Correct, correct? Correct, that's true.

02:53:05.210 --> 02:53:07.360  
Durga, Amar  
Yes, in that case, correct?

02:53:14.350 --> 02:53:14.710  
Subramanian Kamatchi, Gobi Ganesh  
OK.

02:53:16.230 --> 02:53:22.290  
Subramanian Kamatchi, Gobi Ganesh  
OK and OK and we are now looking at everything in the perspective of Zomato, right? So like how we are going to.

02:53:23.140 --> 02:53:25.750  
Subramanian Kamatchi, Gobi Ganesh  
Uh, I mean can consume this service or.

02:53:27.470 --> 02:53:52.980  
Durga, Amar  
Yeah yeah will show no. I will show both the things here as part of authorization server. How to fetch the token first because there you need to go and register yourself right? So like like Facebook or Google where you go and register first right? So here we use IDP because FISDEV we're raising IDP wherein IDP we already registered the users there.

02:53:28.070 --> 02:53:28.450  
Subramanian Kamatchi, Gobi Ganesh  
Or

02:53:40.200 --> 02:53:40.600  
Subramanian Kamatchi, Gobi Ganesh  
correct?

02:53:54.400 --> 02:53:54.810  
Subramanian Kamatchi, Gobi Ganesh  
Correct?

02:53:54.810 --> 02:54:22.480  
Durga, Amar  
So once we have that uh registration in IDP, uh, we can't use like I can't show Gmail or on Facebook at this time. But uh, with IDP, once we registration is done then IDP server you need to write your client code. OK, as part of your micro service application you need to write a client code. There you need to provide few URLs related to IDP. Based on that if you give username password it will give one token.

02:54:23.310 --> 02:54:53.330  
Durga, Amar  
So IDP ultimately generate the token and that we need to take the code. Then we need to use the token in our your micro service application. Because microservices application instead of in order to validate it. So it will take the token. Then what spring tells his based on if your configuration is related to open ID then we're token will be verified once the verification is successful then even.

02:54:24.100 --> 02:54:24.490  
Subramanian Kamatchi, Gobi Ganesh  
OK.

02:54:50.410 --> 02:54:50.720  
Subramanian Kamatchi, Gobi Ganesh  
OK.

02:54:53.380 --> 02:55:00.030  
Durga, Amar  
Microsoft is application interact with the DB and it'll fetch that information and it will give as a response to you.

02:55:00.920 --> 02:55:02.450  
Durga, Amar  
That's the flow will see now.

02:55:01.140 --> 02:55:01.500  
Subramanian Kamatchi, Gobi Ganesh  
OK.

02:55:03.110 --> 02:55:03.710  
Subramanian Kamatchi, Gobi Ganesh  
OK, OK.

02:55:04.150 --> 02:55:04.330  
Ayyanan, Ramya  
So.

02:55:04.380 --> 02:55:07.530  
Ayyanan, Ramya  
So here the client is nothing but the service provider.

02:55:08.920 --> 02:55:34.510  
Ayyanan, Ramya  
I mean my, my client application is a service provider for the IDP, so from service provider I will be requesting IDP to authenticate that you serve. So once IDP has certainly cutted the user then IDP will send back to the chat token to me. Yeah so I will be having that as a token and I will be using that here. How the session management will be happening now?

02:55:17.970 --> 02:55:18.380  
Durga, Amar  
Yes.

02:55:19.170 --> 02:55:19.450  
Durga, Amar  
Yes.

02:55:25.490 --> 02:55:26.810  
Durga, Amar  
Got it, got it.

02:55:27.430 --> 02:55:27.770  
Durga, Amar  
Correct?

02:55:35.900 --> 02:56:09.460  
Durga, Amar  
Yeah, yeah, what we are doing is we are not making this token. Yeah the tokens also can be a long way or short lives can be done for now in that your service provider based on when you you said the user know. So at the time the token will be provided actually how long it will be valid. So for this example what we did is in IDP what we provided is a long lived tokens we provided so the token will not expire but IDP. Use that flexibility if you want to generate the token for 30 minutes or like that.

02:56:11.440 --> 02:56:17.170  
Ayyanan, Ramya  
OK, so uh, after 30 minutes will it send us a log out request?

02:56:17.720 --> 02:56:43.410  
Durga, Amar  
Yes, so every time you need to request off once again you need to generate a new token so that what in our application what we did is we want the token to be valid for Longley, so that as long as the token is validated so that our services can be validated and proceeded further so that is application server, will lowering the verify the process that will take care. So if it is a token is valid or not.

02:56:44.430 --> 02:56:44.770  
Ayyanan, Ramya  
OK.

02:56:45.800 --> 02:56:47.370  
Subramanian Kamatchi, Gobi Ganesh  
OK, so basically we need to have.

02:56:47.270 --> 02:57:16.660  
Durga, Amar  
Basically actually internally application server also communicate with the authorization server. So in part of spring security you need to provide that URL as well. So what is your service provider URL? So internally this weather time of validation so application server interact with authorization server and it Lambda sender. It will verify the token but that will be in our case the spring will take care of that so we should not worry about the how the token is validating.

02:56:48.350 --> 02:56:49.010  
Subramanian Kamatchi, Gobi Ganesh  
A timer.

02:57:13.000 --> 02:57:13.470  
Ayyanan, Ramya  
Yeah.

02:57:17.080 --> 02:57:24.670  
Durga, Amar  
But the spring will take care, but yes, you need to provide that that authorization server details to your micro service application as well.

02:57:25.780 --> 02:57:32.240  
Ayyanan, Ramya  
So if this is nothing but the meta data right? The way I'm going to provide my service provider URL.

02:57:32.800 --> 02:57:42.200  
Durga, Amar  
Yes, it's a metadata, so you never case if we are using application dot XAML file. If you internally will create a metadata only.

02:57:35.180 --> 02:57:35.610  
Ayyanan, Ramya  
Yes.

02:57:39.730 --> 02:57:40.220  
Ayyanan, Ramya  
OK.

02:57:43.880 --> 02:57:44.550  
Ayyanan, Ramya  
OK fine.

02:57:45.860 --> 02:57:56.790  
Ayyanan, Ramya  
So here we have a no the way, uh, using SAML security is very springs, Amol and spring. What right? So which will be the best one for racers will make.

02:57:57.890 --> 02:57:58.470  
Ayyanan, Ramya  
In that case.

02:57:58.280 --> 02:58:05.490  
Durga, Amar  
OK, we are not using what, uh, but again a there is a specific scenario which, uh, for one of the client.

02:58:06.390 --> 02:58:18.250  
Durga, Amar  
Uh, we are going. Basically every rest call is a new request, so we are using as a Openid, Openid and basic core and there are two more flavors we are using in our application.

02:58:19.560 --> 02:58:20.050  
Ayyanan, Ramya  
OK.

02:58:22.020 --> 02:58:27.220  
Durga, Amar  
But Nikhil, where is the one more client where we use both or intercommunication, right?

02:58:22.570 --> 02:58:22.830  
Ayyanan, Ramya  
OK.

02:58:29.590 --> 02:58:37.390  
Damle, Nikhil  
Yeah, for inter services you know, enter microservices communication. I think you know the client credentials flow is is very well suited.

02:58:37.870 --> 02:58:38.320  
Durga, Amar  
In

02:58:38.800 --> 02:58:42.070  
Damle, Nikhil  
But if you open ID Connect also so.

02:58:43.270 --> 02:59:08.660  
Damle, Nikhil  
Leg another authentication as well, you know, so with with the JWT token, you know who the user is, whereas you know with what the access token doesn't really specify who the user is, but the access token specifies you know what can be done with that access to, so I think with if you have a requirement that you also want to know who the user is, I think the open ID Connect is a good mechanism that you know you could create a JWT token.

02:59:09.350 --> 02:59:12.970  
Damle, Nikhil  
And then, you know, use that JWT token to call the micro service.

02:59:15.210 --> 02:59:38.300  
Durga, Amar  
and in our application, user is always a mandated because in order to do some posts or PAX processing part, we always make sure that user is required in our scenario. So what we did is in additionally workflow for one for one of the client we introduced as part of header. We are also capturing some user information as well because.

02:59:38.350 --> 02:59:53.660  
Durga, Amar  
So that is a actually we design internally because inter service communication clients needs some work flow because they don't want to authenticate it. But in order to process our internal services we capture some additional information as part of header request.

02:59:58.270 --> 02:59:58.790  
Ayyanan, Ramya  
OK.

03:00:00.490 --> 03:00:06.060  
Damle, Nikhil  
I think let's look at, you know, maybe once we once we look at the demo, I think things also will be more.

03:00:06.430 --> 03:00:06.890  
Durga, Amar  
Sure.

03:00:07.020 --> 03:00:08.850  
Damle, Nikhil  
Not clear as well, but sorry.

03:00:13.420 --> 03:00:13.770  
Durga, Amar  
Yeah.

03:00:16.540 --> 03:00:18.010  
Durga, Amar  
So I already started.

03:00:17.270 --> 03:00:17.700  
Yadav, Rameshchandra  
I'm gonna.

03:00:18.670 --> 03:00:19.270  
Durga, Amar  
Yeah, go ahead.

03:00:20.170 --> 03:00:39.960  
Yadav, Rameshchandra  
Is it possible we can design a system in such a way that like we have a couple of options for authentication authorization like open? So if a designer system in such a way that like we can use two different authentication mechanism like both with JWT combinations or basic?

03:00:40.850 --> 03:00:41.410  
Yadav, Rameshchandra  
Something.

03:00:42.980 --> 03:00:59.620  
Damle, Nikhil  
Yeah, you can define in the URLs and you URL patterns really like we saw in the configuration. So you can say that maybe a particular set of URL patterns maybe apply basic auth, but for a separate set of URL patterns can apply let's JWT or OAUTH or that kind of thing.

03:01:00.270 --> 03:01:02.560  
Damle, Nikhil  
So it can be configured in in spring security.

03:01:00.670 --> 03:01:01.080  
Yadav, Rameshchandra  
OK.

03:01:03.710 --> 03:01:04.240  
Yadav, Rameshchandra  
Thank you.

03:01:05.980 --> 03:01:14.950  
Durga, Amar  
Yeah it is a Nichols presentation we provided all yours we are going with this approach right? So there I think we can do filters there.

03:01:18.830 --> 03:01:22.970  
Durga, Amar  
Yeah, so coming back to the demo. I hope you can see my SDS tool right.

03:01:26.130 --> 03:01:26.550  
Gawale, Manik  
Yes.

03:01:27.000 --> 03:01:33.110  
Durga, Amar  
OK, so we return 1A CAPI gateway code to interact with the law.

03:01:27.040 --> 03:01:27.530  
Rani, Nisha 2  
Yes.

03:01:34.260 --> 03:01:47.580  
Durga, Amar  
Interact with the IDP so IDP from IDP we got few details. We first of all we get the base URL end where for our team we requested for IDP so there we go and we register the users there.

03:01:48.620 --> 03:02:12.350  
Durga, Amar  
So once we get these details from IDP confirmation IDP based on our configuration, this is a user agent we use as a como and this is a service provider for our testing. Actually we define there so once you configure everything in IDP application then we return the client code here. So based on this client could what it'll do is it'll based on your risk lined.

03:02:13.430 --> 03:02:33.110  
Durga, Amar  
We have one or two hours or do post method and do get method so this internally and based on all the requirement based on you IDP will give all the details. Based on that you need to code and once you authenticate once you provide all these details to IDP. What IDP will do is IDP will give a token.

03:02:34.030 --> 03:02:49.360  
Durga, Amar  
OK, and this ID and this token, uh, we can, uh, use this token can be displayed in the as part of the response body and take this token and we need to call our rest on next applications which are the euro.

03:02:50.120 --> 03:02:55.910  
Durga, Amar  
Microsoft is applications for their, so in our case we these are all micro service applications in our project.

03:02:56.880 --> 03:03:04.690  
Durga, Amar  
So we can, or using the token we can show what is how it validate first of all, let's start about my.

03:03:06.120 --> 03:03:12.270  
Durga, Amar  
My CAPI gateway code so this is the code. We uh I started with the 5000 port.

03:03:13.590 --> 03:03:25.360  
Durga, Amar  
Then I go to, uh, generally I use. I use postman and instead of every year we generate the tokens in this case. So this is so this is the URL and this is a rest endpoint.

03:03:26.250 --> 03:03:47.520  
Durga, Amar  
And if you see my body, uh, this is my login details, so this is my username and this is the password. OK, with this login credentials, when I hit to the send the request, it internally calls the high DP and IDP based on whatever the users are configured there. It will generate the token to me.

03:03:50.470 --> 03:03:56.270  
Durga, Amar  
So here are my request is 200 success? OK, so this is a token.

03:03:59.590 --> 03:04:03.530  
Durga, Amar  
So if you see any editors, actually if you open any editors.

03:04:04.820 --> 03:04:05.530  
Durga, Amar  
So.

03:04:10.740 --> 03:04:18.640  
Durga, Amar  
So like in this, uh, token, this is, uh, in this Jason Web token. If you paste that token, whatever it is generated.

03:04:20.170 --> 03:04:33.270  
Durga, Amar  
Uh, I told about JWT consists of 3/3 sections, right? So one is a header, so here it mentioned about what is the token type and algorithm. So algorithm is RS256 we are, it is used.

03:04:34.250 --> 03:05:04.500  
Durga, Amar  
And in the body, if you see, uh, this is where my service provider URL is and also it is what is from which department it is coming from and what is the login credentials details is there. So if you see here my locking name this and logging in last name and this is my ID client, product client ID and this is my first name and this is my email ID so all this information can be stored in this JW.

03:05:04.550 --> 03:05:08.600  
Durga, Amar  
Token payload, but whereas in a Watt token.

03:05:09.250 --> 03:05:14.050  
Durga, Amar  
This person information cannot be stored, so that's where we can't use other indication there.

03:05:15.980 --> 03:05:30.750  
Durga, Amar  
Then, uh, with. With this payload information below a year by using whatever the algorithm it is there, so it'll generate the signature. Because of that ashing technique algorithm, it stores the signature with the certificates.

03:05:31.850 --> 03:05:53.830  
Durga, Amar  
And just a disclaimer, and this editor will not store any of our tokens. Whatever we pasted here, it will not stored it, so just I want to say so here there is a warning if you that's the reason as editor is not storing any tokens here it will show as invalid signature so.

03:05:55.270 --> 03:05:57.880  
Durga, Amar  
Just for the demo purpose I showed all these details.

03:05:59.830 --> 03:06:00.220  
Durga, Amar  
OK.

03:06:01.050 --> 03:06:01.620  
Ayyanan, Ramya  
OK.

03:06:02.370 --> 03:06:04.150  
Ayyanan, Ramya  
Can you show the dependency?

03:06:04.220 --> 03:06:05.930  
Ayyanan, Ramya  
A partner.

03:06:06.770 --> 03:06:09.030  
Durga, Amar  
Yeah, so if you see a.

03:06:20.460 --> 03:06:28.570  
Durga, Amar  
OK yeah, this is a. This is a border CAPI gateway. Here we use a cloud and spring cloud technical OK independency.

03:06:29.280 --> 03:06:35.500  
Durga, Amar  
Uh, we are not validating it right? So we are. We are generating the token.

03:06:36.550 --> 03:06:40.100  
Durga, Amar  
So here we use a spring cloud gateway.

03:06:42.080 --> 03:06:42.780  
Durga, Amar  
Dependency.

03:06:42.460 --> 03:06:59.030  
Damle, Nikhil  
No, but this project is just, you know, fetch the token from IDP. So yeah, this is like the cloud gateway is used for some other purpose actually, but specially this is just, you know, a couple of rest calls to the IDP authorization server to get hold of the JWT token.

03:06:45.780 --> 03:06:47.470  
Durga, Amar  
So can idp PM.

03:06:59.380 --> 03:07:13.290  
Durga, Amar  
Talking any newer case you can integrate with a third party public applications as well in order to generate the token so it is not required to go with only with IDP, so clients will go with any third party right? Like no need to go with internal so.

03:07:00.070 --> 03:07:01.720  
Damle, Nikhil  
So I think next.

03:07:15.960 --> 03:07:19.310  
Durga, Amar  
So the other spring security is are not not in the picture.

03:07:21.420 --> 03:07:25.870  
Gawale, Manik  
And how you people validating the JWT token on each request?

03:07:25.930 --> 03:07:32.590  
Durga, Amar  
Yeah, that I will show you once the token is generated and during uh when you call to the.

03:07:32.640 --> 03:07:37.750  
Durga, Amar  
Uh, uh API server right application server? That is the time we validate it.

03:07:38.940 --> 03:07:40.010  
Durga, Amar  
I'll show you that one.

03:07:40.860 --> 03:07:41.620  
Durga, Amar  
In few minutes.

03:07:44.480 --> 03:07:48.290  
Durga, Amar  
So far, so clear. So this is only for to generate the token.

03:07:52.600 --> 03:07:53.250  
Ayyanan, Ramya  
Yeah, OK.

03:07:54.780 --> 03:08:13.260  
Durga, Amar  
OK, so once this token is generated and then in your UM, which are the microservices application you develop again here? Also, we are most of the things are explained by Nikhil in basic or or applies the same in terms of the configuration. So we have, uh.

03:08:14.030 --> 03:08:24.860  
Durga, Amar  
We open ID security configuration file is there and which we added an annotation called web enabled web security. So yeah, I already Nikhil already explain about.

03:08:24.920 --> 03:08:32.390  
Durga, Amar  
On the Outlook Web security's right. In basic or so it have like override method of conflict. So which we need to implement it?

03:08:33.300 --> 03:08:51.730  
Durga, Amar  
So you can use a web security config adapter and again is we also use it conditional property again and we can already explain about it, so this is related to spring boot annotation. So here what we mention is instead of basic auth we use OIDC. Use open IDC.

03:08:52.730 --> 03:08:53.060  
Durga, Amar  
OK.

03:08:54.110 --> 03:09:13.220  
Durga, Amar  
And when you're creating a, when you override the config, this is a step actually when the server is started, eat your application is understand that OK, this is only one or two lines step. It will tell that. OK, my application should configured with JWT.

03:09:14.330 --> 03:09:17.470  
Durga, Amar  
Sorry what uh, open ID connect.

03:09:19.340 --> 03:09:26.860  
Durga, Amar  
So here if you see the two line code ear authorize the request. OK here any request. So here we are not doing any filters.

03:09:27.630 --> 03:09:36.230  
Durga, Amar  
So you need a quest comes other indicated with War Two resource server as JWT and the decoder is a JWT decoder.

03:09:36.880 --> 03:09:42.260  
Durga, Amar  
And finally, we're on JW authorization authentication converter.

03:09:43.560 --> 03:09:47.850  
Durga, Amar  
Here we use RJW decoder. We use as a private.

03:09:49.680 --> 03:10:07.610  
Durga, Amar  
Uh here if you see the J code and JW and decode method here, it's like the Resource Service Pro properties and take the URA. Actually this is the URL where we get it, so this URL you need to define your in your application. Daniel file.

03:10:08.680 --> 03:10:26.190  
Durga, Amar  
OK, so that, uh, whenever application server in one to need to other interact with this authentication servers. So this is where you need to provide the URL. So if it's in your application dot Amol file. So this is where we provided the.

03:10:27.350 --> 03:10:27.930  
Durga, Amar  
You are right.

03:10:29.810 --> 03:10:38.180  
Durga, Amar  
So here you section in application dot ML file. This is a security what resource server should be with a delicate issue, right? URA is this one.

03:10:40.770 --> 03:11:03.180  
Damle, Nikhil  
So you know madicken others. Actually, you're asking right? Just to add like this issue are you, are, you know, is at server startup time when the configuration is initialized this issue arise. Used you know to fetch the public key certificate from the IDP. You know authorization server so it fetches it once and keeps that public key certificate with itself.

03:11:05.160 --> 03:11:08.410  
Damle, Nikhil  
And then that will key certificate is used later, you know 4.

03:11:08.490 --> 03:11:11.040  
Durga, Amar  
At the time of verification, validating right.

03:11:09.350 --> 03:11:10.540  
Damle, Nikhil  
Validating yeah.

03:11:20.920 --> 03:11:28.270  
Durga, Amar  
So that's how the issuer URL comes, then the issue, right URL? Uh, it'll be valid.

03:11:29.100 --> 03:11:32.700  
Durga, Amar  
Street and then it'll return the JWT decoder.

03:11:36.570 --> 03:11:37.640  
Durga, Amar  
Again, a.

03:11:38.850 --> 03:11:49.710  
Durga, Amar  
I'm I'm not showing anything about the dependency. If you want a palm dot XML the same whatever we shown in a basic on the same dependencies are applicable in this as well.

03:11:52.570 --> 03:11:55.310  
Damle, Nikhil  
They open the CAPI security form.

03:11:56.690 --> 03:11:57.990  
Durga, Amar  
Yeah, yeah.

03:12:00.390 --> 03:12:02.300  
Durga, Amar  
Sorry so security.

03:12:08.230 --> 03:12:08.530  
Durga, Amar  
Yeah.

03:12:10.010 --> 03:12:12.320  
Durga, Amar  
I said this is the same dependency for use.

03:12:12.950 --> 03:12:15.120  
Durga, Amar  
Uh, for open ID as well.

03:12:16.820 --> 03:12:29.660  
Durga, Amar  
And if you see our microservice application even complete, what are the types of authentications are there? So all these authentications we kept as a separate application called CAP is security as a micro service application.

03:12:32.280 --> 03:12:40.800  
Durga, Amar  
But this this will not have any data on this, so this is we're not exposed to the end user as well. So this is internal jar to the other applications.

03:12:52.810 --> 03:13:06.120  
Durga, Amar  
So this is about a about the configuration about open ID security, open ID connect once this is once we have this configuration is ready then you can start your applications.

03:13:09.140 --> 03:13:12.410  
Durga, Amar  
So I have multiple profiles here, so let me start.

03:13:17.300 --> 03:13:18.720  
Durga, Amar  
My entities application.

03:13:43.570 --> 03:14:00.330  
Durga, Amar  
This time I want to show you about the log. So instead of debugging the code in the configuration file so if you see the log file based on the same similar to basic configuration will be called here or open ended configuration file will be called because of my in my.

03:14:00.880 --> 03:14:09.140  
Durga, Amar  
Uh, Jason file I gave is it open IDC where as a as a variable I gave my other indication mode is open ID connect.

03:14:10.180 --> 03:14:12.980  
Durga, Amar  
So it'll go to respect to have configuration file.

03:14:55.690 --> 03:15:01.130  
Durga, Amar  
Yeah, this is where we provide it. It got connected with.

03:15:02.170 --> 03:15:12.100  
Durga, Amar  
Well, open it. Then it fetches the keys and stores, so that's where it is successful. We shall see the message here. Security authorization using both two with Open ID Connect.

03:15:13.000 --> 03:15:22.550  
Durga, Amar  
You see this message. This is successful in terms of the configuration and your application is ready and my Tomcat embedded Tomcat is started with eight grades report.

03:15:34.260 --> 03:15:53.350  
Durga, Amar  
So even I can run to postman, but I'm just I'm going with the swagger UI for this. And now if you see authorization button here earlier, if you see based on your basic auth you see the client ID and password there. But now I change the configuration mode to open ID.

03:15:54.360 --> 03:16:04.090  
Durga, Amar  
So, my, uh, swagger UI shows authorization but one with CAPI security. With this bearer token. So whatever the token it is generated.

03:16:05.440 --> 03:16:06.430  
Durga, Amar  
It's just, uh.

03:16:07.190 --> 03:16:12.850  
Durga, Amar  
Faucet and authorize as soon as you feed. Click authorize. That's where it will validate it.

03:16:14.560 --> 03:16:23.520  
Durga, Amar  
OK, so as soon as it is validated so it will be showing your I would messages successful message which you can close.

03:16:24.870 --> 03:16:34.240  
Durga, Amar  
And once this validation is already once, he's a validation is taken place. Then you can call any service. For example, let's take one simple gap.

03:16:35.980 --> 03:16:37.470  
Durga, Amar  
So I can use the.

03:16:38.710 --> 03:16:47.440  
Durga, Amar  
Any person client our client Eddie offer my person I can give, but let's give the same person because this person is already configured in the database as well.

03:16:50.250 --> 03:16:55.840  
Durga, Amar  
And if you execute it so your API call already validated on dot com.

03:16:56.850 --> 03:16:59.820  
Durga, Amar  
JW token, it'll process further and.

03:17:01.300 --> 03:17:08.000  
Durga, Amar  
It'll give the response, but if you see the Mike kernel here, so the authorization is a better and this is where complete.

03:17:09.380 --> 03:17:11.070  
Durga, Amar  
Token is a appended.

03:17:14.440 --> 03:17:16.330  
Durga, Amar  
And this is my the rest endpoint.

03:17:16.870 --> 03:17:23.790  
Arputharajan, Beniel  
That means it's perishing their door. For this configure there they will be configuring it somewhere.

03:17:16.990 --> 03:17:17.790  
Durga, Amar  
And this is my.

03:17:24.760 --> 03:17:26.910  
Arputharajan, Beniel  
For the token expired.

03:17:25.090 --> 03:17:25.480  
Durga, Amar  
Yeah.

03:17:26.470 --> 03:17:32.790  
Durga, Amar  
Yeah, if the token expiry as I said to you. So in IDP application. So this is not.

03:17:33.240 --> 03:17:39.680  
Durga, Amar  
Uh, uh, in this application. So in IDP application you need to define so how your token should be there?

03:17:40.690 --> 03:17:41.100  
Arputharajan, Beniel  
OK.

03:17:40.820 --> 03:17:52.880  
Durga, Amar  
All your token should be generated so there we can, uh, this way past week time. Actually in our application in our IDP what we configured is so it's for Long live tokens. We kept it.

03:17:54.020 --> 03:18:04.560  
Durga, Amar  
Based on that, you are, uh, if you configure that. If you have a Pacific Time, as we said earlier, right? So at the time of service startup, it will get the keys.

03:18:06.270 --> 03:18:26.570  
Durga, Amar  
OK, so that keys will tell that. OK, this token is valid for this point of time that will spring will take care spring well understand based on the key and based on whatever the ash ash code is there in the doctor's signature right? So that will check it over the token is valid for that within that period of time or not.

03:18:28.270 --> 03:18:29.300  
Arputharajan, Beniel  
OK, so.

03:18:28.670 --> 03:18:31.920  
Durga, Amar  
Based on that, it'll initially it'll other indicated.

03:18:33.200 --> 03:18:39.290  
Arputharajan, Beniel  
So IDP we want to we want to have some access to it. To make this all these configuration.

03:18:39.820 --> 03:18:52.440  
Durga, Amar  
Yeah, so for that. Yes. For IDP you need to contact a few. There is internal to FS application but you need to contact to the IDP and ask for the permissions.

03:18:53.300 --> 03:18:53.890  
Arputharajan, Beniel  
OK.

03:18:53.690 --> 03:19:07.590  
Ayyanan, Ramya  
Yeah, for IDP, no. First of all you need to request for the firm creation. They will be creating the separate form for you and you know with that you know you can do the all the setup configurations and the keystore.

03:19:07.650 --> 03:19:21.410  
Ayyanan, Ramya  
Yeah, and I mean storage and other service provider related configuration. Everything you need to do that and based on that the changes needs to be done on service provider site. So from service provider.

03:19:21.770 --> 03:19:32.680  
Ayyanan, Ramya  
Uh, I mean that the farming URL and the service provider ID everything you need to configure as a matter data from that you know it will be accessing and it will be communicating.

03:19:33.550 --> 03:19:45.450  
Durga, Amar  
Correct, so ultimately IDP will give you that based on your product they will give one you show your URL. So based on your product they'll give an endpoint, so which you can use it for your application?

03:19:33.630 --> 03:19:33.940  
Arputharajan, Beniel  
No.

03:19:34.390 --> 03:19:35.800  
Arputharajan, Beniel  
So then we got this.

03:19:46.290 --> 03:19:53.960  
Arputharajan, Beniel  
OK, for hands on like is it possible for us to use this or any other like open source available?

03:19:54.130 --> 03:20:05.460  
Durga, Amar  
No, we can't share this information further and so on. But just if I'm giving as a as part of the flow demo we are giving it. But yeah, I checked with your team.

03:20:04.030 --> 03:20:04.460  
Arputharajan, Beniel  
OK.

03:20:06.400 --> 03:20:07.760  
Durga, Amar  
We can share these details.

03:20:06.510 --> 03:20:06.840  
Arputharajan, Beniel  
Yeah.

03:20:07.960 --> 03:20:15.270  
Arputharajan, Beniel  
Not this unlike any open, so it's like so that we can do some hands on for learning path.

03:20:16.580 --> 03:20:19.160  
Seniappan, Yuvarani  
You can break it clock for IDP integration.

03:20:20.830 --> 03:20:21.940  
Arputharajan, Beniel  
Oh, OK.

03:20:23.920 --> 03:20:24.540  
Ayyanan, Ramya  
Yeah, I ping.

03:20:24.700 --> 03:20:26.340  
Ayyanan, Ramya  
When you are in diet may help you.

03:20:26.990 --> 03:20:29.040  
Arputharajan, Beniel  
Yeah, thank you. Yeah sure.

03:20:30.630 --> 03:20:31.060  
Arputharajan, Beniel  
Thanks.

03:20:33.580 --> 03:20:45.080  
Gawale, Manik  
I'm I'm uh, like in the configure you use that certification right? JWT decode or something so it is to protect the JWT key, right?

03:20:46.220 --> 03:20:46.740  
Gawale, Manik  
Ah.

03:20:46.300 --> 03:20:48.820  
Durga, Amar  
Yeah, he, uh, he was saying about this one.

03:20:49.670 --> 03:20:58.810  
Gawale, Manik  
Yeah, there is one private method here this this validator is to protect the JWT token using digital certificates.

03:20:51.820 --> 03:20:52.180  
Durga, Amar  
Yes.

03:20:56.310 --> 03:20:56.710  
Durga, Amar  
Yes.

03:20:59.440 --> 03:21:04.220  
Durga, Amar  
Yeah, with this, uh, it will contact the issuer URA and it'll fix the keys.

03:21:06.730 --> 03:21:14.540  
Gawale, Manik  
Can we just the open open that CAPI? Security configuration? Extends parent class. I need to see what is there.

03:21:06.920 --> 03:21:07.550  
Durga, Amar  
The ones that.

03:21:15.800 --> 03:21:16.440  
Durga, Amar  
This one is working.

03:21:19.000 --> 03:21:23.690  
Durga, Amar  
Yeah, yeah, we have a different types of uh, authentication modes and.

03:21:26.900 --> 03:21:30.660  
Durga, Amar  
In the year we are, uh, just for firewalls on, yeah.

03:21:28.840 --> 03:21:29.510  
Gawale, Manik  
I just

03:21:34.150 --> 03:21:34.560  
Gawale, Manik  
OK.

03:21:35.460 --> 03:21:49.950  
Durga, Amar  
And this is actually here. We don't mention, but here we mentioned some of the URLs to be some of the filters we added actually, but this the actual file which now based on your uh, mode. This is the way the the configuration works.

03:21:53.050 --> 03:21:53.460  
Gawale, Manik  
Yep.

03:21:55.180 --> 03:22:03.800  
Gawale, Manik  
And suppose like I want to face the current logged in user in my rest call in my rest methods opposing the controller so.

03:22:05.250 --> 03:22:07.020  
Gawale, Manik  
All that it won't work.

03:22:08.550 --> 03:22:09.910  
Durga, Amar  
Yeah, yeah.

03:22:11.490 --> 03:22:12.750  
Durga, Amar  
We are using this.

03:22:13.920 --> 03:22:15.490  
Durga, Amar  
User ID right sorry.

03:22:16.140 --> 03:22:19.340  
Gawale, Manik  
No, not well, which is also slated with current JWT token.

03:22:20.300 --> 03:22:20.760  
Gawale, Manik  
Men.

03:22:20.650 --> 03:22:21.130  
Durga, Amar  
Yeah.

03:22:21.950 --> 03:22:23.090  
Gawale, Manik  
In the current situation.

03:22:24.880 --> 03:22:25.560  
Durga, Amar  
Can you repeat that?

03:22:24.900 --> 03:22:33.030  
Damle, Nikhil  
For the current request, right for the current recreational. So Spring Boot provides, you know a class called security context.

03:22:26.400 --> 03:22:28.120  
Gawale, Manik  
Yeah yeah yeah yeah.

03:22:33.800 --> 03:22:34.190  
Gawale, Manik  
Uh-huh

03:22:42.110 --> 03:22:42.580  
Gawale, Manik  
OK.

03:22:42.630 --> 03:22:52.990  
Damle, Nikhil  
Simple can be accessed, you know, at any point of time anywhere in your rest calls using the security context. So the security context is nothing but you know it's it's actually stored in the thread local.

03:22:53.620 --> 03:23:00.340  
Damle, Nikhil  
So if you live know about thread local right, the thread local is is a space, which is, you know, specific to this current thread.

03:23:01.080 --> 03:23:17.110  
Damle, Nikhil  
And the security context is stored on the thread local internally by spring boot, but it's available anywhere you know on on any of the controller service layer. So user security context to get the principle. The principle will tell you know the the username.

03:23:02.110 --> 03:23:02.380  
Gawale, Manik  
So.

03:23:19.140 --> 03:23:24.370  
Gawale, Manik  
OK, so entire data I will it will be available whatever they're in the row in the database.

03:23:26.050 --> 03:23:26.630  
Gawale, Manik  
Maybe?

03:23:26.360 --> 03:23:26.980  
Damle, Nikhil  
Ah.

03:23:27.200 --> 03:23:33.760  
Damle, Nikhil  
There is no database, right? So in this case you know the password. You don't have the password with your application, you know the password is stored on IDP.

03:23:35.020 --> 03:23:35.480  
Damle, Nikhil  
So the.

03:23:35.050 --> 03:23:36.640  
Durga, Amar  
Only these details will be there.

03:23:36.830 --> 03:23:37.260  
Damle, Nikhil  
Yeah.

03:23:36.920 --> 03:23:43.430  
Gawale, Manik  
Yeah, other other details, last name or whatever personal those all will be available in the principal yourself.

03:23:40.540 --> 03:23:41.170  
Damle, Nikhil  
Correct, correct?

03:23:43.760 --> 03:23:47.430  
Damle, Nikhil  
Principle, yeah, so it's like like part of the payload. The things in the payload.

03:23:46.460 --> 03:23:46.730  
Gawale, Manik  
No.

03:23:58.730 --> 03:24:01.390  
Durga, Amar  
Yeah, that's about a open ID demo.

03:24:04.670 --> 03:24:05.530  
Durga, Amar  
Any questions?

03:24:12.630 --> 03:24:24.430  
Ayyanan, Ramya  
Uh, actually I have one questions regarding that repo. Artifactory artifacts, you know we are referring way those are against with the Veracode and blackduck issues.

03:24:26.390 --> 03:24:27.420  
Durga, Amar  
Already we're talking.

03:24:29.400 --> 03:24:35.760  
Ayyanan, Ramya  
No, actually not for yeah independency we are. We are using some repos right? I mean artifacts.

03:24:29.460 --> 03:24:29.840  
Durga, Amar  
Yeah.

03:24:37.440 --> 03:24:38.870  
Durga, Amar  
OKA

03:24:37.990 --> 03:24:41.240  
Ayyanan, Ramya  
Yeah, yeah so yeah.

03:24:40.710 --> 03:24:46.240  
Durga, Amar  
this is a actually, uh, this artifact based on your application, we had, yes.

03:24:46.290 --> 03:24:48.980  
Durga, Amar  
Uh, uh, we are using that.

03:24:51.230 --> 03:24:54.980  
Durga, Amar  
This is our project and that artifact ID we use in.

03:24:51.580 --> 03:24:52.190  
Ayyanan, Ramya  
Yeah, that.

03:24:56.550 --> 03:24:59.220  
Durga, Amar  
I forgot what? Yeah yeah, play it's a.

03:24:58.190 --> 03:24:58.790  
Damle, Nikhil  
Bitbucket.

03:25:00.980 --> 03:25:04.780  
Durga, Amar  
Then we configure a Docker images on. It will be stored right?

03:25:06.160 --> 03:25:08.160  
Ayyanan, Ramya  
Yeah, I, I'm the dark dark.

03:25:06.310 --> 03:25:07.740  
Patro, Patanuru Santosh K  
I'm not. I think that code.

03:25:10.570 --> 03:25:20.840  
Patro, Patanuru Santosh K  
I think the question is whatever the third party library that we are using, whether it has been there, any kind of pay vulnerability reported in the blood talk and the barcode scan.

03:25:10.740 --> 03:25:11.310  
Durga, Amar  
What is it?

03:25:14.130 --> 03:25:14.550  
Ayyanan, Ramya  
Yeah.

03:25:21.510 --> 03:25:39.680  
Durga, Amar  
Yeah, we follow both bag the converter code Veracode hand at this point of time we always we are other indicate before every month we release the product before we really see, we make sure that whatever the dependencies are we what we added to the project confirmed by Veracode and Black Duck.

03:25:40.270 --> 03:25:40.680  
Patro, Patanuru Santosh K  
OK.

03:25:40.370 --> 03:25:55.410  
Durga, Amar  
Yeah, we we get for your alert from black duck. Based on that what we do is we do up gradation in the next Sprint on the current Sprint. If it is a very high alert or high then we try to fix it. If it is a medium then we'll try to.

03:25:55.550 --> 03:26:08.510  
Durga, Amar  
I'm proceed further release again in the medium also if it is fixed required by the policy, we make sure that we've tried to fix it, so that's where our application keep on upgrading all the dependencies.

03:26:09.520 --> 03:26:10.520  
Durga, Amar  
Every time to time.

03:26:11.570 --> 03:26:42.020  
Damle, Nikhil  
As long as I think if you are, you know mostly on the current versions of spring boot you you won't, you know land into any issues with Veracode or blankets. Nothing left to stay current because you know even on even in spring boot you know if you're not current at at some point you know blackduck, flags, flags, some of those dependencies that you are having some challenges. So I think we then upgrade our. You know all those dependencies to latest versions and then.

03:26:30.790 --> 03:26:31.370  
Durga, Amar  
Yeah, Becky.

03:26:42.620 --> 03:26:44.850  
Damle, Nikhil  
Black Duck is mostly fine with all that.

03:26:46.000 --> 03:26:50.310  
Ayyanan, Ramya  
OK, this is being taken from our FIS artifactory or from Maven Central.

03:26:50.840 --> 03:26:52.040  
Durga, Amar  
Yes, if a aside.

03:26:51.580 --> 03:26:51.930  
Ayyanan, Ramya  
Right?

03:26:54.350 --> 03:26:55.490  
Ayyanan, Ramya  
Can you show me that, uh?

03:26:54.600 --> 03:26:55.240  
Durga, Amar  
So you know.

03:26:54.770 --> 03:26:55.030  
Patro, Patanuru Santosh K  
And.

03:26:56.190 --> 03:26:57.750  
Ayyanan, Ramya  
Not effect yarn please.

03:26:58.830 --> 03:27:02.280  
Durga, Amar  
Uh, I don't know whether I can share show or not, but.

03:27:02.150 --> 03:27:03.630  
Damle, Nikhil  
Which one OK?

03:27:03.490 --> 03:27:05.050  
Durga, Amar  
Dot M2 file there.

03:27:04.920 --> 03:27:07.780  
Ayyanan, Ramya  
Now artifact URL for the spring boot.

03:27:09.230 --> 03:27:09.610  
Ayyanan, Ramya  
Yeah.

03:27:09.480 --> 03:27:13.590  
Damle, Nikhil  
Or spring boot fear taking directly from Maven Central, actually not.

03:27:13.210 --> 03:27:14.850  
Ayyanan, Ramya  
OK, yeah, that's what I'm asking.

03:27:14.740 --> 03:27:15.470  
Damle, Nikhil  
Yeah yeah, yeah.

03:27:16.250 --> 03:27:20.010  
Durga, Amar  
Yeah, internally it uses both I face outfit.

03:27:20.570 --> 03:27:49.620  
Ayyanan, Ramya  
Yeah, yeah, you know our application images will be posted to our FIS artifactory, but I'm asking about that Spring Framework and spring boot artifact. He repos so you know basically some of the images will be loaded in our artifactory from the where I can, you know, we can refer it, but some of them needs to be taken from Nelson truth, so that's what I'm asking whether I can refer from men central or if the if it is an HR. If artifactory I can take it from there.

03:27:28.630 --> 03:27:29.240  
Durga, Amar  
Oh, OK.

03:27:49.680 --> 03:27:52.400  
Ayyanan, Ramya  
That's why I was asking, thank you, thank you, thank you.

03:27:51.010 --> 03:27:51.360  
Durga, Amar  
Well.

03:27:53.800 --> 03:28:02.210  
Durga, Amar  
But uh, both the details I mentioned in one of the same profile. So in that I have some credentials there, so that's where I I I thought of not to share that.

03:28:02.790 --> 03:28:03.880  
Ayyanan, Ramya  
Yeah, thank you.

03:28:07.220 --> 03:28:16.000  
Patro, Patanuru Santosh K  
I I'm not. I wanted to know about the how we are just implementing the user permissioning in this particular application.

03:28:16.400 --> 03:28:33.770  
Patro, Patanuru Santosh K  
Majors authentication part I understood with that different protocol you can able to implement. But let's say if you're reading the JWT token and the Panda token payload, we can get the user information the username. So after that username how? If you have if you plan to implement.

03:28:34.840 --> 03:28:40.670  
Patro, Patanuru Santosh K  
Had different permission, however, spring provide anything out of the bus to implement any permissioning.

03:28:42.500 --> 03:28:43.120  
Patro, Patanuru Santosh K  
Hey there.

03:28:43.650 --> 03:28:49.500  
Damle, Nikhil  
So on the you know on the JWT token you can actually add something called as claims.

03:28:50.250 --> 03:28:54.950  
Damle, Nikhil  
So I mean, if you can, you know so our class that CAPI JWT validator.

03:28:55.680 --> 03:28:56.230  
Durga, Amar  
Yeah, this.

03:28:57.630 --> 03:29:02.020  
Damle, Nikhil  
So in the JW token we we haven't made a good use of that really here, but.

03:29:02.640 --> 03:29:08.580  
Damle, Nikhil  
You know it's possible to use those claims, uh, and then you know those claims will actually.

03:29:10.030 --> 03:29:17.170  
Damle, Nikhil  
So there JW token, can you know have different kind of claims? The claims can be used in as the authorization mechanism.

03:29:18.200 --> 03:29:21.290  
Damle, Nikhil  
To control access to a few things.

03:29:22.350 --> 03:29:25.780  
Damle, Nikhil  
We really haven't made a lot of use of these claims to yeah.

03:29:23.670 --> 03:29:24.150  
Patro, Patanuru Santosh K  
And this.

03:29:27.590 --> 03:29:42.410  
Patro, Patanuru Santosh K  
So you are saying, this payload can be configured the IDP label to configure the different claim for the particular user and based upon that we can able to retrieve that claim in the application and based upon the claim we can implement the authorization.

03:29:34.120 --> 03:29:34.600  
Damle, Nikhil  
Keep source.

03:29:43.180 --> 03:29:43.770  
Damle, Nikhil  
Good.

03:29:43.250 --> 03:29:43.850  
Durga, Amar  
Correct?

03:29:44.730 --> 03:29:52.420  
Damle, Nikhil  
So that user you know can be configured to have a few privileges in IDP and then those rules based on rules and then.

03:29:49.680 --> 03:29:50.080  
Durga, Amar  
So.

03:29:53.050 --> 03:29:56.650  
Damle, Nikhil  
Those can be rules or claims and then based on that you can control.

03:29:57.490 --> 03:30:00.610  
Damle, Nikhil  
Access from within this application.

03:30:02.900 --> 03:30:03.710  
Damle, Nikhil  
For that user.

03:30:06.640 --> 03:30:12.170  
Damle, Nikhil  
Frankly, we haven't made use good use of that, but there is this facility. Yes, yeah, sorry, go ahead.

03:30:06.680 --> 03:30:07.370  
Patro, Patanuru Santosh K  
And now.

03:30:12.830 --> 03:30:27.040  
Patro, Patanuru Santosh K  
Yeah, so another use catalysts means we have a 10 methods in one particular with Alice in the web service and we want to control the access of this particular 10 method depend upon the user.

03:30:28.440 --> 03:30:32.170  
Patro, Patanuru Santosh K  
How best we can think we can also try to implement that.

03:30:32.670 --> 03:30:36.990  
Durga, Amar  
So you're talking about the business part, or you're talking about, uh, dentition part.

03:30:38.160 --> 03:31:02.250  
Patro, Patanuru Santosh K  
I am talking about the authorize in parallel. The authentication part has been handled OK by the any portal Callebaut. After that there comes to the authorization part they want to manage. OK, let's say we obtain methods has been exposed out of 10 methods for the one with. I want to expose only five method. Otherwise I want to as per the other five method. If I want to control that.

03:31:02.530 --> 03:31:05.630  
Patro, Patanuru Santosh K  
How I can able to? What is the best way to do that?

03:31:08.530 --> 03:31:14.460  
Durga, Amar  
You're talking about the filters about the URLs means. Here methods means the different endpoints or.

03:31:15.520 --> 03:31:37.290  
Patro, Patanuru Santosh K  
It is the same and I don't know. I mean so the the Member in the same endpoint we will be having get let's say in one microservice the microservice having the less attend different control rules. OK for some so in there. So I want to based upon the user I want to access them to the specific controllers. Let's give that example.

03:31:24.530 --> 03:31:24.800  
Durga, Amar  
2.

03:31:28.700 --> 03:31:29.130  
Durga, Amar  
OK.

03:31:39.800 --> 03:31:45.350  
Durga, Amar  
Oh, based on the different roles of the user user you want to exude that methods.

03:31:43.530 --> 03:31:44.110  
Patro, Patanuru Santosh K  
Yes.

03:31:45.980 --> 03:31:46.500  
Patro, Patanuru Santosh K  
Yes.

03:31:51.060 --> 03:31:55.490  
Durga, Amar  
So Umm, and going into the business layer action so this based on the rules.

03:31:59.120 --> 03:32:02.520  
Gawale, Manik  
We can use the matchers right URL matches in the.

03:32:03.480 --> 03:32:07.920  
Durga, Amar  
Yeah, but it's point is in one endpoint. If there are 10 methods are there so.

03:32:07.780 --> 03:32:08.250  
Gawale, Manik  
OK.

03:32:08.030 --> 03:32:08.700  
Patro, Patanuru Santosh K  
Yes.

03:32:08.920 --> 03:32:15.300  
Durga, Amar  
Yeah, so it's point is not. I can if you have matches I can do in the configuration itself based on the role.

03:32:14.780 --> 03:32:17.620  
Gawale, Manik  
There must be. There must be some annotations.

03:32:19.070 --> 03:32:22.770  
Gawale, Manik  
Like I don't know method level annotations, I think.

03:32:21.980 --> 03:32:42.050  
Patro, Patanuru Santosh K  
Miss app technically means we know how to implement, but I just wanted to know whether Spring boot proper, any kind of pace, has flexibility where based upon the means we can give some access to the particular user based upon which particular methods, something, some configuration someplace and any idea.

03:32:45.860 --> 03:32:49.560  
Damle, Nikhil  
No, actually even I don't have that idea. It's a good question we can.

03:32:50.230 --> 03:32:53.930  
Damle, Nikhil  
Look up some information, but like what you're saying is, you know, based on.

03:32:52.060 --> 03:32:57.080  
Durga, Amar  
If it is classed level, at least so we can have some conditions partial method level.

03:32:58.950 --> 03:32:59.510  
Durga, Amar  
Now we need.

03:32:59.120 --> 03:33:12.840  
Damle, Nikhil  
I think if there any service level you know annotations and then maybe based on the security context. Spring Boot has some capabilities, right? I think that is the question that can some spring boot inherent capabilities we used to.

03:33:13.590 --> 03:33:19.240  
Damle, Nikhil  
Handle such accesses kind of so not sure. Actually, really you know, but maybe will try to look up.

03:33:17.850 --> 03:33:18.290  
Patro, Patanuru Santosh K  
OK.

03:33:20.550 --> 03:33:50.740  
Patro, Patanuru Santosh K  
So miss currently what we did that means whenever you are introducing any methods. OK, we are just web on database where we meant we just make a insert insert to this particular method and with this map that particular user for video to this particular method. That is how we are currently controlling. But I I'm thinking that's the kinda pay very old way we can come. If there there could be some interesting where that Springwood will provide then I think we can plan to use.

03:33:30.510 --> 03:33:31.110  
Damle, Nikhil  
Lester

03:33:33.600 --> 03:33:33.830  
Durga, Amar  
And.

03:33:44.060 --> 03:33:44.460  
Durga, Amar  
Yeah.

03:33:45.530 --> 03:33:45.890  
Durga, Amar  
I am.

03:33:50.800 --> 03:33:53.500  
Patro, Patanuru Santosh K  
So anyways, for him I have nothing is.

03:33:53.040 --> 03:33:53.640  
Damle, Nikhil  
yeah.

03:33:54.170 --> 03:33:54.670  
Patro, Patanuru Santosh K  
We can.

03:33:54.730 --> 03:33:55.110  
Patro, Patanuru Santosh K  
I'm looking.

03:33:57.590 --> 03:34:03.910  
Durga, Amar  
Yeah, we didn't think about it because we didn't have that use case. But yeah, let me let us think once again.

03:33:58.010 --> 03:33:58.380  
Damle, Nikhil  
Yeah.

03:34:04.580 --> 03:34:07.420  
Durga, Amar  
And we feel any idea then can share tomorrow.

03:34:07.960 --> 03:34:09.380  
Patro, Patanuru Santosh K  
Serums thank you.

03:34:17.850 --> 03:34:18.900  
Durga, Amar  
Any questions for that?

03:34:29.010 --> 03:34:31.580  
Durga, Amar  
So let me stop sharing my screen.

03:34:39.310 --> 03:34:50.830  
Durga, Amar  
Yeah, that's all about today's session about spring security. If you have any questions, you can just post in the chat or also we can discuss in tomorrow's session as well.

03:34:53.290 --> 03:34:54.500  
Ayyanan, Ramya  
I have only one question.

03:34:55.490 --> 03:34:56.070  
Durga, Amar  
Yeah, go ahead.

03:34:56.220 --> 03:34:56.540  
Ayyanan, Ramya  
Yeah.

03:34:58.290 --> 03:35:15.740  
Ayyanan, Ramya  
So while I know when you I know why logging in your application IDP will generate the token and it will share with you, so while you log out from your application, will you send for the any API to kill the? I mean this try the token to IDP.

03:35:15.260 --> 03:35:15.980  
Durga, Amar  
No, uh.

03:35:18.780 --> 03:35:35.970  
Durga, Amar  
No, it's not a killer token for the IDP because still we said that the token is valid as long as we Longley. So if for every request header, if the token is, if it is, token is not there, it will not authenticate it. If we token is present.

03:35:37.220 --> 03:35:40.850  
Durga, Amar  
Uh, it will, uh, it will be based on the keys it'll validate and it'll go further.

03:35:42.210 --> 03:35:51.620  
Durga, Amar  
And every requested the stateless so we don't store anything. So along the request is done, that request is completely asked last whatever it is there in the header.

03:35:53.440 --> 03:35:55.020  
Durga, Amar  
Even the stateless request right?

03:35:55.910 --> 03:35:57.240  
Ayyanan, Ramya  
Yeah, OK.

03:35:57.090 --> 03:36:01.900  
Damle, Nikhil  
It's good just so good question though, you know maybe we'll have to see if IDP.

03:35:57.340 --> 03:35:58.730  
Durga, Amar  
So no need to kill anything.

03:36:01.060 --> 03:36:03.820  
Seniappan, Yuvarani  
Yeah, I I I have one more question adding to.

03:36:03.870 --> 03:36:05.220  
Seniappan, Yuvarani  
You order me like OK?

03:36:04.600 --> 03:36:15.000  
Durga, Amar  
But Nikhil, uh, so even though, uh, if it says if you talk about Ramya's question Nikhil, it is a stateless request, right? So why we need to write nearly killed?

03:36:13.080 --> 03:36:13.810  
Damle, Nikhil  
It is, yeah.

03:36:15.530 --> 03:36:16.520  
Ayyanan, Ramya  
So when I.

03:36:15.720 --> 03:36:18.270  
Damle, Nikhil  
No, not to my. Yeah, sorry go ahead.

03:36:18.800 --> 03:36:19.320  
Durga, Amar  
Can't wait.

03:36:19.660 --> 03:36:21.030  
Ayyanan, Ramya  
No problem, good naked.

03:36:22.400 --> 03:36:48.070  
Damle, Nikhil  
No, I'm I'm just trying to, you know, say that water. And yeah, I think is asking is like you know, once you are from the clients perspective, right? You know once you are, maybe there's a UI application and the user is logged out, you know so you don't have any use of that JWT token. Further, even the fits. So not from a microservices application perspective, but from a consumer perspective. If you're holding a JW token and you no longer planning to, you know call that.

03:36:42.690 --> 03:36:42.980  
Durga, Amar  
OK.

03:36:48.720 --> 03:36:53.580  
Damle, Nikhil  
Microservices endpoints. So maybe but depends really rubbed me. I think if IDP.

03:36:51.190 --> 03:36:56.470  
Durga, Amar  
That you rate standpoint, I can think about it, but not a foot micro service standpoint.

03:36:51.380 --> 03:36:51.810  
Ayyanan, Ramya  
Yeah.

03:36:55.340 --> 03:36:58.520  
Damle, Nikhil  
If ID provides any endpoint to kill the token, maybe.

03:36:59.260 --> 03:37:12.270  
Gawale, Manik  
I think it provides actually there are two modes. If you log out from USP service provider then either you can call the IDP and kill the session and you will get logged out from IDP as well.

03:37:13.190 --> 03:37:13.970  
Gawale, Manik  
But if it.

03:37:13.250 --> 03:37:21.380  
Ayyanan, Ramya  
Yeah, I know the API endpoint for the delete session, but you know I was asking about this at talker the Jot token.

03:37:20.950 --> 03:37:22.590  
Gawale, Manik  
Still specific to this year.

03:37:23.180 --> 03:37:23.640  
Ayyanan, Ramya  
Yeah.

03:37:24.360 --> 03:37:24.680  
Durga, Amar  
OK.

03:37:26.480 --> 03:37:39.650  
Ayyanan, Ramya  
Because actually, you know, I'm in the I'm in middle of the IDP SSO integration on my project, so I know I'm I'm just thinking that whether I can use or spring, SAML or spring or open night stream would open ID.

03:37:40.300 --> 03:37:43.540  
Ayyanan, Ramya  
So that's why I didn't understand getting the information.

03:37:45.120 --> 03:37:47.860  
Ayyanan, Ramya  
So that I can use spring, SAML or spring opening.

03:37:50.580 --> 03:37:54.820  
Durga, Amar  
Yeah, we didn't explore this. One lessons take this question again.

03:37:53.220 --> 03:37:53.440  
Ayyanan, Ramya  
K.

03:37:55.760 --> 03:37:56.260  
Ayyanan, Ramya  
OK.

03:37:55.800 --> 03:37:59.870  
Durga, Amar  
We will discuss and also if we have any will check with the Open ID.

03:37:59.920 --> 03:38:00.220  
Durga, Amar  
Yeah.

03:38:01.270 --> 03:38:03.740  
Durga, Amar  
Our documents, as will envy, come back to you.

03:38:04.740 --> 03:38:06.030  
Ayyanan, Ramya  
Sure, thank you, thanks.

03:38:04.850 --> 03:38:06.650  
Durga, Amar  
Can you post your question in the chat?

03:38:07.340 --> 03:38:11.320  
Ayyanan, Ramya  
Yeah, yeah, sure I will do it. Yeah will do. Thank you, thank you, thank you man.

03:38:07.550 --> 03:38:07.830  
Durga, Amar  
Alright.

03:38:11.010 --> 03:38:11.310  
Durga, Amar  
Yeah.

03:38:16.290 --> 03:38:19.100  
Damle, Nikhil  
Sorry, I think there's another question ready you've Rani.

03:38:20.740 --> 03:38:23.930  
Seniappan, Yuvarani  
So yes, but I think I got the answer from Manik.

03:38:23.940 --> 03:38:24.830  
Damle, Nikhil  
OK, OK.

03:38:27.240 --> 03:38:28.190  
Durga, Amar  
Yeah, good question.

03:38:40.190 --> 03:39:10.100  
Durga, Amar  
OK, if there is no more questions, a lesson will share this PPT OK and also this recording. Those are available with you all and you can go through it and tomorrow and day after. Tomorrow is a big day for you guys so it's all. Or you can try and you can ask any questions. So tomorrow what we'll do is we'll give on the sample application advanced layer in terms of the controller service or end repository and how to call 2 Pacific battles and how to fix the respond back that is complete flow will give a demo.

03:39:10.480 --> 03:39:13.820  
Durga, Amar  
And then we will ask you to start practicing it.

03:39:14.920 --> 03:39:15.370  
Durga, Amar  
OK.

03:39:17.710 --> 03:39:18.880  
Ayyanan, Ramya  
Shared painting.

03:39:20.610 --> 03:39:25.340  
Patro, Patanuru Santosh K  
I'm not. I think there yesterday PPT has not shared in a said that today.

03:39:25.150 --> 03:39:25.400  
Durga, Amar  
Oh

03:39:25.460 --> 03:39:27.760  
Durga, Amar  
OK, should he will share both the PPD skin.

03:39:27.960 --> 03:39:28.330  
Patro, Patanuru Santosh K  
So.

03:39:32.290 --> 03:39:34.550  
Durga, Amar  
Thanks everyone, let's meet tomorrow.

03:39:35.790 --> 03:39:38.100  
Patil, Tatyasaheb  
Thank you, thank you all. We talk tomorrow.

03:39:36.100 --> 03:39:36.570  
Seniappan, Yuvarani  
Thank you.

03:39:37.180 --> 03:39:38.470  
Patro, Patanuru Santosh K  
Thank you everyone.

03:39:37.360 --> 03:39:42.080  
Ayyanan, Ramya  
Thanks bye thank you. Thank you thank you thank you, thank you.

03:39:37.440 --> 03:39:38.230  
Subramanian Kamatchi, Gobi Ganesh  
Thank you, thank you.

03:39:39.020 --> 03:39:39.420  
Mourya, Gourav  
Like

03:39:40.110 --> 03:39:40.630  
Ayyanar, Hema  
Thank you.

03:39:40.130 --> 03:39:41.800  
Thirugnanam, Kumaran  
thank you thanks bye.