**Security Mechanisms**

For the given assessment, I have added basic HTTP authentication.

We can use the following security mechanisms for authentication and authorisation: -

1. HTTPS by using SSL certificate

To force HTTPS in our Spring Boot app, we can extend WebSecurityConfigurerAdapter and require a secure connection.

SSL can be implemented by defining its properties in application.yml such as keystore, hostingServer and enabling the secure connection.

Http11NioProtocol proto = (Http11NioProtocol)connector.getProtocolHandler();

connector.setScheme(**"https"**);  
connector.setSecure(**true**);  
proto.setKeystoreFile(**sslProperties**.getKeyStoreLocation());  
proto.setKeystorePass(**sslProperties**.getKeyStorePassword());  
proto.setSslEnabledProtocols(**"TLSv1.2"**);

1. Customer SAML / System SAML – passing saml token in the header and forwarding it to the downstream systems using request header forwarder.

* A customer SAML is a token that is issued by the enterprise identity manager once a user has successfully passed log in from a digital channel using their username and password. The customer SAML contains important information such as role, loginId, password.
* System SAML is equivalent to a user but is provided as a SAML token that a recipient system can verify. As a SAML token it also usually contains the roles that the user has been granted.

1. For mobile security, HTTP basic authentication is not a good idea.

We store the API key securely in order for things to work. In addition to this, HTTP Basic Authentication requires that raw API keys be sent over the network for every request, thereby increasing the chance of risk in the long run.

Oauth2 - Provides token authentication.

A user enters username or email and password from mobile app.

POST request is sent from mobile app to API with username or email and password data included (OVER SSL).

API validates the user credentials, and create an access token for the user that expires after a certain amount of time.

This access token can be used on the mobile device, treating it like an API key.

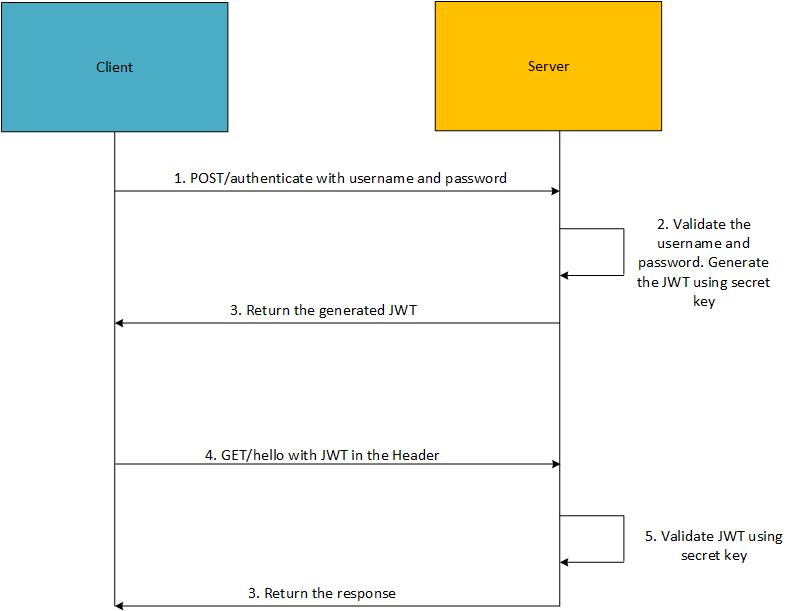
For using Oauth2, we add the following dependency in pom

spring-security-oauth2-autoconfigure and @EnableOAuth2Sso in the main application.

Also define certain oauth2 params in application.yml such as clientId, clientSecret, accessTokenUri.

Enforcing SSL - Plain HTTP is fine for testing but an Authorization Server should only be used over SSL in production.

1. We can also use JWT authentication for securing an exposed REST API



1. Using Encrypted keys and passwords to communicate between microservices.
2. Encoding sensitive data like customer personal details, account details. Encoding algorithms can be used to encode such data.
3. Role based and path-based mapping can also be configured in SecurityConfig and allowed roles can be set.

.antMatchers(**"/app/\*\*"**).permitAll()  
.antMatchers(**"/public/\*\*"**).permitAll()  
.antMatchers(**"/secure/\*\*"**).hasRole(**"USER"**)