Como configurar ticket de otobo:

```
private void llamarOtoboSSL()throws JsonProcessingException {
 // Crear las instancias de los objetos necesarios
 String userLogin = "myUser";
 String password = "passss";

 String title = "titulo del ticket"; // Campo vacío
 String queue = "Desarrollo de Sistemas";
 String customerUser = "myUser";
 String state = "new";
 String priorityID = "3";
 int typeID = 1;

 CreateTicketRequestDto.Ticket ticket = new CreateTicketRequestDto
         .Ticket(title, queue, customerUser, state, priorityID,
typeID);

 log.info("* Objeto CreateTicketRequestDto creado");

 String communicationChannel = "EmailWebservice Create Example";
 String from =  "test@test.de";
 String subject =  "Webservice Create Example";
 String body = "This was created by a Webservice request!";
 String contentType = "text/html charset=utf-8";

 CreateTicketRequestDto.Article article = new CreateTicketRequestDto
         .Article(communicationChannel, from, subject, body,
contentType);

 CreateTicketRequestDto request = new
CreateTicketRequestDto(userLogin, password, ticket, article);

 ticketsOtoboService.createNewTicketconSSL(request);
}



@Service
```

```java
@Slf4j
public class TicketsOtoboService {


   private RestTemplate createRestTemplateWithCustomSSL() throws
Exception {
       // Ruta del archivo PEM
      String pemFilePath = "C:\\desarrollando\\otobo llamar api\\ssl
cert\\server-cert-testing.pem";


       // Cargar el certificado desde el archivo PEM
       CertificateFactory factory =
CertificateFactory.getInstance("X.509");
       X509Certificate cert = (X509Certificate)
factory.generateCertificate(Files.newInputStream(new
File(pemFilePath).toPath()));

       // Crear un KeyStore temporal con el certificado cargado
       KeyStore trustStore =
KeyStore.getInstance(KeyStore.getDefaultType());
       trustStore.load(null, null);
       trustStore.setCertificateEntry("server", cert);

       // Construir el SSLContext con el KeyStore
       SSLContext sslContext = SSLContextBuilder.create()
               .loadTrustMaterial(trustStore, (chain, authType) ->
true)
               .build();

       // Crear un SSLConnectionSocketFactory con el SSLContext
       SSLConnectionSocketFactory socketFactory = new
SSLConnectionSocketFactory(
               sslContext, NoopHostnameVerifier.INSTANCE);

       // Crear el HttpClient con el SSLConnectionSocketFactory
       CloseableHttpClient httpClient = HttpClients.custom()

.setConnectionManager(PoolingHttpClientConnectionManagerBuilder.creat
e()
                       .setSSLSocketFactory(socketFactory)
                       .build())
               .build();

       // Crear un RestTemplate con el HttpClient configurado
```

```
        HttpComponentsClientHttpRequestFactory factory2 = new
HttpComponentsClientHttpRequestFactory(httpClient);
        return new RestTemplate(factory2);
    }

    public CreateTicketResponseDto
createNewTicketconSSL(CreateTicketRequestDto createTicketRequestDto)
{
        log.info("* Inicia metodo crearNuevoTicket.");
        log.info("* DTO : " + createTicketRequestDto.toString());

        try {
            String url2 =
"https://192.168.230.17/otobo/nph-genericinterface.pl/Webservice/ws1/
TicketCreate";
            log.info("Url crearNuevoTicket: " + url2);

            HttpHeaders headers = new HttpHeaders();
            headers.setContentType(MediaType.APPLICATION_JSON);
            headers.add("Accept", "*/*");

            HttpEntity<CreateTicketRequestDto> entity = new
HttpEntity<>(createTicketRequestDto, headers);

            log.info("Llamando al endpoint....");

            RestTemplate restTemplate1 =
createRestTemplateWithCustomSSL();

            ResponseEntity<String> response = restTemplate1.exchange(
                    url2, HttpMethod.POST, entity, String.class);

            CreateTicketResponseDto createTicketResponseDto = new
CreateTicketResponseDto();

            if (response.getStatusCode().is2xxSuccessful()) {
                String responseBody = response.getBody();
                log.info("* Respuesta de la API: " + responseBody);

                createTicketResponseDto.setRawResponse(responseBody);

                try {
                    ObjectMapper objectMapper = new ObjectMapper();
                    CreateTicketResponseDto parsedResponse =
objectMapper.readValue(responseBody, CreateTicketResponseDto.class);
```

```
createTicketResponseDto.setTicketID(parsedResponse.getTicketID());

createTicketResponseDto.setArticleID(parsedResponse.getArticleID());

createTicketResponseDto.setTicketNumber(parsedResponse.getTicketNumbe
r());

createTicketResponseDto.setError(parsedResponse.getError());
                } catch (JsonProcessingException e) {
                    log.error("No se pudo parsear la respuesta JSON,
pero se capturó la respuesta completa en rawResponse");
                }

                if (createTicketResponseDto.getError() != null) {
                    log.error("* Error detectado en la respuesta: " +
                            "ErrorMessage: " +
createTicketResponseDto.getError().getErrorMessage() +
                            ", ErrorCode: " +
createTicketResponseDto.getError().getErrorCode());
                }
                return createTicketResponseDto;
            } else {
                log.error("* Error: Respuesta no exitosa o cuerpo
nulo");
                createTicketResponseDto.setRawResponse("Error:
Respuesta no exitosa o cuerpo nulo");
                return createTicketResponseDto;
            }
        } catch (Exception e) {
            log.error("* Error durante crearNuevoTicket: " +
e.getMessage());
            CreateTicketResponseDto createTicketResponseDtoError = new
CreateTicketResponseDto();
            createTicketResponseDtoError.setRawResponse("Excepción: "
+ e.getMessage());
            return createTicketResponseDtoError;
        }
    }
```