ZLTECH

ZL UA API Comparison
Services API and REST API

Last Updated: February 6, 2024
Document Revision 1.0

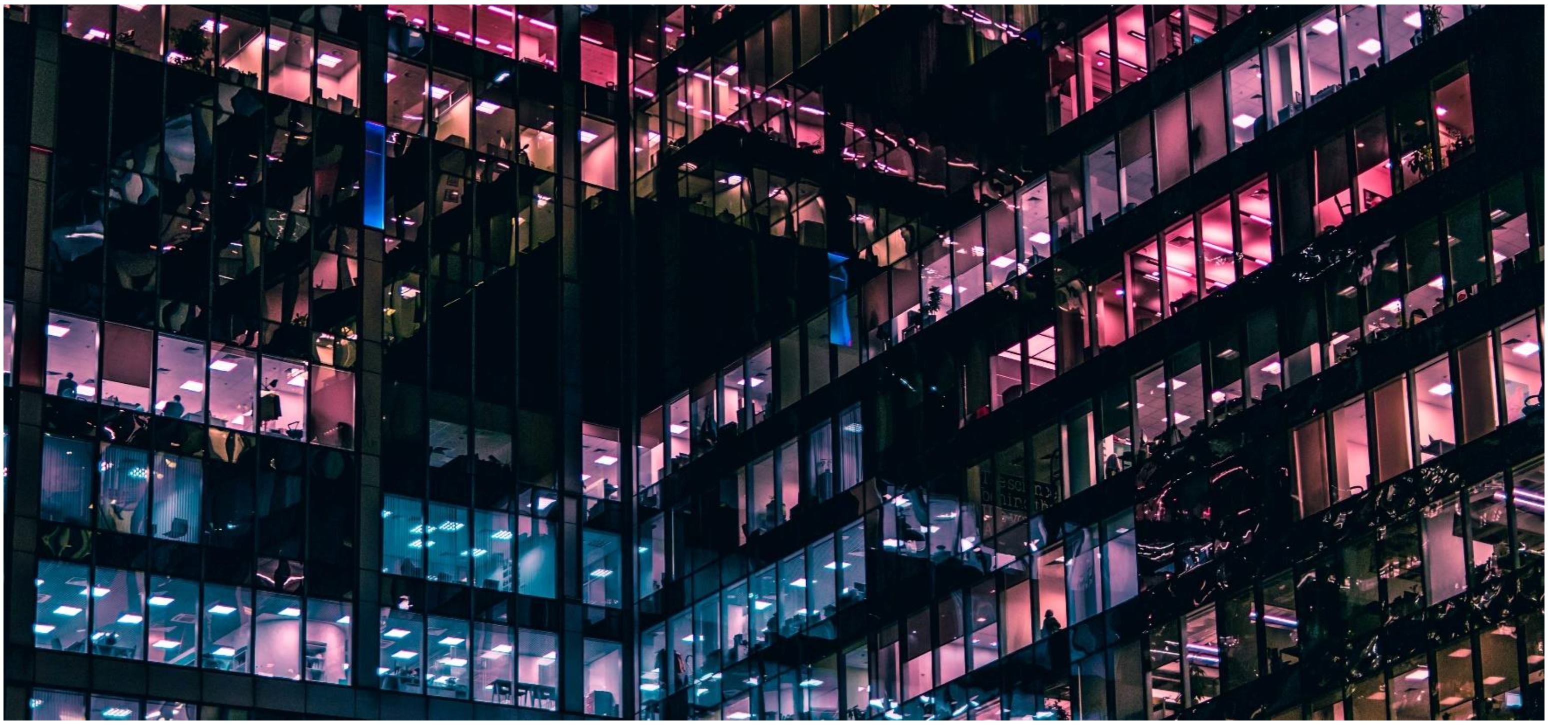


Table of Contents

Introduction

This document provides details on the ZL Unified Archive's (UA) API offerings: the **Services API** and the **REST API**. It includes the following sections:

- *ZL UA Services API*: This section provides background information on how the Services API works, including authorization details and example code.
- *REST API*: This section describes the advantages the REST API provides over the Services API. It also includes background information on how the REST API works, including authorization details and example code.

ZL UA Services API

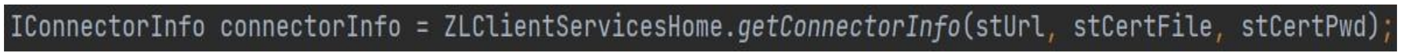
The ZL UA (Unified Archive) presently uses the Services API. The Services API works with Java and C# only. For executing services, you need **ZLServices** and **zlthin-common** two solutions/jars.

This section provides background details on the Services API.

Authorization

IConnectorInfo

IConnectorInfo is the basic object that is needed for all the Services API calls. This object provides network connectivity to the ZL server, as well as user authentication based on the certificate issued by the ZL server. An instance of this object can be created by using this line:



```
IConnectorInfo connectorInfo = ZLClientServicesHome.getConnectorInfo(stUrl, stCertFile, stCertPwd);
```

Figure 1: IConnectorInfo

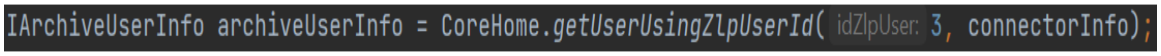
In this line:

- **stUrl** is the URL of the ZL server as explained above
- **stCertFile** is the ZL server issued certificate file
- **stCertPwd** is the password used to encrypt the certificate.

After making an instance of this **IConnectorInfo**, you can call the Services API by passing all the required parameters to the method along with the **IConnectorInfo** instance.

Services API Examples

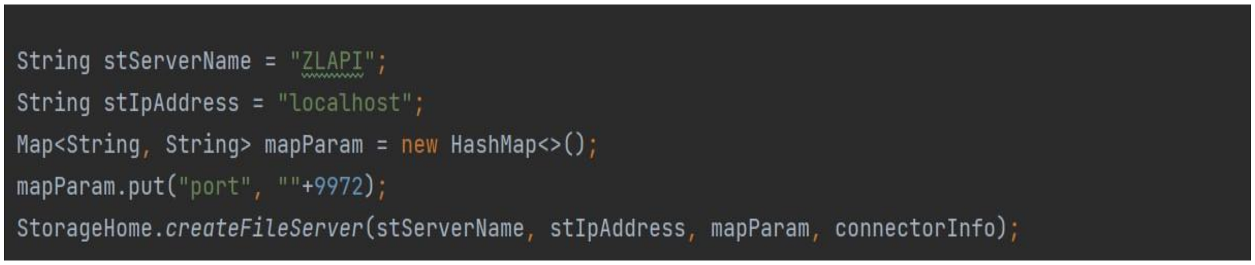
This example obtains user information using a user ID.



```
IArchiveUserInfo archiveUserInfo = CoreHome.getUserUsingZlpUserId( idZlpUser: 3, connectorInfo);
```

Figure 2: Getting User Information Using User ID

This example creates a file server.



```
String stServerName = "ZLAPI";
String stIpAddress = "localhost";
Map<String, String> mapParam = new HashMap<>();
mapParam.put("port", ""+9972);
StorageHome.createFileServer(stServerName, stIpAddress, mapParam, connectorInfo);
```

Figure 3: Creating a File Server

REST API

REST APIs provides a straightforward, scalable, and flexible way to enable communication between different components of a software system, making them a popular choice for building modern web services and applications.

Advantages of Using ZL UA's REST API over its Services APIs

In ZL UA, the REST API includes the following advantages over the Services API:

- The Services API uses **IConnectorInfo** for authentication. This is tightly connected to the ZL code base and is needed for all Services API calls. REST API uses **bearer tokens** for authentication**.** Once authenticated, you can make any REST API call using an HTTP request.
- REST APIs are simple and easy to understand. This simplicity contributes to better scalability and ease of maintenance. Standard HTTP methods like GET and POST make it easy for clients to understand and interact with APIs.
- REST API responses are generally provided in JSON format, which is easy to understand and integrate into applications. JSON supports most programming languages and facilitates straightforward parsing.
- Updates to the Services API are limited, but REST API can be easily upgraded.
- REST API supports writing the client in different languages, whereas the Services API only supports writing the client in Java and C#.

You can call ZL UA's REST API in several ways, and you can write your own client or use existing tools like Postman and Swagger. Examples are provided in the following sections.

Authorization in the REST API

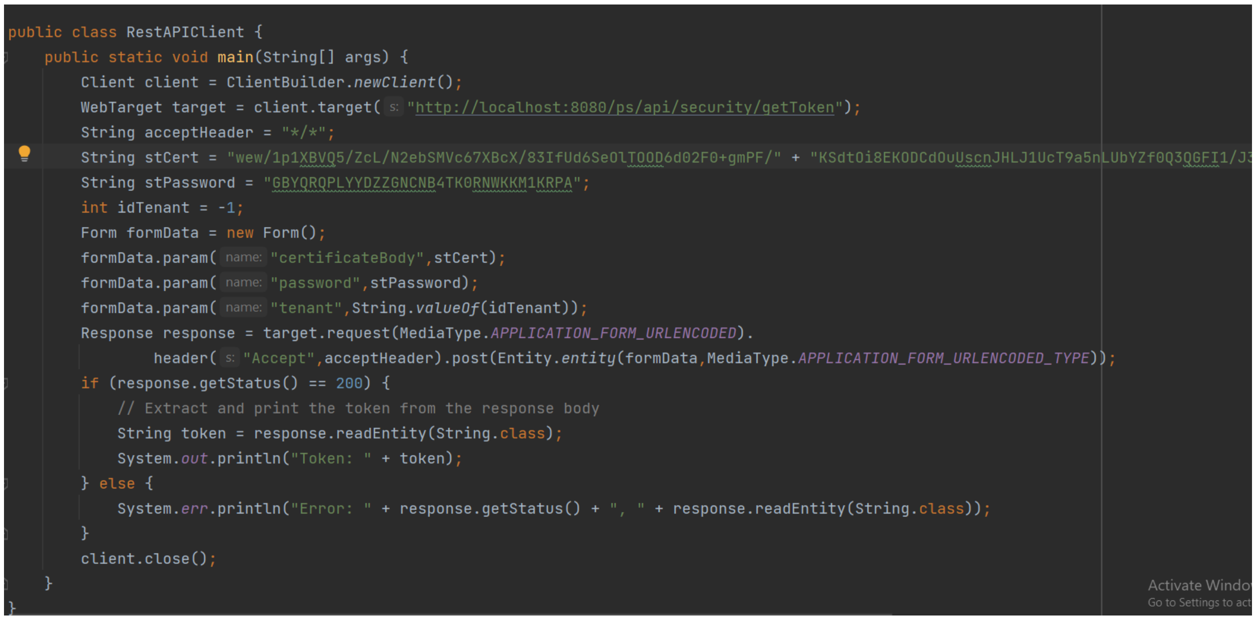
Before making any REST API call, you need to authenticate the application using ZL bearer tokens. You need to send the bearer token in the headers along with the requested URL to the REST API.

To get the ZL bearer token, you need to call the **getToken** API. You need to pass three parameters in this API.

1. **certificateBody**: ZL server issued certificate file body.
2. **Password**: the password used to encrypt the certificate.
3. **tenantId**: Tenant id of the ZL server.

A client code example for calling the **getToken** API is shown below. In this example:

- **stCertFile** is the ZL server issued certificate file.
- **stPassword** is the password used to encrypt the certificate.
- **idTenant** is the tenant id of the ZL server.



```java
public class RestAPIClient {
    public static void main(String[] args) {
        Client client = ClientBuilder.newClient();
        WebTarget target = client.target( "http://localhost:8080/ps/api/security/getToken");
        String acceptHeader = "*/*";
        String stCert = "wew/1p1XBVQ5/ZcL/N2ebSMVc67XBcX/83IfUd6SeOlTOOD6d02F0+gmPF/" + "KSdtOi8EKODCdOuUscnJHLJ1UcT9a5nLUbYZf0Q3QGFI1/J3
        String stPassword = "GBYQRQPLYYDZZGNCNB4TK0RNWKKM1KRPA";
        int idTenant = -1;
        Form formData = new Form();
        formData.param( "certificateBody",stCert);
        formData.param( "password",stPassword);
        formData.param( "tenant",String.valueOf(idTenant));
        Response response = target.request(MediaType.APPLICATION_FORM_URLENCODED).
                header( "Accept",acceptHeader).post(Entity.entity(formData,MediaType.APPLICATION_FORM_URLENCODED_TYPE));
        if (response.getStatus() == 200) {
            // Extract and print the token from the response body
            String token = response.readEntity(String.class);
            System.out.println("Token: " + token);
        } else {
            System.err.println("Error: " + response.getStatus() + ", " + response.readEntity(String.class));
        }
        client.close();
    }
}
```

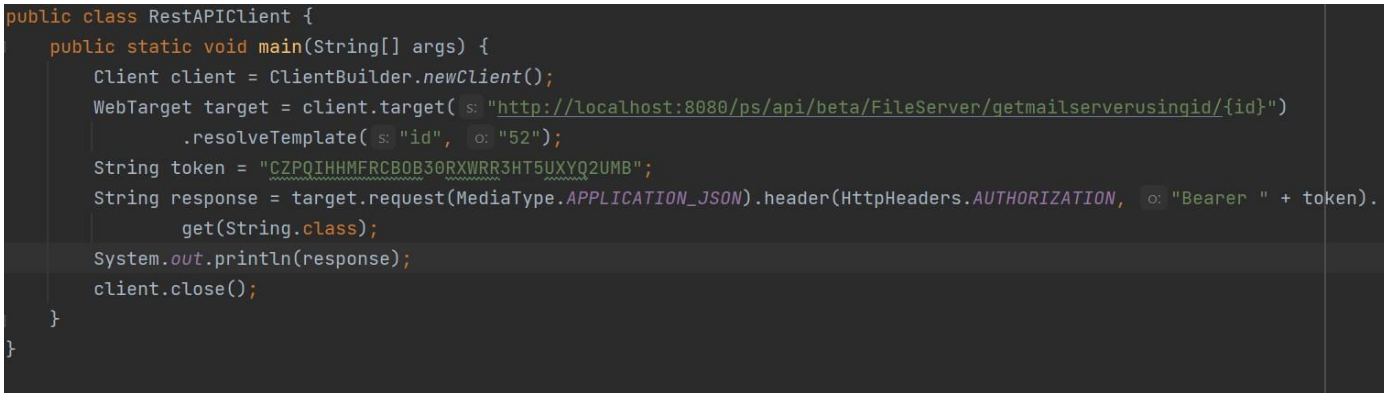
Figure 4: getTokenAPI

You will get the bearer token after calling this API, and you can send this token in headers to authenticate the REST API calls.

REST API Java Client Examples

Figures 5 and 6 show examples of REST API client implementations.

Example 1 - GET Operation: Getting File Server Information

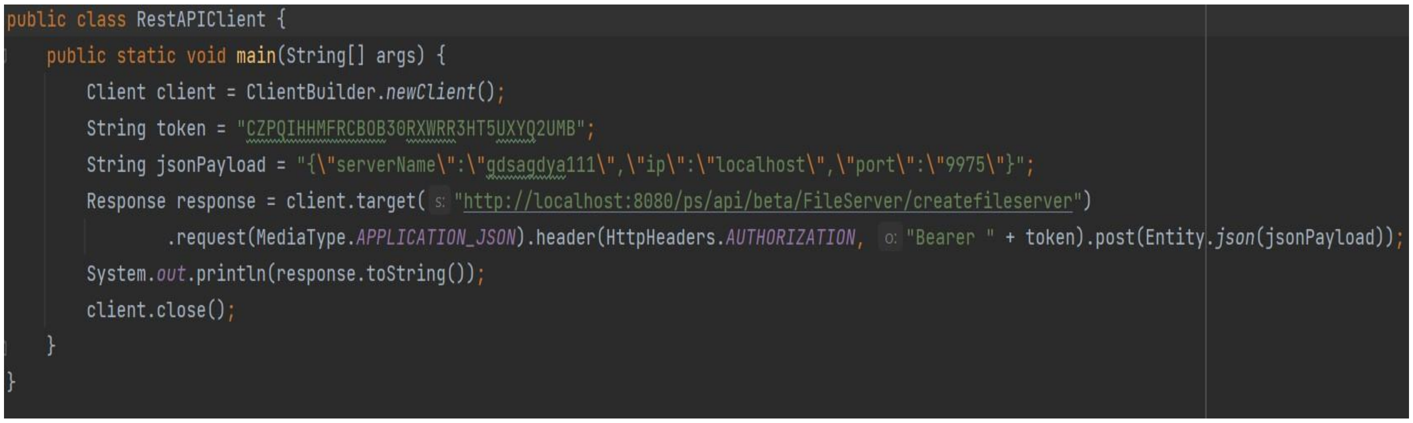


```java
public class RestAPIClient {
    public static void main(String[] args) {
        Client client = ClientBuilder.newClient();
        WebTarget target = client.target( s: "http://localhost:8080/ps/api/beta/FileServer/getmailserverusingid/{id}")
                .resolveTemplate( s: "id",  o: "52");
        String token = "CZPQIHHMFRCBOB30RXWRR3HT5UXYQ2UMB";
        String response = target.request(MediaType.APPLICATION_JSON).header(HttpHeaders.AUTHORIZATION,  o: "Bearer " + token).
                get(String.class);
        System.out.println(response);
        client.close();
    }
}
```

Figure 5: Getting File Server Information

Example 2 - Post Operation: Creating File Server

In this example, a client is created and a target URL is build. The request includes ZL bearer tokens in the headers which will hit the ZL API.

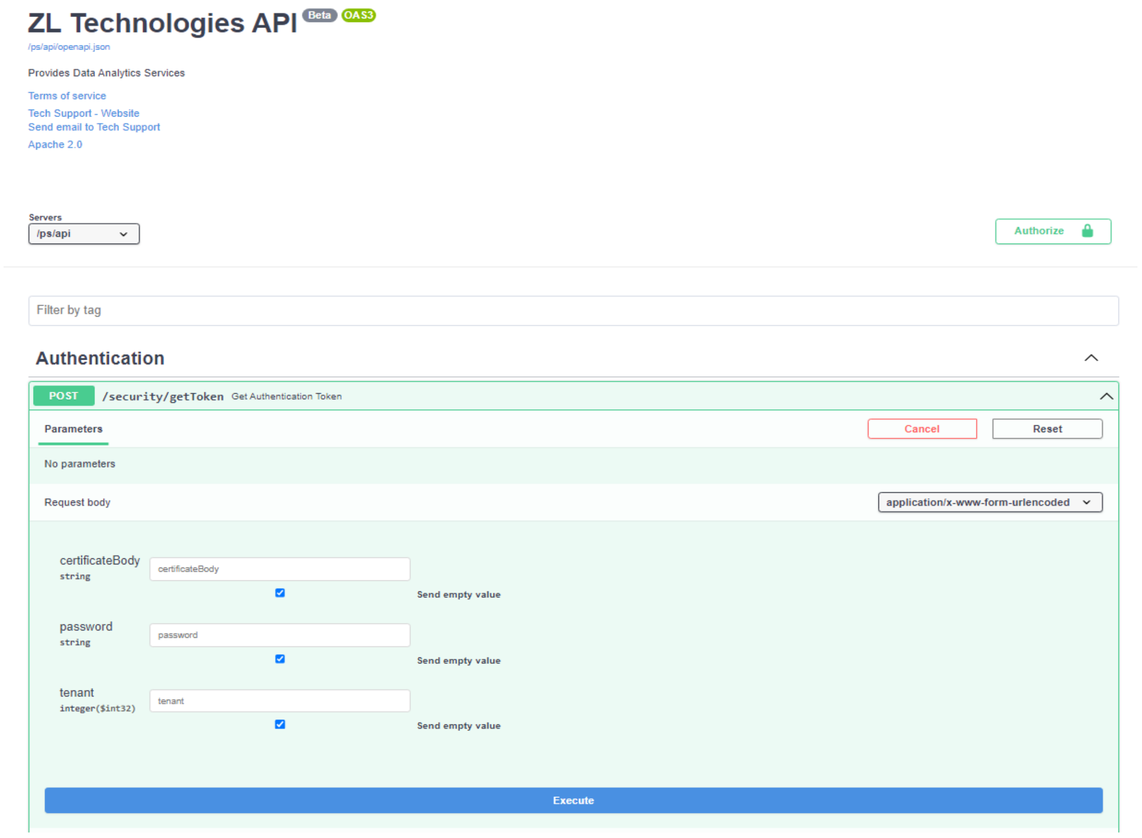


```java
public class RestAPIClient {
    public static void main(String[] args) {
        Client client = ClientBuilder.newClient();
        String token = "CZPQIHHMFRCBOB30RXWRR3HT5UXYQ2UMB";
        String jsonPayload = "{\"serverName\":\"gdsagdya111\",\"ip\":\"localhost\",\"port\":\"9975\"}";
        Response response = client.target( s: "http://localhost:8080/ps/api/beta/FileServer/createfileserver")
                .request(MediaType.APPLICATION_JSON).header(HttpHeaders.AUTHORIZATION,  o: "Bearer " + token).post(Entity.json(jsonPayload));
        System.out.println(response.toString());
        client.close();
    }
}
```

Figure 6: Creating File Server

REST API Using Swagger Client

This section shows examples of using the REST API with a Swagger client. Before using the REST API in Swagger, you must authenticate using the consumer's certificate, password and tenant identification. After authenticating, a token will be generated. You will need to authorize in swagger using that token. This is shown below.



ZL Technologies API Beta OAS3
/ps/api/openapi.json

Provides Data Analytics Services

Terms of service
Tech Support - Website
Send email to Tech Support
Apache 2.0

Servers: /ps/api

Authentication

POST /security/getToken Get Authentication Token

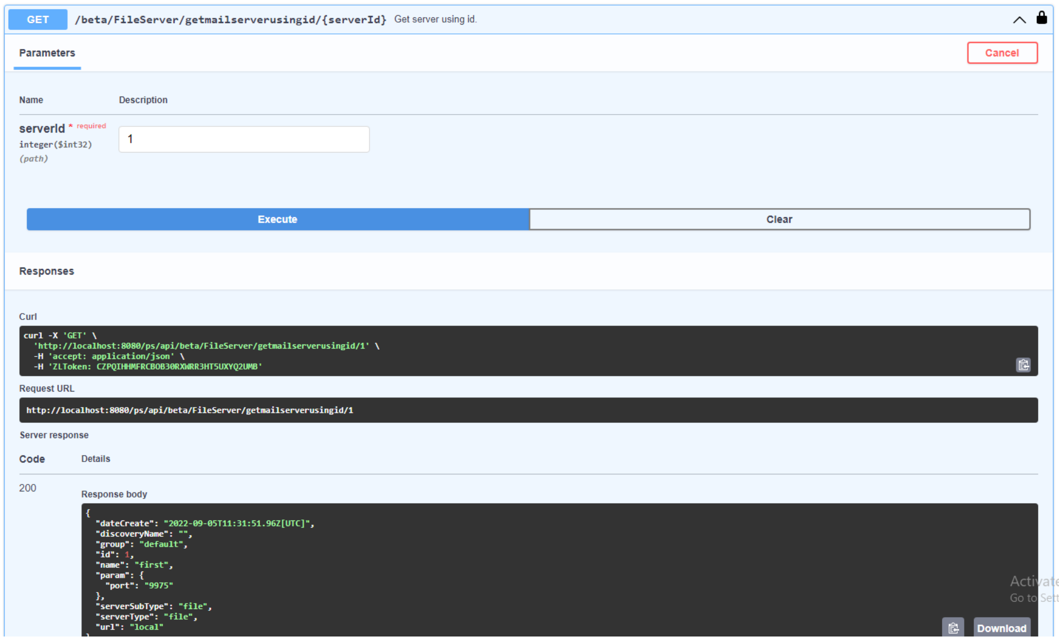
No parameters

Request body: application/x-www-form-urlencoded

certificateBody string — certificateBody — ✓ Send empty value
password string — password — ✓ Send empty value
tenant integer($int32) — tenant — ✓ Send empty value

Figure 7: Authentication

Example 1 - GET Operation: Getting File Server Information



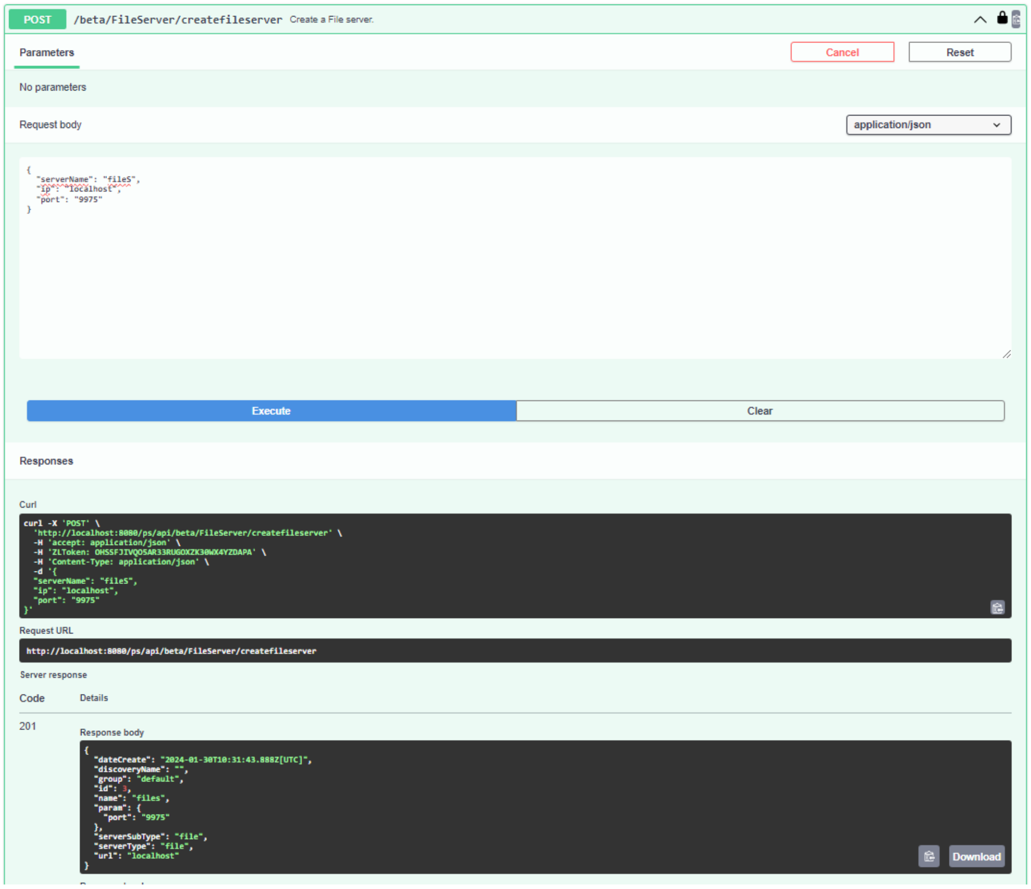
```
curl -X 'GET' \
  'http://localhost:8080/ps/api/beta/FileServer/getmailserverusingid/1' \
  -H 'accept: application/json' \
  -H 'ZLToken: CZPQIHHMFRCBOB30RXWRR3HT5UXYQ2UMB'
```

```
http://localhost:8080/ps/api/beta/FileServer/getmailserverusingid/1
```

```
{
  "dateCreate": "2022-09-05T11:31:51.96Z[UTC]",
  "discoveryName": "",
  "group": "default",
  "id": 1,
  "name": "first",
  "param": {
    "port": "9975"
  },
  "serverSubType": "file",
  "serverType": "file",
  "url": "local"
```

Figure 8: Getting File Sever Information

Example 2 – POST Operation: Creating File Server



```
{
  "serverName": "fileS",
  "ip": "localhost",
  "port": "9975"
}
```

```
curl -X 'POST' \
  'http://localhost:8080/ps/api/beta/FileServer/createfileserver' \
  -H 'accept: application/json' \
  -H 'ZLToken: OHSSFJIVQO5AR33RUGOXZK30WX4YZDAPA' \
  -H 'Content-Type: application/json' \
  -d '{
  "serverName": "fileS",
  "ip": "localhost",
  "port": "9975"
}'
```

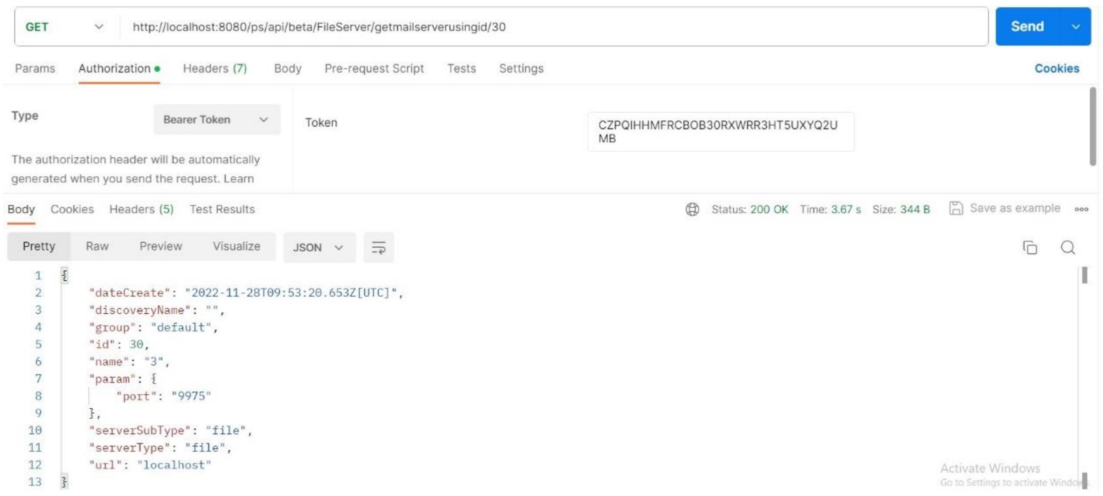
```
http://localhost:8080/ps/api/beta/FileServer/createfileserver
```

```
{
  "dateCreate": "2024-01-30T10:31:43.888Z[UTC]",
  "discoveryName": "",
  "group": "default",
  "id": 3,
  "name": "files",
  "param": {
    "port": "9975"
  },
  "serverSubType": "file",
  "serverType": "file",
  "url": "localhost"
}
```

Figure 9: Creating File Server

REST API Using Postman

REST API calls are authorized using ZL bearer tokens which are sent in the bearer token field in the postman along with the HTTP request. This is shown below.



GET http://localhost:8080/ps/api/beta/FileServer/getmailserverusingid/30 Send

Params | Authorization | Headers (7) | Body | Pre-request Script | Tests | Settings | Cookies

Type: Bearer Token

Token: CZPQIHHMFRCBOB30RXWRR3HT5UXYQ2U MB

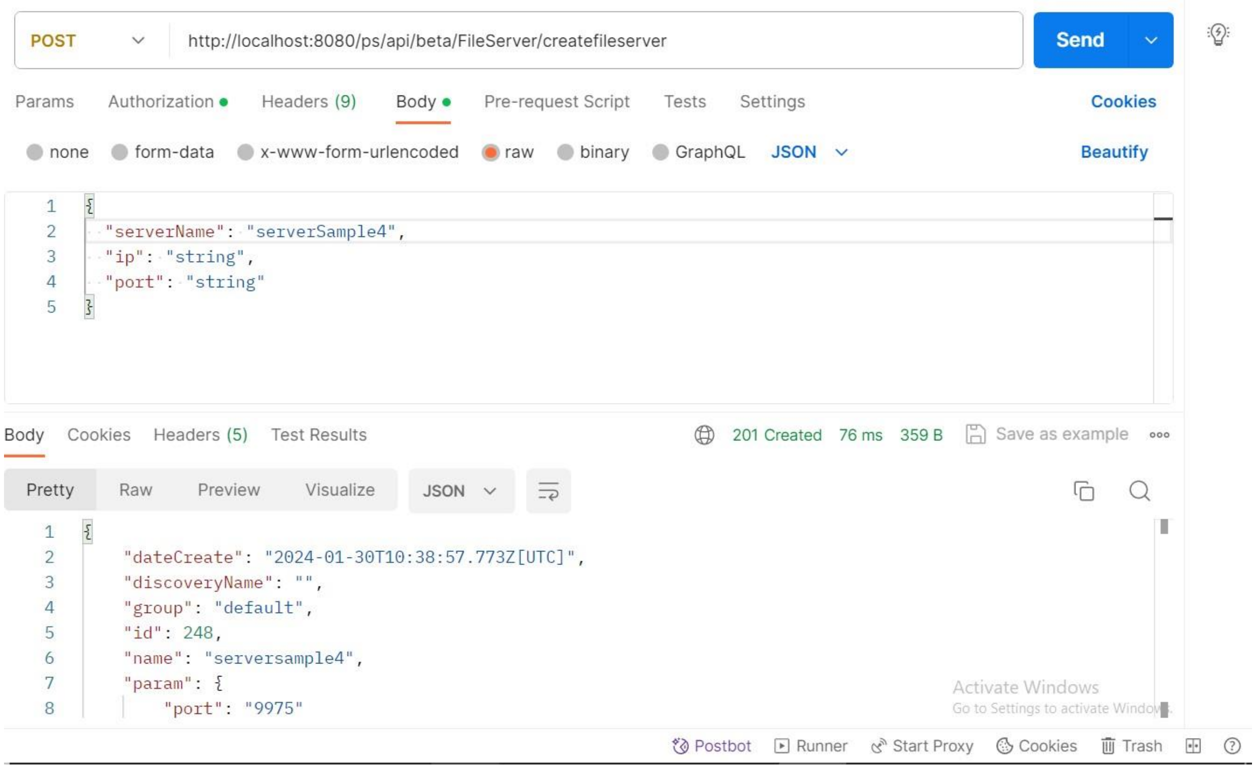
The authorization header will be automatically generated when you send the request. Learn

Body | Cookies | Headers (5) | Test Results — Status: 200 OK Time: 3.67 s Size: 344 B — Save as example

Pretty | Raw | Preview | Visualize | JSON

```
{
    "dateCreate": "2022-11-28T09:53:20.653Z[UTC]",
    "discoveryName": "",
    "group": "default",
    "id": 30,
    "name": "3",
    "param": {
        "port": "9975"
    },
    "serverSubType": "file",
    "serverType": "file",
    "url": "localhost"
}
```

Figure 10: Example 1 - GET operation: Getting File Server Information



POST http://localhost:8080/ps/api/beta/FileServer/createfileserver Send

Params | Authorization | Headers (9) | Body | Pre-request Script | Tests | Settings | Cookies

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON | Beautify

```
{
  "serverName": "serverSample4",
  "ip": "string",
  "port": "string"
}
```

Body | Cookies | Headers (5) | Test Results — 201 Created 76 ms 359 B — Save as example

Pretty | Raw | Preview | Visualize | JSON

```
{
    "dateCreate": "2024-01-30T10:38:57.773Z[UTC]",
    "discoveryName": "",
    "group": "default",
    "id": 248,
    "name": "serversample4",
    "param": {
        "port": "9975"
```

Postbot | Runner | Start Proxy | Cookies | Trash

Figure 11: Example 2 – POST Operation: Creating File Server