Section 1

Hello, and welcome back. In this section, we are going to see how our SAP Netweaver Gateway works.

For example, let's assume that we are working as a developer. We will be working Eclipse with the UI5 plug-in, and there will be some AJAX calls which will be responsible for getting data from outside world to our SAP UI5 application. These AJAX calls will be actually hitting your SAP OData services, which will be responsible for creating, reading or updating any new entries in your SAP tables.

That's how you communicate via SAP. This is a clear separation between your UI and your backend infrastructure, which is what SAP wanted to have with the SAP UI5 application development.

Now, what will happen when the customers will be using this application?

Basically, the SAP UI5 application will be running their mobile, tablet, or desktop – whichever device they choose to work with. The SAP UI5 project will be actually hosted inside your SAP.
A few parts of your project will be actually hosting this web application. In a nutshell, your SAP UI5 application is not a device-specific app like your Android or iOS app, but is actually a hosted application. This hosting is done in your server, and the server which we are using is SAP Netweaver Gateway server.

If you've worked previously with Tomcat or any Apache server, or any kind of server which basically hosts in application, your SAP UI5 Project will be hosted in that web application server, and will be running in your client browser. That's how they will be opening the application. Once they open the application, the communication for data will be done by your OData services.

This is how the SAP UI5 application will run in your production landscape. All these systems will be actually production systems and your client will be using a web browser to open this UI5 application.

Now, let's see about the services and how this SAP Netweaver Gateway is actually configured.

To understand the SAP Netweaver Gateway in retrospect of SAP UI5 application development, we have to understand three parts of it: how the

Netweaver Gateway fits into SAP, how the service creation is done with the Netweaver Gateway, and how the testing is done for further development.

So the first main question which you can ask now is: how can we create the services? These services will be actually responsible for OData communications, so how can we create them if you want to create a new application?

Second is: how can we expose them to outside world? How can we put it in a web server where anyone can access the data via OData? We will see into the authorizing concept, and the authorization process will be in-built with our services.

The third main question is: how can we test the services that we just created?

These are the three main questions which we will be seeing now, and will cover your SAP Netweaver Gateway development altogether. We will be diving into each section and see how they're done.

How can we create the services?

The transaction which you need to remember is "SEGW".

Let us open our SAP system, and go to "SEGW" transaction. You can see that this is the SAP NetWeaver Gateway Service Builder. There are many services created here, and it's very simple to do it.

Create a project here and name the service. For example, "Z_DEMO_TEST3". Write a description – for example, "testing SAP UI5". Give the Package name. Make it a Local Object. It will create four folders inside the newly created service: Data Model, Service Implementation, Runtime Artifacts, and Service Maintenance. This is how we create our service.

The first thing which we have to do after creating a service is to tell the service which are all the elements or entities which the service will be involving, or which are all the fields which will be read or dealt with by the services. We can do pretty much automated things by using RFC or your BAPI calls here. For now, let's keep it very simple and we'll show the basic way which has been used many times, because sometimes we don't have automated BAPI for our custom service created. In that case, we will have to go with and understand this service creation process for custom tables.

Let us see the structure of a custom table first. Go to /ose11 transaction. At the Database table, we can type: for example, “ZUI5_demo_table”. This is a very simple table with five fields, and one of them is the key field. The five fields are: sales documents number, client, date on which the sales document was created, name of the person who created this, and the customer PO number. We can also use the standard tables for this, but for the demonstration purpose, this Z table has been created if we want to write and update statements as well.

If we want to see the table content, these are all the data which were put by your table maintenance which is the “sm30” transaction. This is standard ABAP, and basic maintenance and table. “ZUI5_DEMO_TABLE” is the table which we are going to use for our SAP UI5 application services.

Going back to the SAP NetWeaver Gateway Service Builder, right-click on Data Model, and go to Import the DDIC structure which is a three-step process.

Under Create an Entity Type or Complex Type, put in the table name which the service is going to use. Also put the table name of the ABAP Structure which the service is going to use from our SAP. Click Next.

Next it will ask all the fields which will be included in the service, and select all of them. Click Next.

It would be asking if you want to have any one of the fields as key. For example, make the ZVBLEN as the key. Hit Finish.

If you expand Data Model, you can see inside Entity Types the Properties which are added to the ZUI5_demo_table structure, and you can see the details of each of them. When we imported these entities from our table, we are telling the service that these are all the fields which will be involved. Now, we have to tell the service that if we want to have the commonly known CRUD operation; should we be able to create, update, sort, or filter it – these are all the operations which we have to tell if allowed to do on these fields, by marking the tick boxes. Save this.

The next step is to generate the provider classes. These are some of the classes which will be generated. We will have to overwrite or implement these classes so that whenever we are calling the service, the implemented ABAP call will be run and whatever result is returned from those classes or interfaces will be shown in the service response. Make it as a local object.

Once we are able to create the service implementation, then we can see inside Runtime Artifacts that there are new files which have been created. The DPC stands for the Data Provider Class and the MPC stands for Model Provider Class. We will be implementing the DPC and the MPC.

Before doing that, we would like to register the service under Service Maintenance. You can do it afterwards as well, but we will test this service for now.

Technical Service Name will be the name of our service. Assign this as a Local Object. Click on Continue.

Go to Maintain, which will automatically redirect to the Activate and Maintain Services transaction. Under ICF Nodes, we can see the status of the service. Green-colored status means this can be reached from outside SAP via web browser.

First, we will test with the Gateway Client. Click Execute. Under HTTP Response, you will be able to see some results. The HTTP status code is 200, which is okay. If we change the format to JSON, then it will be more readable.

Now, let us do a browser call to show how it looks in the browser. The SAP system will be asking for credentials. It will not allow to see the service or any data before giving some authentication. Use the username and password of your SAP logon. You can now open this in your web browser, and see the results. You can also see the metadata information, which will give all the fields involved in the services and basic information.

Going back, we created our services. We told the services which are all the data elements or entities which will be involved in the services. We directly imported all entities from our Z structure. We registered the service, and then we checked if everything is working or not.

We have seen the SEGW transaction where we are creating our services. When you want to tell your SAP system how to expose your services to the outside world, then we use the SICF transaction. If you have done your web application development, you might have seen SICF where we maintain all our services present in your SAP system.

Let us open /osicf. Under Service Name, search for Z_*. You can see that there are services, all of which are activated, and can now be deactivated. Z_DEMO_TEST3_SRV is activated because we previously went inside the SEGW Service Maintenance folder, where we can directly register and activate our services. We don't have to go to SICF to do that, but we can still go to the

SICF transaction to activate and deactivate your services. If we deactivate the service, then the service will not run.

Now, we can also test the service directly from SICF, and it will ask to open it in the web browser.

One more transaction which you will be seeing is the Service Maintenance transaction. This t code is actually opening your service maintenance screen where you are seeing all the services present in the system. But when we go via the shortcut, we are directly shown our own services.

This is the basic maintenance screen which will be used to do all kinds of applications. We can deactivate the ICF node, which will stop our service; we can go to the gateway client (also referred to as SAP internal browser) to test our services; we can add new services and new system aliases; and, we can see the error logs which is important when we are developing the services, as we will also have some issues to fix.

Now let's go into the part where we will be actually doing some of the operations with our services.

For now, in the web browser, all you can see is the metadata and the basic service information. The first thing we want to do is to implement the class.

In the SAP Netweaver Gateway Service Builder, if you go inside the Service Implementation folder, you can go to ZUI5_demo_tableSet, and now there is the CRUD operation. The CRUD operation is very popular and it covers almost four parts: creating, reading, updating, and deleting your entries from SAP.

Let us first implement the GetEntitySet, It is like a SELECT *; it will give all results of your table with all of the structures which you have defined. Right-click on GetEntitySet, and Go to ABAP Workbench. Save the project.

In the Class Builder, we have to navigate inside Methods, then Inherited Methods, and we will see the GetEntitySet. Right-click on GetEntitySet then Redefine, and it will open this method in edit mode.

You can see some of the parameters which has been imported by this method, and two of the parameters have been exported and are basically the exception.

There are many imported parameters which are from where you can pass some of the filtering criteria, but the main focus or attention should be at the ET_ENTITYSET. It will be returned by this method, and whatever information

ET_ENTITYSET will be having will be shown in your web browser once your service is returned. In a simple way, ET_ENTITYSET will be your body of the response.

Let's try to implement ET_ENTITYSET for now. We will do a very basic SELECT *:

SELECT * FROM ZUI5_DEMO_TABLE into CORRESPONDING FIELDS OF TABLE ET_ENTITYSET

Save it and check if things are correct. Activate all of the objects which are selected.

Now, you have your activated object created.

In your web browser, if you open ZUI5_demo_tableSet, you can see the data coming out. Let us showcase this in a very nice JSON format, and you can see that the data which you were seeing in your SAP tables are being shown in the web browser.

Basically your UI5 application will be doing your AJAX calls, and they will be getting these JSON objects and show them or do anything they want to do in the UI5.

This is the basic GetEntitySet which will be a kind of SELECT *. You can also pass filter criteria here, and get specific entries or results by the key field. You've also seen the $top and $skip in the OData queries. We will also be able to do the operation with our services.

You have to understand that all of these are actually done by our ABAP codes. You have to write those coding in the Class Builder if you want to implement some feature. For example: $top=2, so that you will only see two records.

Here, it's giving four records, because we have not implemented the top feature. We would have to give this logic or build this understanding in our ABAP code to deal with top or filtering criteria. We will be getting all those information of what the URL has through these imported parameters. We will be populating our ET_ENTITYSET table, and whatever the final response is there in the table will be shown in the web browser

In the next section, we will be seeing more of service implementation. Hope that creating, registering and activating the services, as well as basic implementation and testing the GetEntitySet are clear to you.

To summarize our section 1, we saw the architecture of SAP NetWeaver Gateway system, and how we need to utilize our SAP NetWeaver Gateway system to build application on top of it. Then, we went to specific to how UI5 will be using SAP NetWeaver Gateway system.

We came to know three main questions which need to be understood to start gateway development. Those questions are:

1. How can we create services?
2. How can we expose them to outside world?
3. How can we test them?

To look for the answer, we went inside the gateway system to know how it is structured. We also understood all the details we have to consider while building a service, like knowing the t codes; seeing how basic service is created and exposed, and how they are accessed from the internet; and, what, as a developer, we have to do to build those services.

At the end, we implemented our first service where we are able to read multiple records of a table from SAP.

In the next section, we are going to see the very famous CRUDQ operation in details, and see all the scenarios that can come in implementing the service for a productive backend development.

Section 2

In this section, we are going to see CRUD operation with SAP NetWeaver Gateway.

What is a CRUD Operation? Sometimes, it is also referred as CRUDQ. Basically, CRUD stands for Create-Read-Update-Delete, and Q stands for Query.

For example, this is your backend table.

In your SAP UI5 application, you are going to use AJAX calls. Via AJAX calls, you are going to maintain backend data or information

For example, you have to maintain this table. What kind of functionality do you require? First of all, you can create, update, and read some records based upon some key column inputs or key value inputs. You can also read the entire records, like the SELECT * operation. You can also delete some of the records. These are all the functionalities which you could require to maintain a single backend table or multiple sets of backend table, and will be exposed via your gateway services.

You will be using normal AJAX calls to create, read, update, delete or query records. We implement this CRUD operation in the gateway so that those backend functionalities is available to your UI5 application via AJAX calls.

So let's see how to do it.

Let us go to SE11 so we can show the table on which we are going to do the operation. The table is ZUI5_DEMO_TABLE, and Display the content.

It has five columns. One of them is Mandate which specifies which client we are in. If we see the records, there are four records created by my services, and can be updated and can be read.

So let's see how those operations can be done via your gateway.

To go into the gateway, go into the SEGW transaction. Create a new service, for example: Z_DEMO_TEST4. Put in the description: demo for recording. The package can be given as $TMP. If there is no package, you can either write it down or just click on the Local Object. Click on the check button.

It created the project to implement the services. You can save this, or you might lose the data if you go into some other transaction. The first step after saving is to import the structure which your service will be dealing with. We will be dealing with the table structure which was just showed to you: ZUI5_demo_table. Put in the Entity Name as well. These names can be different, but we're making it the same so it would be easy for us to recognize in our implementations. Go next.

Select all parameters, and click Next.

We have to mention which is the key field: ZVBLEN. This table may not have some functional sense or may not be functionally accurate, as this table structure is just created so that you can understand how to implement CRUD operation for now. Hit Finish, and Save.

We have imported the structure which will be used by the services. Under Data Model, if you click inside Entity Types, we can see the structure, in which you can see its individual properties.

Select all of these, and click Invert. You are saying to your gateway service that you will be requiring the Create operation for all of these entities. They are creatable, and updatable. Save.

The third step is to go to the project, and Generate Runtime Objects. It will generate the Model Provider Class, and Data Provider Class. We will be implementing DPC, so that we will be getting the data from our services via our AJAX calls which you will be using in your UI5 application. Make this as a local object.

Now that the runtime objects are generated, if you go inside Runtime Artifacts, then you can see that both DPC and DPC extensions are created, as well as MPC and MPC extensions. We have to focus on DPC because when you query your services from your browsers, the data which you will be getting will be actually given to you by the BAP syntax which you are going to write, and those are done by the DPC.

The next thing to do is go to Gateway Hub, and register your service. This is giving us the description of what will be the service name and all. Select Local Object, and it will automatically fill-in $TMP in the Package Assignment. Continue, and save.

Now that you have registered your service, you can go directly on maintenance and see if the service is running fine. The service has been automatically selected, and you can see that the OData node is active as the status is green.

You can go to your gateway client, also referred to as a local browser inside SAP, to test your services. Press Execute. It gives you some result, and the status code is 200 which means everything is okay.

Now, we have successfully registered the services, so let's start implementing the CRUD operations.

Let's go back to the service builder which is the SEGW transaction. We are going to implement all the CRUD operations one by one. The first operation which we will be implementing is Query, because it is like the SELECT * and is the most simple one. Right-click here, and Go to ABAP Workbench. As it says it has not yet been implemented, click on the check button to implement.

You can see that it opened an SE80 transaction, or the Object Viewer. You can go inside Methods, and inside Inherited Methods, there are 5 methods which we will be implementing.

The first method which we would like to implement is GET_ENTITYSET. Just to give you an overview about this query, this is for selecting multiple columns like your SELECT *.

We have to implement this function: ZUI5_DEMO_TABLES_GET_ENTITYSET, in which the first part is the name of the structure we gave while importing the Z structure from ABAP, and GET_ENTITYSET is the method which we are going to implement.

Right-click on the GET_ENTITYSET, and Redefine. It will open the method which we have to implement, then we can delete all the command section here. One thing you can see is that there are so many parameters here, some of which are import parameters to this method, and some are export parameters. You can see ET_ENTITYSET, and also some exceptions which can be passed if some error occurred in your program and you would want to modify some issues or mistakes in the import parameters from your services. These parameters can be utilized in your ABAP codes.

We are writing this ABAP code:
SELECT * on the ZUI5_DEMO_TABLE into CORRESPONDING FIELDS OF TABLE ET_ENTITYSET.

ET_ENTITYSET is our export parameter. It is actually the structure of this table. This is because we had created this service and imported the table structure, therefore the exporting table structure will be similar to that table.

We are just selecting the entire data of this table, and passing it to ET_ENTITYSET.

This is the syntax of CORRESPONDING FIELDS. Sometimes it's not good for the performance, but for testing outlets, go for it.

Save it, and activate GET_ENTITYSET. It selects some of the other objects which are dependent upon our GET_ENTITYSET. Press Continue.

It's activated.

So how can we test this GET_ENTITYSET? Go to Gateway Client, and use this URL:

/sap/opu/odata/sap/Z_DEMO_TEST4_SRV/ZUI5_demo_tableSet/?$format=json

Z_DEMO_TEST4_SRV is the name of our service, ZUI5_demo_tableSet is how it recognizes the name of the method, and ?$format=json is the additional functionality which specifies what the format of output you want to see.

The HTTP method should be GET because we are reading the information. If you press Execute, then you can see that all records of the table are given to us. You can see that there are 4 records in the table, and are shown here in the gateway client as well.

The same thing can be done in your web browser. You have to add the host name and the port number, and the rest is the same as what you write in your gateway client. When we enter, we get the same results.

Gateway client testing should be done first before the browser testing.

This is the implementation of your Query, which is like your SELECT * or selecting multiple columns.

The next thing which we are going to see is the Read operation, which will be like SELECT single.

For example, you have this table and you want to read this column which has the key value of 2. To do that, we have to implement GET_ENTITY. Previously, we implemented GET_ENTITYSET to read multiple or all records.

We go to Class Builder, right-click on GET_ENTITY, and Redefine. Remove the command section.

This is the development code which is a very simple ABAP coding.

We are going to use some of the importing parameters here because the key values will determine which column to read from this table. For example, if we want to read the second record, then we should be passing this value so that we can filter this table with this value. This particular value will be passed via our URL, and that URL value will be available to us in our methods by our importing parameters. These importing parameters, specifically the IT_KEY_TAB, will be having all of our key values which is passed via URL.

We're going to READ TABLE it_key_tab WITH KEY name = 'Zvblen' (which is the key field of the structure), therefore, we can read this name. Once we get the value which is passed via URL, then we can easily write a SELECT single * from ZUI5_demo_table (the structure) into CORRESPONDING FIELDS OF er_entity (which is exported from this function) where zvblen eq ls_key_tab-value (which is what we got from our URL). Save this code, activate, and put a debugging point.

Go to the Gateway Client, and the URL to access the GET_ENTITY is:
/sap/opu/odata/sap/Z_DEMO_TEST4_SRV/ZUI5_demo_tableSet('0000000002')

This is the same as with GET_ENTITYSET, but with additional key values in the bracket. If we Execute, it will stop at the debugging point.

This IT_KEY_TAB is actually the importing parameter. If we go inside it, we can see that it has only one value, and has 3 columns. Column 1 is Row, Column 2 is a CString value called NAME, and Column 3 is also a CString value called VALUE.

Zvblen is the key of the table and the value which we passed via URL is the 0000000002, and we are just reading that particular record into ls_key_tab.

IT_KEY_TAB is a table basically from the structure /wbep/s_mgw_name_value_pair. Inside our methods, we're using this structure so that we can have a local variable which has the same structure of this table. Where we can store our records is available in IT_KEY_TAB, and we can pass it to our SELECT single query. Here, you can see this local variable we're accessing which is the ls_key_tab-value.

Going back in our ABAP debugger, the value of the lead operation if we put the debugging point here should be 0000000002. After the operation, the sy-subrc value will be zero. If it is not zero, then you can throw an exception here, which will be reflected in your service, and the message will be shown. If it is successful, then it will just populate our er_entity table.

First of all, let us see sy-subrc; it should be zero. Double-click on er_entity. The value inside er_entity will be the single record which is read, and which is the same second record as presented in our /ose11. This is something which will be exported.

Just to give you a heads up, we implemented this READ operation. The URL which is required is this one. Because it is like a single selection, you should pass a key to your selection so that a single record can be read. You can have multiple keys as well, separated by a comma, so that your ls_key_tab will have multiple records.

This is how simple it is to implement your Read operation. In the next section, we will see the rest of the CRUDQ operation.

In this section, we are going to see the Create, Update and Delete functionalities of the CRUD.

Create is used to create new records in your table, and the POST operation of your HTTP AJAX request is used here. The Create process is a bit tricky to run or use, but the implementation is quite simple.

So let's go to our SAP screen and see how to implement the Create operation.

To implement, we will go back to the Class Builder screen for our project. We can see the CREATE_ENTITYSET here in the methods. Right-click, and select Redefine. You can see here that we have the method which we have to implement. Delete the call method. Copy this code here.

What this code does is actually reading an IO_DATA_PROVIDER. Save, and Activate.

What basically happens when you are sending some data to create a new record is that those data will be available in IO_DATA_PROVIDER. This is actually a table which you have to read, and you will be importing the values into this particular work area: ls_request_input_data.

Once we get the values, then we're directly calling the INSERT statement and using the ls_userinfo. ls_userinfo is actually a type of table, and we are

individually copying all of our past data from ls_request_input_data into ls_userinfo, so that ls_userinfo has 4 fields or entity.

We are also putting all of the data back to er_entity, which is the variable which is exported out of this method.

Let's now run this method using our gateway client. If you go to gateway client, you first have to get the data. To get the data of one particular record or row, you have to use GET_ENTITY which we created in the previous section, and Execute. Now we will be copying this response, by clicking Use As Request. Once we do that, then in the left side, you can see that you called some data here, and the content type is copied as well.

We have to create a new record. What we have to use to create a new record is POST. Then your Create methods will be automatically executed.

You can see there are some errors, and it says "Method Not Allowed". Let us go to this error transaction: /IWFND/ERROR_LOG. This is for checking errors. When you are doing some developments in gateway, it is frequent that you will be getting errors.

It says that the data we provided does not exist in the archive. I'm passing this as my parameter, and I don't require this parameter. Get rid of that, and try to execute.

The problem was created by our parameter here, because when you do a Create call then the data should be passed in your response. The URL should be ZUI5_demo_tableset which is the GET_ENTITYSET URL. You don't have to specify in the key which records you're going to update, because this is not an Update statement, but a Create statement. We have to give a generic URL, and that URL should be off-set type.

You can see that in our Redefinitions path, we have our Create function here to run this method with those data which we have passed in the response body, and we will be giving this data to the set URL.

One of the important things when we do an AJAX call via UI5 application is that you also have to give the X-CSRF-Token. X-CSRF-Token is automatically copied in the response when the data is used as request, but when you are calling from the UI5 application, you have to manually give the X-CSRF-Token. It is used to prevent forgeries. This is a security mechanism so that the web service only accepts data from legitimate sources who already have X-CSRF-Token

previously. When you do a GET call, you automatically get an X-CSRF-Token, which could be used via POST request as well.

If we are doing this with a web browser, to do a GET request for the first time, the gateway will ask for credentials so the security is maintained via service calls.

Now, let's go to the second functionality which is our Update scenario.

We go to our Update scenario. Here, we will be reading the data entry, and we are going to use the UPDATE ABAP statement. We will debug this program this time so that you can see what is happening inside your ABAP syntaxes.

Let us go to the Class Builder, and this time we are going to implement the Update functionality. Let's Redefine. Get rid of all the commands and coding, and copy and paste this new coding. Save it, and then Activate. We have the active object generated.

We can see that the "Tom" which we created as a new entry is now existing in our table. Now, with the Update record, we will update this created entry, and we will change the name of TOM to JERRY. To test your Call Update, you go to gateway client.

What we have to do now is to read the record, because you have to know which records you want to update. First, you need to do a Read operation first to get exact records, so you could use that response as a request again.

Copy and paste this statement here:
/sap/opu/odata/sap/Z_DEMO_TEST4_SRV/ZUI5_demo_tableSet('0000000007')

Do a GET request. We're getting the "Tom" data which we just created in the table. You can see the HTTP Response, and the data which is in XML is the response body. Click on Use As Request, to copy the same response as a request. This is copied in the left side.

This is going to be our request which we will send to our server. We are going to update the name to "Jerry". What my ABAP program tells now is that whatever data is passed in the body, read that table in ls_request_input_data and update that table accordingly where my ZVBLEN table field is equal to this data record.

So, based upon the ZVBLEN field, we are going to update these table records with the new entries. This is a very simple ABAP program. You have to pass the key in URL from where the update should occur.

In this case, as we are using an Update, the HTTP method should be PUT. Execute, and what we are expecting is that the name of the new record should be changed to “Jerry”. We are successfully able to update the records. As we can see here, the response status is 204. If we see our data browser to see the records, the name has changed to “Jerry”. We have successfully implemented the Update operation.

Let us put a debugging point here to show you what is happening inside. Go to gateway client, and let’s manually select a debugging point here by writing /h, and Execute. Now, we are able to go inside our debugging point which we set. We can see that this is the import parameter, and there is a method of this import parameter called read_entry_data. You have to pass the structure ls_request_input_data, to read this entry data, which is the same as when you created the service. It will also depend upon what response you sent to the service; that response body will be actually read here.

Switch to the next step and see what we got in the variable ls_request_input_data. You can see that our entire record will be there for “Jerry02”.

What we are doing a normal query of Update. We are updating zui5_demo_table, and setting all of the column values to the new record values which we got. We are selecting a record from the ls_request_input_data, and zvblen is the key field which will tell which records to update.

You can see that Update operation is working because when we try to finish this, we’re getting a 204, which is the status code of success.

Let’s go to /ose11, and see the records quickly. You can see that “Jerry02” is already updated.

We are able to see how to implement the Update operation.

Now the last one is the Delete operation, which is quite simple. We will be reading the same table in the same way we did for the Read operation. We will read the key, and we will delete the entry or the row where those keys exist.

Basically these Delete, Update and Create operations should be done by a function module. They should not be done by normal ABAP DELETE, UPDATE,

or CREATE statement directly. It's always better to use the existing function modules than to write your code, because there are data inconsistencies that can arise and there are lots of scenarios which can come.

We will go to Class Builder, and Redefine the Delete method.

Copy our syntax here. Save, then Activate it. Now, it is reading the import parameter which we will pass form the URL, and it's a single import parameter. Zvblen is the key name which we will be reading, and according to that read key value, we will be deleting the record from the zui5_demo_table.

You can also have sy-subrc check. If it is not deleted, then there is some problem, and you can throw these exceptions.

For now, let us only Delete, and expect everything is going to be perfect. If not, then the internal exceptions will kick off and give us the error message. Let us Activate it.

Let us go into the gateway client. We are going to do the same thing. For example, if we do a GET, then we will get the record of the 7th number, which is the record of "Jerry02". To delete this record, we just have to select the HTTP Method DELETE. Execute.

We can get a success message, which is 204, as the 200 series of response messages are of success series. That means my record would have been deleted.

Let us check. Now, we can see that "Jerry02" is not there.

One way to check if your operation is done is you can do one more Read operation for that record. What happens is that it is not able to return anything. Therefore, there is no record now, and that means your Delete operation is successful.

Now, we have successfully implemented the CRUDQ operations. As you have seen, everything we do in gateway is done in ABAP calling or ABAP development, and you just have to expose those ABAP functionalities via NetWeaver Gateway to outside world, which will be utilized by your UI5 application.

This is the summary of Section 2.

In this section, we saw CRUDQ operation, where it stands for Create, Read, Update, Delete, and Query.

We implemented each one of them in SAP NetWeaver Gateway system step-by-step, and tested them in the gateway client and in web browser.

We also came to know the authentication mechanism for NetWeaver Gateway system while accessing the service from internet.

In the next section, we are going to see CRUDQ operation with SAP UI5 app. The procedure to access these services will be a bit different with UI5 app, so we will be going into details of how one will be utilizing the service which you just created in the SAP UI5 app.

You may have one question, “do I need to know SAP UI5 to understand the next section?”

The answer to that question is: the basic understanding of MPC & UI5 will certainly help, but even if you are new to SAP UI5, we will be taking step-by-step manners so that you understand what we are doing.

As a backend developer, we should understand how these services are going to be exposed and utilized in the applications, so that when we are talking to a front end developer, we are aware about what they will be using, and how they will be using the services.

Section 3

In this section, we are going to see how to implement our CRUD operation inside our SAP UI5 application.

In the previous section, we saw how to create the CRUD operation, and we tested that in SAP NetWeaver Gateway Client. We also tested that in browser, but specifically, we didn't test the methods like Create, Update and Delete in the browser. This is because creating those scenarios using AJAX calls is a complicated task, as there are some security exceptions which you have to deal with when you do so.

There are some classes present inside the library which will make it a lot easier, and that would be the go-to method to use your Create, Update and Delete methods inside the SAP UI5 application.

Let's see the gateway where we used to see the Update, Create and Delete processes, and where we actually copied the data. For example, if we GET the details, we can use this as request, we can change these values, and we can do a DELETE, or PUT operation. We have used PUT, POST, DELETE, and GET, wherein the PUT was your Update, the POST was your CREATE, the DELETE was your normal Delete, and the GET is used to Read the records.

Let us show the UI5 application which has already been built so we can test our scenarios.

These are simple sap.m.Input boxes, and there are three sap.m.Buttons. For example, let us create a new record, with the first entry as 0000000011, then the client, the timespan, name and sales order number. You can have labels before these input fields, but as we just wanted to show the CRUD operation, we didn't focus much on user experience.

Let us try to Add it, and see if we have a new entry with ZVBLEN as 0000000011, for the name "Ajay", and sales order number 0000000021, this time span and the client ID.

So if we go to the table and refresh it, you will be able to see this newly created record.

Let us go back to the browser. If we want to change the name of the record to "Sammer", then we can do an Update here. Go back and check the table. It has updated.

Now if we want to delete this record, we can just go ahead and Delete, and let's also verify that. If the record is not present, then it already automatically sensed a successful message. That is something which we need to do as a boundary test condition to see if those conditions or cases might arise in an application.

Let's stick to something which we want to learn today, and that is how to make these 3 features work.

Let us go to Eclipse. Basically we are having SAP UI5 application. As you can see, data_operation is the name of the application. The index page is simple. We have one view page and one controller page, which are created by default.

If you see the view page, there are 5 inputs and every one of them has an ID. These are the values which are displayed inside the input. We have given the width, which is 70% of the webpage. There are only 3 buttons: Add, Update and Delete. This type is here to give the standard coloring of the buttons. After that, we have this press event so if the button is pressed, then it calls onSend from the controller, and it passes the Add parameter.

This is the onSend function inside the controller page, and you are able to see that it has an sOperation. The "Add" value will be assigned to your sOperation, and will be passed while we are calling the onSend function.

You can see here that we are having an AJAX call when we are a doing an onInit whenever the application initializes. This will be loaded at the beginning, or this will be the first function which will be called when you start the application.

This is because whenever we are doing an AJAX call, which is in GET, to some service to read some records, it will ask us for username and password. It will be a standard pop-up message which you will see. Once you give your username and password there, then all the other calls will not require any more accesses, and they will all be using the same credentials. In the same application, if there are furthermore calls, they will all be using the same authentication, and you will not require it.

You can see that we are actually setting the model. We are having a CSRF token here. We are actually passing the headers X-CSRF-Token fetch. We have to store your CSRF token in your model or as a variable.

We are not seeing how to implement this operation using normal AJAX call because that is very complicated, and is also not advisable in your UI5

application to have normal AJAX calls for your Update, Create, and Delete operations. You will be seeing the normal way which will be used in your application, and which is much simpler and better than your AJAX calls. In a nutshell, these AJAX calls are only used to get you authorized.

After that, in the onSend function, according to your parameter it can either be add, update or delete, to do this specific operation.

We are storing all the data values inside a JSON object called oData. We are having keys called ZVBLEN, ZERDAT, ZERNAM, and ZBSTK. To get these, go to the NetWeaver Gateway Client, and you filter it with $format=json. Execute this, and copy. These are all the key fields, and you can copy the exact character types here, as the capital and small letters should be taken care of, and should be mentioned as it is.

The “0000000009” here is only “9”, but you have to pass the leading zeros as well because this is VBLEN field type. That is something which is mandatory for that particular data element to be updated or created.

In the beginning, we are having an oModel. This is the standard class which you will be using to get an access to an object which has all capabilities to create, update and remove.

If you go to browser, and open this documentation of SAP UI5, you can see that the class sap.ui.model.odata.ODAtaModel has many functions like remove, refresh, and create. We will be using this class and the functions present inside this class. This is advisable when you are doing a UI5 application for those operations like Create, Update, & Delete.

Once we get this object, we have to pass this service URL:
ehp7prd.sap.in:8200/sap/opu/odata/sap/Z_DEMO_TEST4_SRV/

Once you put this particular URL in your browser, you will be actually seeing the metadata. The system name is ehp7prd.sap.in:8200/sap/opu/odata/sap/. After that, the service name is Z_DEMO_TEST4_SRV/. This is the URL which will be put in this ODataModel, and get access to all operations like create, update, and remove or delete.

Inside the controller, you are able to see that it has an sOperation. Once we pass this sOperation, then you are just calling these URLs which we saw in the gateway.

Let us go back to the gateway client. After this service URL, we are passing this particular string. If you see the previous sections, this part was different. This will be the first parameter for your oModel.<function> .

When we create, we only pass Set. When we update, we pass Set and which particular value should be updated. When we delete, we also pass the Set with the key parameter which should be removed.

The next parameter will be the data, which is required in the create and update processes. In the delete or remove process, we don't require data because it will just delete according to the key which you have provided.

After that is a null parameter, followed by two functions: if you have functionSuccess, then the first function is executed; and, if it is functionFailure, then the second function will be executed. This is the same throughout the create, update and remove processes.

We are using an sap.m. MessageToast, which is a small popup which appears for about 3 seconds, and says the record is successfully created, updated or deleted.

This is something which we will be seeing again while in our browser. This is how UI5 application or implementation is being made. It just takes the filled values and, according to which button we press, it executes the specific function of our model.

There are few things to remember here. The first is you should not start your Chrome browser directly from the Desktop, because you are doing a local development. If you want to do an AJAX operation to a server from your local host, then they will not be allowed in the Chrome. You have to open Chrome in a security-disabled mode. To do that, right-click on the Chrome icon, and select Properties. Copy the location of your Chrome application, and change your directory to that particular location on cmd.exe. Write this special command:
chrome.exe --disable-web-security

Sometimes, this creates a lot of confusions. Your AJAX calls are not working because your browser will not allow your server calls from your local host, so you have to disable your web security.

This is something you have to do in Chrome. Firefox will work fine without disabling the web security.

Let us open the Eclipse, and copy the URL of the application which we are seeing in the browser.

This is the challenge method, which is because we wrote one GET function call in the init() method. This will allow us to get authorized from the SAP. This is the standard SAP login, and we have to give the username and password.

To put it in a simple way, this call which we do in the init() is there to get authorized initially. Once you are authorized by your backend, then you will be able to do all other operations like create, update, and delete. You need to get authorized once.

We can get authorized even by calling the URL in a new browser, like the incognito mode, for example.

Now that we are authorized, our temporary files have the information of authentication, so that the next time we do a call, it will not ask for any more username and password for this particular session.

The same thing happens for the first time opening the application. You will be challenged and your credentials will be taken, and all operations are performed with that authentication. To do this operation, we are using this type of oModel: sap.ui.model.odata.ODataModel .

There will be issues when you are dealing with these services. If you get errors, the best way is to look at the gateway and go to the transaction /IWFND/ERROR_LOG to see all errors which have occurred. Click on the time stamp, and you can get all the details below Error Context.

(Video course repeats the whole demonstration from security username and password recognition, to the create and update operations, to sOperation in the controller)

The key of the table is Zvblen. You can see here that we are putting a \'. This is because when we want a new URL, we cannot write '' because it stands for end of the string in the javascript. Therefore, use \' which is called an escape sequence. To make ' appear in the string, we have to use an escape sequence in the beginning and in the end of the key parameter. In the Chrome browser, if you go to Inspect Element and to the Network tab, and try do an update, you will see that your URL looks like this:
ehp7prd.sap.in:8200/sap/opu/odata/sap/Z_DEMO_TEST4_SRV/ZUI5_demo_tableSet('0000000014')

When you are doing this kind of development, you will be going into Console many times, to test some snippets and try a lot of things.

Now, if we try to delete this record, the same thing will happen – you will be passing the oModel for deleting this key, and it will be deleting the record. Let us check the table. Now, we have the record deleted.

You can also push this code to SAP.

In Eclipse, right-click on data_operation, go to Team and select Share Project. Select which SAP system where we want to push the UI5 code. It will ask for the client, username and password for the SAP logon.

After that, we will be selecting the BSP application. If you have any BSP application, then you can filter it from here, or you can create a new BSP application.

We will create a new BSP application for this demonstration. Let us give the name, description and the package. If you have any existing package, you can manually type it in, or you can filter as well. Let us use $TMP to make it a local object. Click Finish.

Now, we have successfully created this project inside our SAP.

We will push our shared entire project. Right-click on data_operation, go to Team and select Submit. Select all the files which will be shared to SAP, then click Finish.

Go to the SE80 transaction, and go inside the BSP application. Select the object name which we just pushed (ZDATA_OPERATION).

Let's try to run our application. Navigate into the index page and test it. The default behavior is that it will open in Internet Explorer. Some of the fields like client name, language and app caches, are appended at the end of the URL by default.

Now we are inside this application which is running from SAP. Let us try to do the same add and update operations.

We can check here directly that the record is there.

This is the summary of Section 3. With this section, we came to the end of our SAP NetWeaver Gateway course.

In this course, we saw gateway implementation where gateway exists within our SAP backend. This model is also sometimes referred to as embedded deployment, and is the preferable model for learning gateway.

We saw CRUDQ operation from SAP UI5 application, and we came to know how the services we built in the previous section will be utilized by the front end developer in the SAP UI5 applications.

As these services are exposed as REST-based services, or specifically OData protocol-based services, they can be utilized in many other applications. If you are building an Android or a Java application, or even PHP applications, these services will be able to communicate, and do the exact work of giving data and executing operation in your SAP backend.

SAP UI5 already provides some of the libraries and classes which are going to be used by your front end developer team to make their life a lot easier. We also saw how they will be going to use those libraries.

After that, we saw how to push SAP UI5 application to SAP, how to launch the application from SAP into your browser, and how to use your application. That will be exactly how customers are going to use your application, but the difference is that they will be having a quick launch URL for your application instead of executing it from your object repository.

We will be also giving some bonus section with this course which will explain how you can do the same operation over SAP HANA.

The new lecture will be talking about how to create basic applications on SAP HANA, how to expose them as REST-based services, and how to deploy SAP UI5 app on top of it.

We will be probably taking example of our IoT-based course, which is one of our Udemy courses and is also one of the top courses in the SAP category. Please check this course out as well, as it will be a very valuable addition apart from gateway experience.

Please give us nice feedback and rating. This will help us to add new materials and goodies to this course in the future. Once new things come into the market related to SAP middle way development, we will be presenting and adding them to this course as a bonus section.

Thank you for being part of this course, and goodbye.