Creating REST APIs based on SQL statements

IBM Db2 for i and integrated web services integration

By Nadir Amra, IBM Software Engineer
Published 01 June 2016 (updated 05 May 2024)

Abstract: You may already be using an integrated web services server to expose ILE programs and service programs as RESTful web services. This tutorial introduces a powerful new feature of the integrated web services server – the ability to deploy SQL statements as RESTful web services.

Introduction

For many years now the integrated web services for IBM i has focused on giving IBM i customers the ability to expose Integrated Language Environment (ILE) programs and service programs as REST and SOAP APIs (also known as web services). Using the HTTP Web Administration GUI interface and with a few clicks of the mouse button, you magically have an API that is based on a program or service program that may be written in RPG, Cobol, or C, and even command language (CL)!

In the latest release of the IBM i operating system (as of August 2019), IBM i 7.4, this same interface can now be used to deploy SQL statements as Representational State Transfer (REST) APIs, enabling IBM® Db2® to act as a RESTful service provider.

In this tutorial, we take you through the steps of deploying SQL statements as REST APIs.

Prerequisites

This section lists the software prerequisites and the prior knowledge that you need to possess to create REST APIs using SQL statements.

Software

To get all the PTFs required by the integrated web services server in support of REST, you will need to load the latest HTTP Group PTF. The IBM Support web page *IBM i Group PTFs with level* lists the HTTP group PTFs for each of the supported releases of the IBM i operating system.

Note: The steps in this article were performed on IBM i 7.3. Panels may look different if you are on an older or newer release. And if you are on an older release, some features discussed in this article may be unsupported on that release.

Assumptions

Before reading this tutorial, you need to read "Part 1: Building a REST service with integrated web services server for IBM i" in order to have a basic understanding of the REST principles and the terminology used. You should also be familiar with the fundamental concepts of JavaScript Object Notation (JSON) and XML.

The RESTful application

The example we use in this discussion is a sample Student Registration Application (SRA). This example is used to show how an ILE service program is exposed as a REST API in the tutorial "Part 3: Building a REST service with integrated web services server for IBM i". In this tutorial, we use the same database file to demonstrate how we can achieve the same functionality but without a service program as the interface to the database file.

The student registration management functions we want to provide in this sample SRA application must enable you to:

- Register new students
- Edit registered student information
- List registered students
- Get information about a student
- Remove student registrations

The only object we have is the database file, STUDENTDB, where student records are stored.

Things to get done before deployment

When deploying a RESTful web service, you should have answers to the following questions at the bare minimum:

1. How do I want the URIs to look like?
2. What HTTP methods will the resource support?
3. What incoming content types should be supported?
4. What type of data should be returned?
5. What SQL statements should be used to implement the APIs?

Table 2 shows a summary of the mappings that we want between HTTP methods and Uniform Resource Identifiers (URIs) for the SRA.

Table 2. HTTP method and URI mappings

HTTP method	**URI**	**Description**
GET	/*context-root*/students	Returns all student registrations
GET	/*context-root*/students/{*id*}	Returns student registration
POST	/*context-root*/students	Registers a new student
PUT	/*context-root*/students	Updates registered student
DELETE	/*context-root*/students/{*id*}	Removes registered student

Note: The default *context-root* for an integrated web services server is `/web/services`. The context root for a server can be changed.

For each of the URI mappings, we need to identify the SQL statements that will be used and to associate a procedure identifier with the SQL statement. You need this information when deploying services based on SQL statements. Table 3 shows a summary of the mappings that we want between URI and SQL statements for the SRA.

Table 3. URI mapping to SQL mappings

URI	**Procedure identifier**	**SQL statement**
/*context-root*/students	GETALL	`SELECT * from STUDENTDB`
/*context-root*/students/{*id*}	GETBYID	`SELECT * from STUDENTDB`

		`WHERE "studentID" = ?`
/*context-root*/students	ADD	`INSERT INTO STUDENTDB ("studentID", "firstName", "lastName", "gender") VALUES(?,?,?,?)`
/*context-root*/students	UPDATE	`UPDATE STUDENTDB SET "firstName" = ?, "lastName" = ?, "gender" = ? WHERE "studentID" = ?`
/*context-root*/students/{*id*}	REMOVE	`DELETE FROM STUDENTDB WHERE "studentID" = ?`

What you need

The example REST API developed in this tutorial assumes a database of student registrations and focuses on allowing you to retrieve, add, delete, and update these student registrations using normal REST conventions.

Step 1. Set up the application database file

In this example, the STUDENTDB DB file will be created in the STUDENTS library. If you have done the steps in part 2 of the series of articles, you can skip this step. To create the library, issue the following CL command:

```
CRTLIB STUDENTS
```

To create the table, issue the following SQL command:

```
CREATE TABLE STUDENTS/STUDENTDB
 ("studentID"  FOR COLUMN studentID CHAR (9) NOT NULL,
  "firstName"  FOR COLUMN firstName CHAR (50) NOT NULL,
  "lastName"   FOR COLUMN lastName  CHAR (50) NOT NULL,
  "gender" FOR COLUMN gender CHAR (10) NOT NULL,
  PRIMARY KEY ( studentID ))
  RCDFMT studentr
```

To populate the table with sample student registration data, issue the following SQL command:

```
INSERT INTO STUDENTS/STUDENTDB
 (studentID, firstName, lastName, gender)
 VALUES('823M934LA', 'Nadir', 'Amra', 'Male'),
       ('826M660CF', 'John', 'Doe', 'Male'),
       ('747F023ZX', 'Jane', 'Amra', 'Female')
```

You must ensure that the user profile that will be running the service has authority to the library and database file. In this example we will be using the default user profile for the server, QWSERVICE. So, issue the following CL command:

```
CHGAUT OBJ('/qsys.lib/students.lib/studentdb.file')
       USER(QWSERVICE)  DTAAUT(*RWX)
```

Step 2. Create the integrated web services server

To deploy an ILE program object as a REST service, you need to have an integrated web services server created, and it must be version 2.6 or later. If you have one already created, you can skip this section. If you need to create one, see "Part 2: Building a REST service with integrated web services server for IBM i" to learn how to create a server.

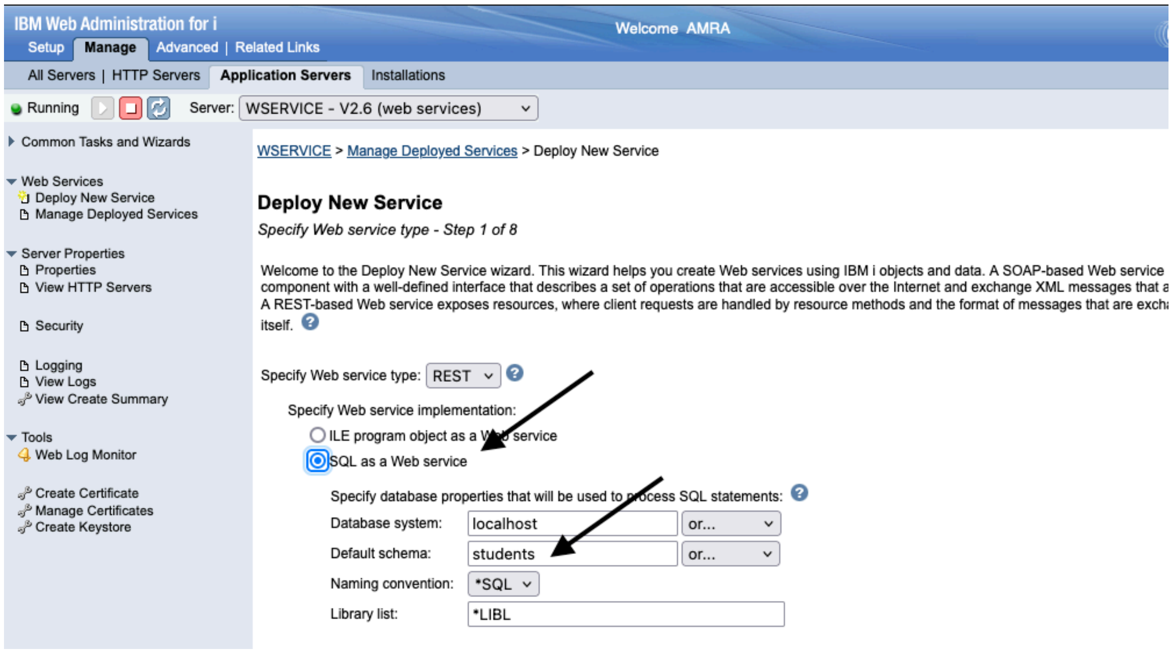
Step 3. Deploy the SQL statements as a RESTful web service

Now we deploy the SQL statements that make up the SRA as a RESTful web service.

Step 3-1. Deploy SQL statements as a web service

Click the **Deploy New Service** wizard link that is in the navigation bar. You should see the page in Figure 1.

Figure 1. Deploy web service – step 1



Deploy New Service

Specify Web service type - Step 1 of 8

Welcome to the Deploy New Service wizard. This wizard helps you create Web services using IBM i objects and data. A SOAP-based Web service component with a well-defined interface that describes a set of operations that are accessible over the Internet and exchange XML messages that a
A REST-based Web service exposes resources, where client requests are handled by resource methods and the format of messages that are exch
itself.

Specify Web service type: REST

Specify Web service implementation:
ILE program object as a Web service [✓ SQL as a Web service]

Specify database properties that will be used to process SQL statements:

Database system:	localhost	or...
Default schema:	students	or...
Naming convention:	*SQL	
Library list:	*LIBL	

This page gives you the option to either deploy a SOAP or REST web service, and to choose whether the web service will be based on an ILE program/service program or SQL statements. When you select **SQL as a Web service**, the wizard shows a page requiring various database properties that must be set.

By default, the database system that processes the SQL statements is the local host (the system hosting the integrated web services server). You can specify a remote system if the database files reside on a remote server.

In this example, we have specified the default schema to be **students**. The system uses the default SQL schema to resolve unqualified names in the SQL statements. For example, in the statement `SELECT * FROM MYTABLE`, the system looks only in the default SQL schema for MYTABLE. The following conditions apply, depending on whether the naming convention is set to *SQL (SQL naming) or *SYS (system naming):

- For SQL naming, if *LIBL is specified for default schema, then the first or second entry (whichever is not *LIBL) in the library list becomes the default schema. If the first and second entry in the library list is *LIBL, then the user profile becomes the default schema.
- For system naming, if *LIBL is specified for default schema, no default SQL schema is set, and the system uses the specified libraries in the library list to search for unqualified names.

You can specify the following naming convention to use when referring to tables:

- *SQL indicates that SQL naming (as in **schema.table**) should be used.
- *SYS indicates that system naming (as in **schema/table**) should be used.

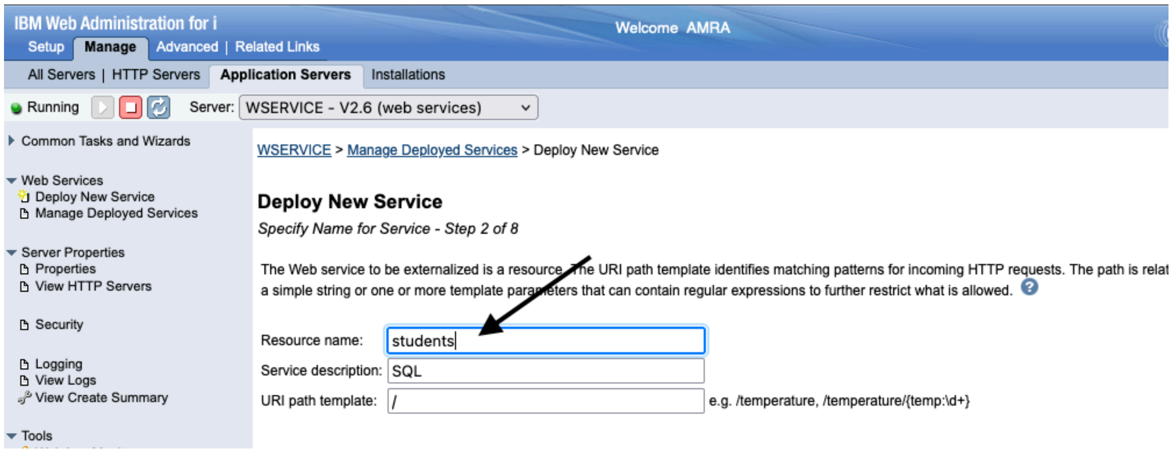
Finally, you may specify one or more libraries that you want to add to or replace the library list of the server job. The system uses the specified libraries to resolve unqualified stored procedure names, and stored procedures use them to resolve unqualified names. To specify multiple libraries, use commas or spaces to separate individual entries. You can use *LIBL as a placeholder for the current library list of the server job.

Click **Next** at the bottom of the page.

Step 3-2. Specify a name for the resource (web service)

Now we need to give the web service (that is, the resource) a meaningful service name and description. The resource name has been changed to **students** (see Figure 2).

Figure 2. Deploy web service – step 2



WSERVICE > Manage Deployed Services > Deploy New Service

Deploy New Service

Specify Name for Service - Step 2 of 8

The Web service to be externalized is a resource. The URI path template identifies matching patterns for incoming HTTP requests. The path is relat
a simple string or one or more template parameters that can contain regular expressions to further restrict what is allowed.

Resource name:	students
Service description:	SQL
URI path template:	/ e.g. /temperature, /temperature/{temp:\d+}

You can set a URI path template for the resource. For this example, we do not need to specify anything because the path to the resource after changing the resource name is what we want:

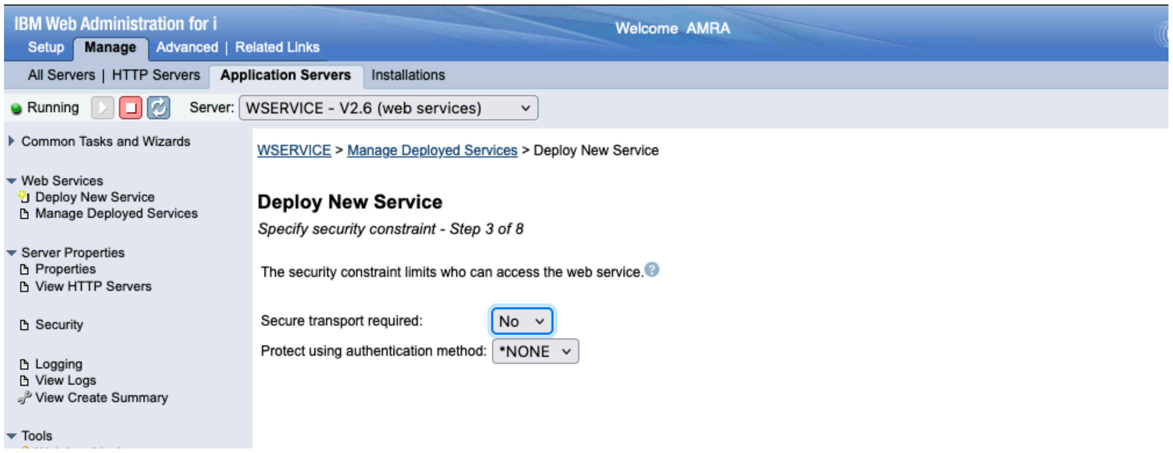
/*context-root*/students

Click **Next** at the bottom of the page.

Step 3-3. Specify security constraint

The security constraint limits who can access the web service. To protect the web service, an authentication method other than *NONE needs to be specified (see Figure 3). If the web service is protected and roles have been defined, you will have the option to indicate what roles are authorized to the web service. If roles have not been defined, then all authenticated users are allowed access to the web service.

Figure 3. Deploy web service – step 3



WSERVICE > Manage Deployed Services > Deploy New Service

Deploy New Service

Specify security constraint - Step 3 of 8

The security constraint limits who can access the web service.

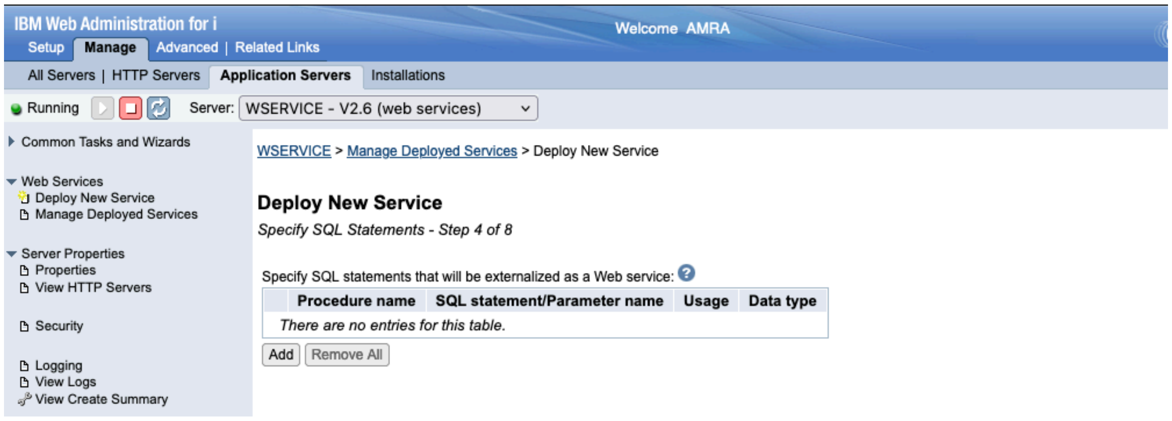
Secure transport required:	No
Protect using authentication method:	*NONE

The security constraint panel is beyond the scope of this article. We accept the default values and click on the **Next** button at bottom of form.

Step 3-4. Specify SQL statements

The wizard shows a page that allows you to add SQL statements (see Figure 4). SQL statements are associated with a procedure so you also have to specify a meaningful procedure name (because it will be used as an identifier when returning result sets in response to a client request).

Figure 4. Deploy web service – step 4


IBM Web Administration for i
Welcome AMRA
Setup | Manage | Advanced | Related Links
All Servers | HTTP Servers | Application Servers | Installations
Running
Server: WSERVICE - V2.6 (web services)
Common Tasks and Wizards
Web Services
Deploy New Service
Manage Deployed Services
Server Properties
Properties
View HTTP Servers
Security
Logging
View Logs
View Create Summary
WSERVICE > Manage Deployed Services > Deploy New Service

Deploy New Service

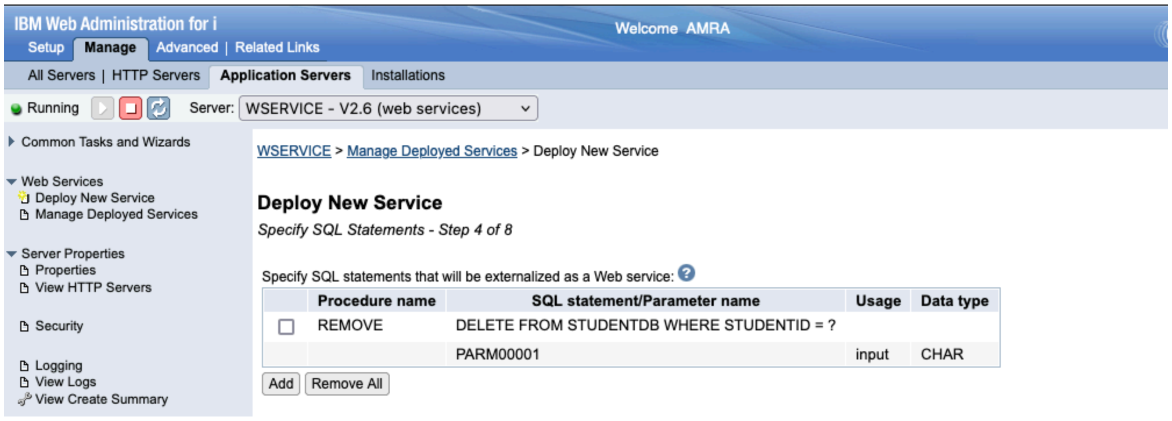
Specify SQL Statements - Step 4 of 8

Specify SQL statements that will be externalized as a Web service:

Procedure name	SQL statement/Parameter name	Usage	Data type
There are no entries for this table.			

Click **Add** to add a SQL statement. Figure 5 shows the page after we added the SQL statement for removing a student record.

Figure 5. Deploy web service – step 4 (remove student record)


IBM Web Administration for i
Welcome AMRA
Setup | Manage | Advanced | Related Links
All Servers | HTTP Servers | Application Servers | Installations
Running
Server: WSERVICE - V2.6 (web services)
Common Tasks and Wizards
Web Services
Deploy New Service
Manage Deployed Services
Server Properties
Properties
View HTTP Servers
Security
Logging
View Logs
View Create Summary
WSERVICE > Manage Deployed Services > Deploy New Service

Deploy New Service

Specify SQL Statements - Step 4 of 8

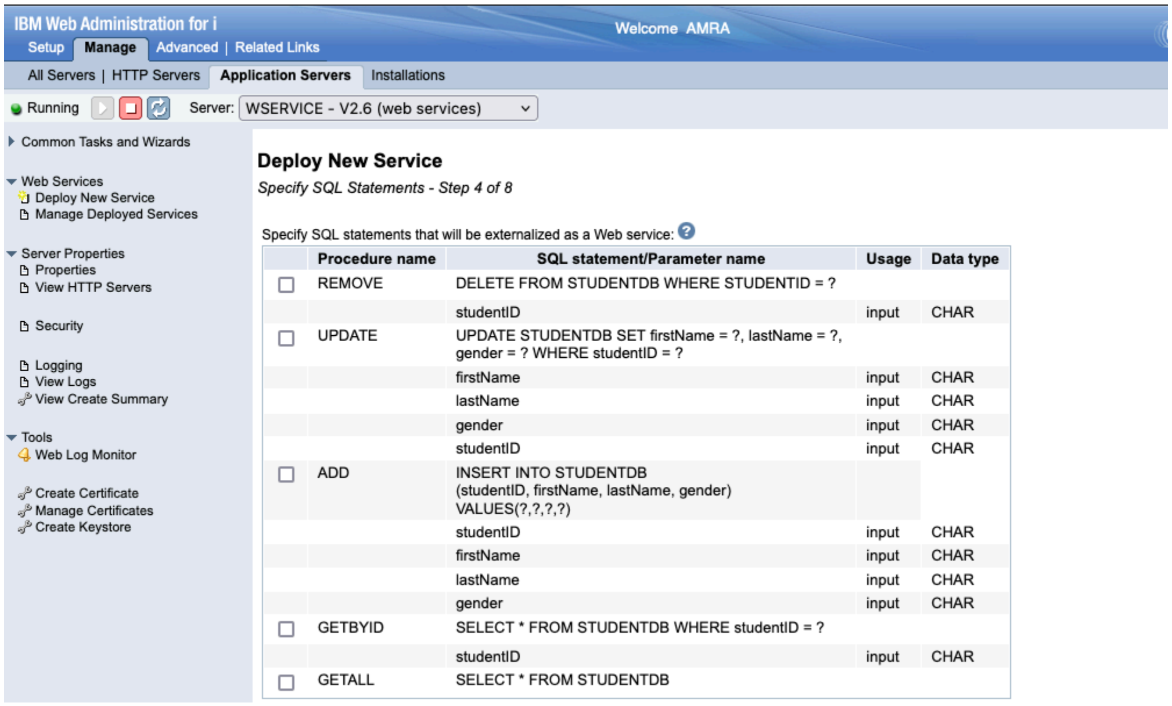
Specify SQL statements that will be externalized as a Web service:

	Procedure name	SQL statement/Parameter name	Usage	Data type
☐	REMOVE	DELETE FROM STUDENTDB WHERE STUDENTID = ?		
		PARM00001	input	CHAR

In Figure 5, we specify `REMOVE` for procedure name and the SQL statement to be used to remove a record from the database. Notice that we use a parameter marker for student ID that will be removed. When you click **Continue** after adding the SQL statement, the page will display a parameter corresponding to the parameter marker. You can (and should) change the identifier associated with the parameter by selecting the SQL statement. You want to change the parameter identifier for parameters if the identifier is going to be part of the HTTP payload of the client request (payloads are normally associated with the POST or PUT HTTP methods).

In Figure 6, you can see the rest of the SQL statements. Note that the parameter identifiers have been changed for all the SQL statements.

Figure 6. Deploy web service – step 4 (all SQL statements added)



Server: WSERVICE - V2.6 (web services)

Deploy New Service

Specify SQL Statements - Step 4 of 8

Specify SQL statements that will be externalized as a Web service:

	Procedure name	SQL statement/Parameter name	Usage	Data type
☐	REMOVE	DELETE FROM STUDENTDB WHERE STUDENTID = ?		
		studentID	input	CHAR
☐	UPDATE	UPDATE STUDENTDB SET firstName = ?, lastName = ?, gender = ? WHERE studentID = ?		
		firstName	input	CHAR
		lastName	input	CHAR
		gender	input	CHAR
		studentID	input	CHAR
☐	ADD	INSERT INTO STUDENTDB (studentID, firstName, lastName, gender) VALUES(?,?,?,?)		
		studentID	input	CHAR
		firstName	input	CHAR
		lastName	input	CHAR
		gender	input	CHAR
☐	GETBYID	SELECT * FROM STUDENTDB WHERE studentID = ?		
		studentID	input	CHAR
☐	GETALL	SELECT * FROM STUDENTDB		

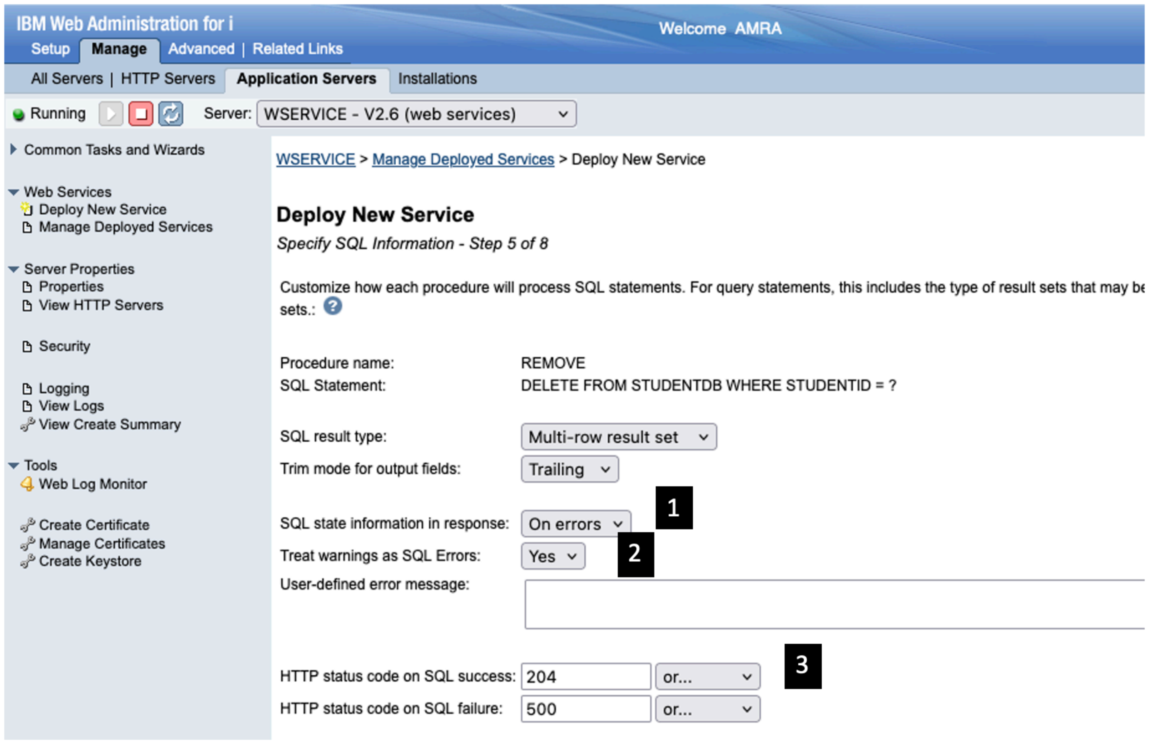
After adding all the SQL statements, click **Next**.

Step 3-5. Specify SQL information

Procedures contain SQL statements. For each procedure, you must specify how the output from SQL statements is to be handled.

The first procedure to be processed is `REMOVE`.

Figure 7. Deploy web service – step 5 (REMOVE)



Server: WSERVICE - V2.6 (web services)

WSERVICE > Manage Deployed Services > Deploy New Service

Deploy New Service

Specify SQL Information - Step 5 of 8

Customize how each procedure will process SQL statements. For query statements, this includes the type of result sets that may be
sets.:

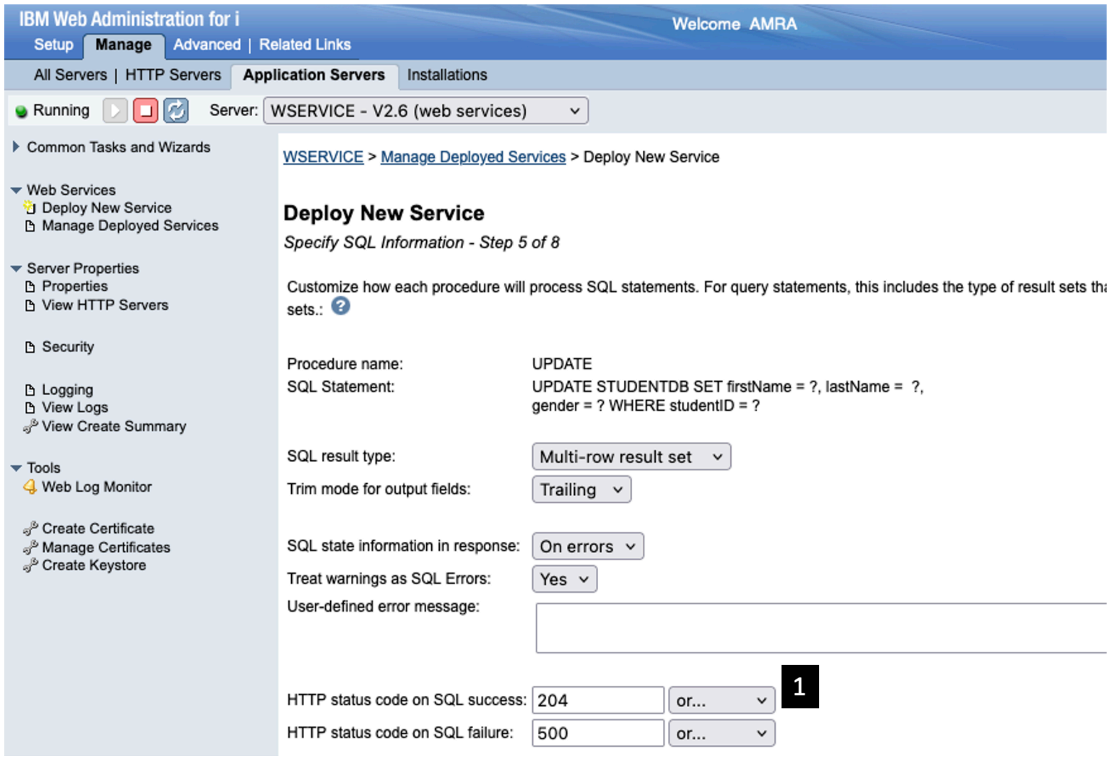
Procedure name:	REMOVE
SQL Statement:	DELETE FROM STUDENTDB WHERE STUDENTID = ?
SQL result type:	Multi-row result set
Trim mode for output fields:	Trailing
SQL state information in response:	On errors
Treat warnings as SQL Errors:	Yes
User-defined error message:	
HTTP status code on SQL success:	204 or...
HTTP status code on SQL failure:	500 or...

Looking at Figure 7, you can find that:

- We have taken the default value for including SQL state information in response (**1**), which is only if the SQL operation fails to run successfully.
- We have also taken the default value for treating SQL warnings as errors (**2**). This option only applies to SQL warnings related to getting DB connections, preparing SQL statements, and running the SQL statements. If a warning happens in any of these cases, it will be treated as a SQL error.
- We have indicated that a 204 (No Content) HTTP status code (**3**) is to be returned on a SQL statement that has run successfully. We have taken the default HTTP status code of 500 (Server Error) if the SQL operation fails to run successfully.

Click **Next** to process the `UPDATE` procedure (shown in Figure 8).

Figure 8. Deploy web service – step 5 (UPDATE)



Deploy New Service

Specify SQL Information - Step 5 of 8

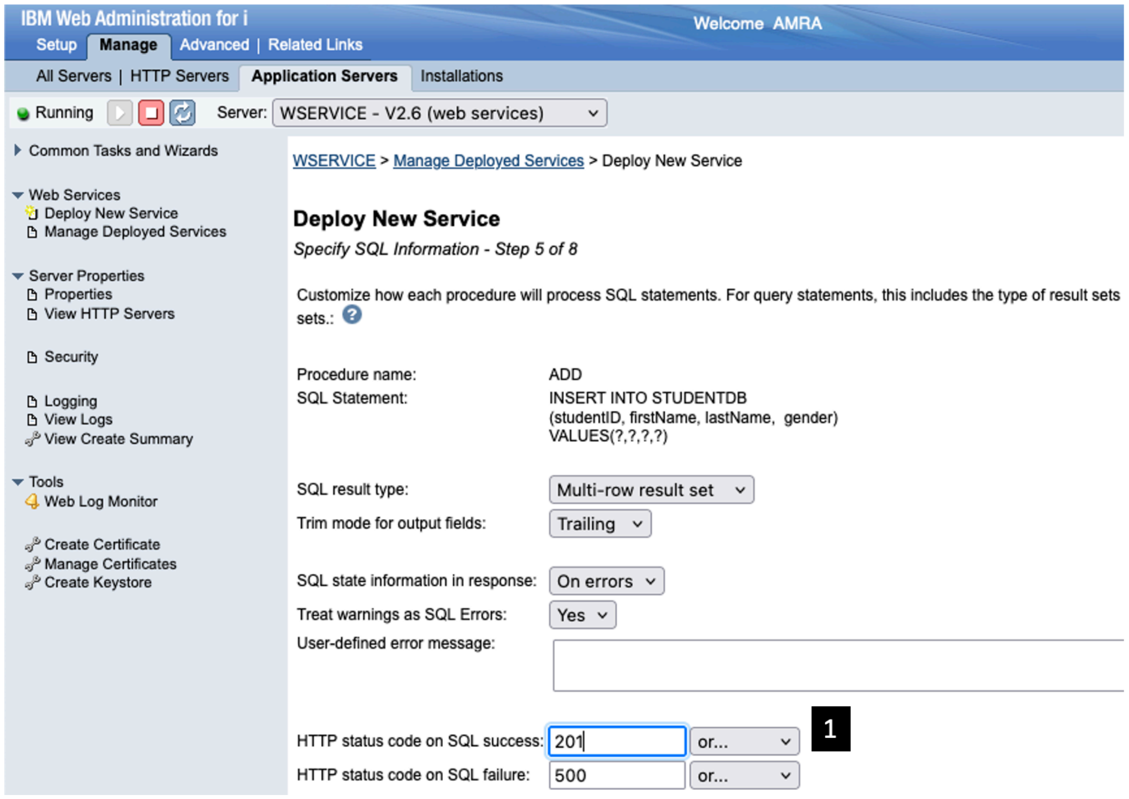
Customize how each procedure will process SQL statements. For query statements, this includes the type of result sets th
sets.:

Procedure name:	UPDATE
SQL Statement:	UPDATE STUDENTDB SET firstName = ?, lastName = ?, gender = ? WHERE studentID = ?
SQL result type:	Multi-row result set
Trim mode for output fields:	Trailing
SQL state information in response:	On errors
Treat warnings as SQL Errors:	Yes
User-defined error message:	
HTTP status code on SQL success:	204 or...
HTTP status code on SQL failure:	500 or...

Looking at Figure 8, you will find that we have taken the default values for the fields except for where we have indicated that a 204 (No Content) HTTP status code (**1**) is to be returned on a SQL statement that has run successfully.

Click **Next** to process the `ADD` procedure (Figure 9).

Figure 9. Deploy web service – step 5 (ADD)



Deploy New Service

Specify SQL Information - Step 5 of 8

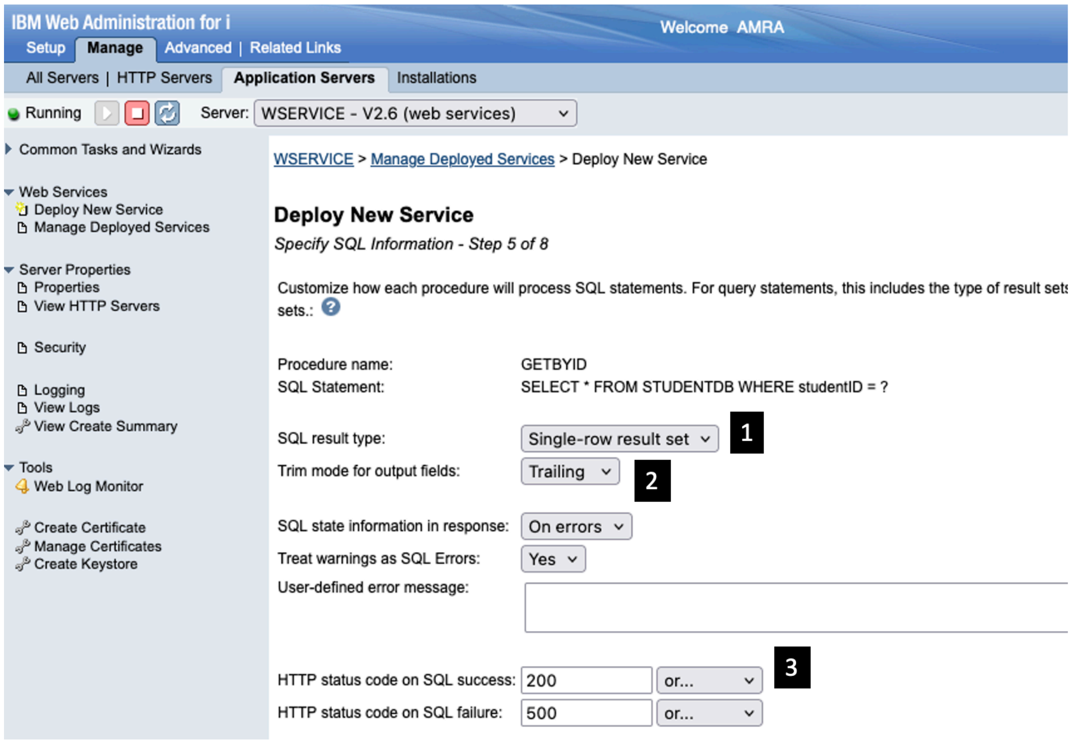
Customize how each procedure will process SQL statements. For query statements, this includes the type of result sets sets.:

Procedure name:	ADD
SQL Statement:	INSERT INTO STUDENTDB (studentID, firstName, lastName, gender) VALUES(?,?,?,?)
SQL result type:	Multi-row result set
Trim mode for output fields:	Trailing
SQL state information in response:	On errors
Treat warnings as SQL Errors:	Yes
User-defined error message:	
HTTP status code on SQL success:	201 or...
HTTP status code on SQL failure:	500 or...

Looking at Figure 9, you will find that we have taken the default values for the fields except for where we have indicated that a 201 (Created) HTTP status code (**1**) is to be returned on a SQL statement that has run successfully.

Click **Next** to process the `GETBYID` procedure (Figure 10).

Figure 10. Deploy web service – step 5 (GETBYID)



Deploy New Service

Specify SQL Information - Step 5 of 8

Customize how each procedure will process SQL statements. For query statements, this includes the type of result sets sets.:

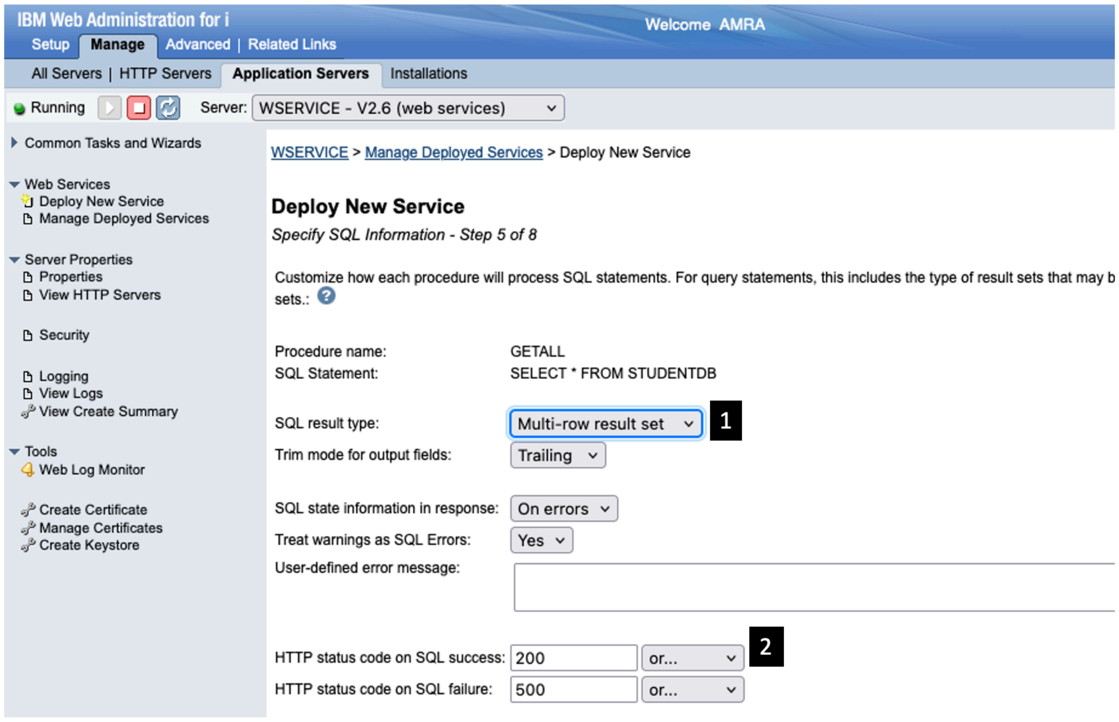
Procedure name:	GETBYID
SQL Statement:	SELECT * FROM STUDENTDB WHERE studentID = ?
SQL result type:	Single-row result set
Trim mode for output fields:	Trailing
SQL state information in response:	On errors
Treat warnings as SQL Errors:	Yes
User-defined error message:	
HTTP status code on SQL success:	200 or...
HTTP status code on SQL failure:	500 or...

Looking at Figure 10, you can find that:

- The SQL operation returns a single-row result set (**1**). This ensures that the response returned is not an array of objects, but a single object. If you wanted the response to return an array of objects, you would specify a multi-row result set. Note that if you specify a single-row result set, then only one row will be returned, even in the case where the result set has multiple rows.
- The default trim mode (**2**) for output character fields is to remove trailing blanks from fields. You also have the option to remove leading blanks, both leading and trailing blanks, and to not remove blanks. Using the trim function within the SQL statement will perform better than setting this option to a value other than `None` (not to remove blanks).
- We have taken the default values for HTTP status codes (**3**): 200 (OK) for success and 500 (Server Error) for failure.

Click **Next** to process the `GETALL` procedure (Figure 11).

Figure 11. Deploy web service – step 5 (GETALL)



Deploy New Service

Specify SQL Information - Step 5 of 8

Customize how each procedure will process SQL statements. For query statements, this includes the type of result sets that may b sets.:

Procedure name: GETALL
SQL Statement: SELECT * FROM STUDENTDB

SQL result type: Multi-row result set
Trim mode for output fields: Trailing

SQL state information in response: On errors
Treat warnings as SQL Errors: Yes
User-defined error message:

HTTP status code on SQL success: 200 or...
HTTP status code on SQL failure: 500 or...

Looking at Figure 11, you can find that:

- The SQL operation returns a multi-row result set (**1**). This ensures that the response returned is an array of objects.
- We have taken the default values for HTTP status codes (**2**): 200 (OK) for success and 500 (Server Error) for failure.

At this point, we are done with setting SQL-related information for each procedure. Click **Next**.

Step 3-6. Specify resource method information

Before discussing this step, it is a good idea to summarize the REST information for the RESTful application that is to be deployed. Table 4 summarizes REST information for each of the resource methods (that is, procedures) of the SRA.

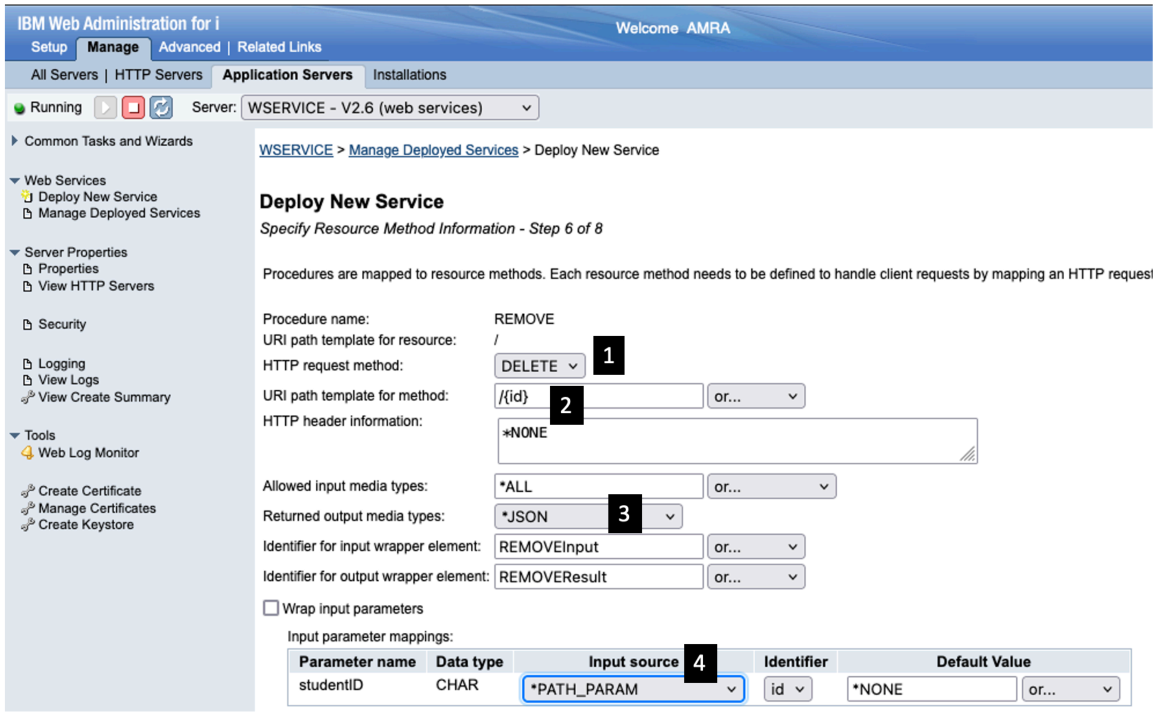
Table 4. REST information for each procedure

REMOVE	**URL**	/*context-root*/students/*{id}*
	Method	DELETE
	Request body	None
	Returns	204 No content
		500 Server error
UPDATE	**URL**	/*context-root*/students
	Method	PUT
	Request body	JSON
	Returns	204 No content

		500 Server error
ADD	**URL**	/*context-root*/students
	Method	POST
	Request body	JSON
	Returns	201 Created
		500 Server error
GETBYID	**URL**	/*context-root*/students/*{id}*
	Method	GET
	Request body	None
	Returns	200 OK and JSON
		500 Server error
GETALL	**URL**	/*context-root*/students
	Method	GET
	Request body	None
	Returns	200 OK and JSON
		500 server error

First procedure to be processed is the `REMOVE` procedure.

Figure 12. Deploy web service – step 6 (REMOVE)



Deploy New Service

Specify Resource Method Information - Step 6 of 8

Procedures are mapped to resource methods. Each resource method needs to be defined to handle client requests by mapping an HTTP request

Procedure name:	REMOVE
URI path template for resource:	/
HTTP request method:	DELETE
URI path template for method:	/{id}
HTTP header information:	*NONE
Allowed input media types:	*ALL
Returned output media types:	*JSON
Identifier for input wrapper element:	REMOVEInput
Identifier for output wrapper element:	REMOVEResult

☐ Wrap input parameters

Input parameter mappings:

Parameter name	Data type	Input source	Identifier	Default Value
studentID	CHAR	*PATH_PARAM	id	*NONE

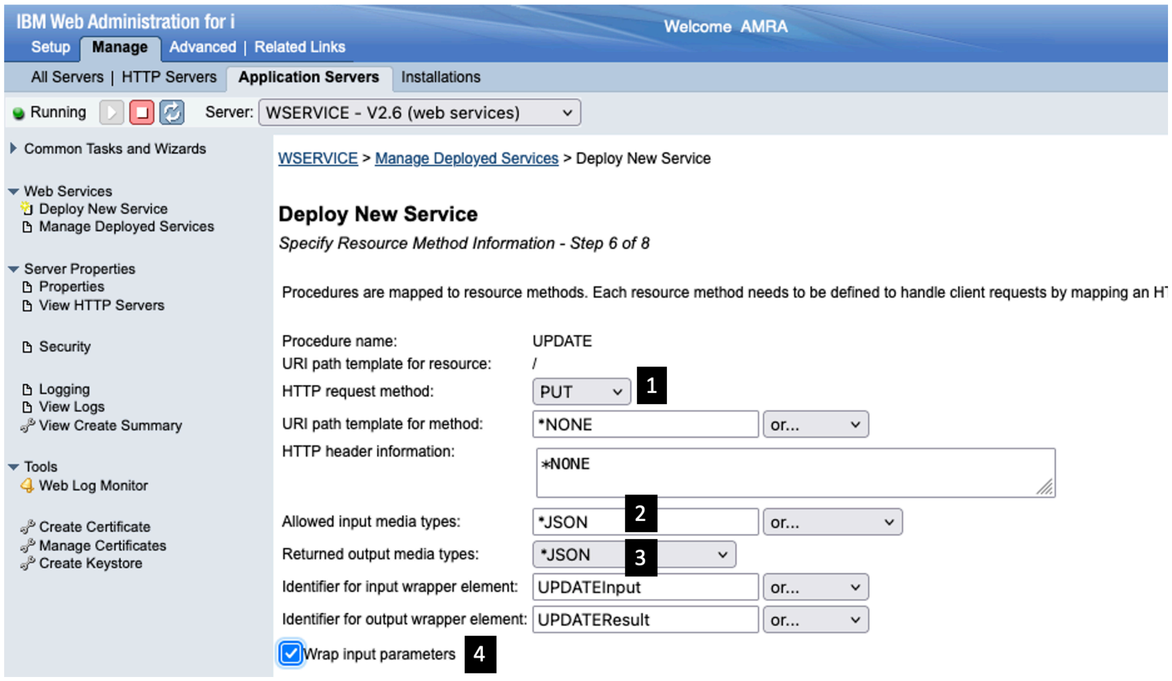
Looking at Figure 12, you can find that:

- The HTTP request method (**1**) is set to DELETE.
- Recall from Table 3 that the URI has the following format:
  /*context-root*/students/*{id}*
  That is, the student identifier information is passed in as part of the URI. So, we specify a URI path template (**2**) so that any HTTP DELETE request that matches the URI is passed to the `REMOVE` procedure.

- There is no entity body returned by the procedure. However, we needed to specify a value and therefore, retained the default which is JSON (**3**).
- We want to inject the URI path variable `id` into the `studentID` parameter. We do this by specifying the input source as `*PATH_PARAM` (**4**) and selecting the identifier to be inserted.

Click **Next** to process the `UPDATE` procedure (Figure 13).

Figure 13. Deploy web service – step 6 (UPDATE)



WSERVICE > Manage Deployed Services > Deploy New Service

Deploy New Service

Specify Resource Method Information - Step 6 of 8

Procedures are mapped to resource methods. Each resource method needs to be defined to handle client requests by mapping an H

Procedure name:	UPDATE
URI path template for resource:	/
HTTP request method:	PUT
URI path template for method:	*NONE
HTTP header information:	*NONE
Allowed input media types:	*JSON
Returned output media types:	*JSON
Identifier for input wrapper element:	UPDATEInput
Identifier for output wrapper element:	UPDATEResult

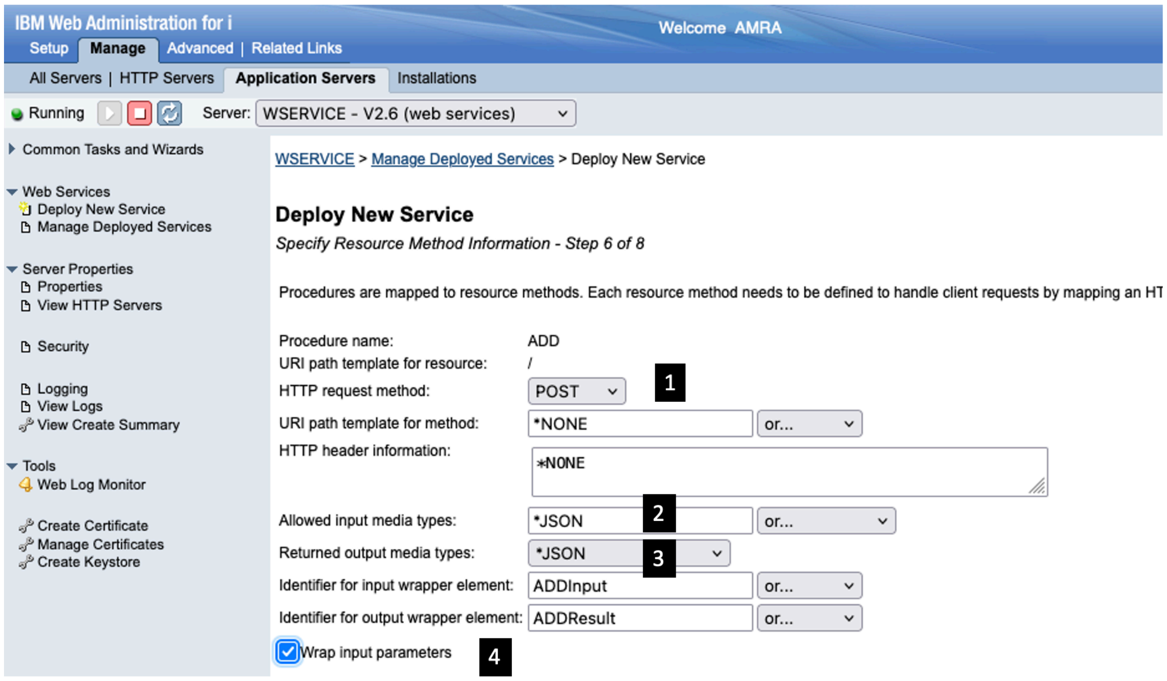
[✓] Wrap input parameters

Looking at Figure 13, you can find that:

- The HTTP request method (**1**) is set to PUT.
- The format of the input data is JSON (**2**).
- There is no entity body returned by the procedure. However, we need to specify a value, and therefore, we retained the default which is JSON (**3**).
- Because the request is in the payload of the client request, we specify that the parameters should be wrapped (**4**).

Click **Next** to process the `CREATE` procedure (Figure 14).

Figure 14. Deploy web service – step 6 (ADD)



WSERVICE > Manage Deployed Services > Deploy New Service

Deploy New Service

Specify Resource Method Information - Step 6 of 8

Procedures are mapped to resource methods. Each resource method needs to be defined to handle client requests by mapping an HT

Procedure name:	ADD	
URI path template for resource:	/	
HTTP request method:	POST	
URI path template for method:	*NONE	or...
HTTP header information:	*NONE	
Allowed input media types:	*JSON	or...
Returned output media types:	*JSON	
Identifier for input wrapper element:	ADDInput	or...
Identifier for output wrapper element:	ADDResult	or...

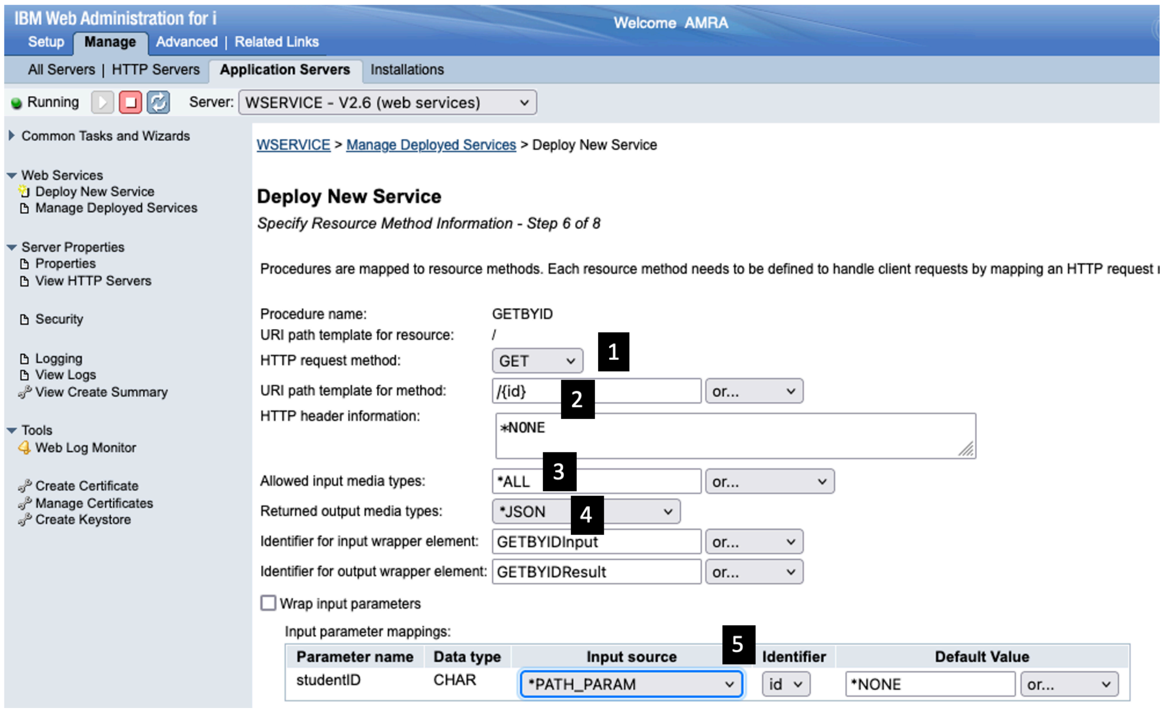
[✓] Wrap input parameters

Looking at Figure 14, you can find that:

- The HTTP request method (**1**) is set to POST.
- The format of the input data is JSON (**2**).
- There is no entity body returned by the procedure. However, we need to specify some value, and therefore, we retained the default which is JSON (**3**).
- Because the request is in the payload of the client request, we specify that the parameters should be wrapped (**4**).

Click **Next** to process the `GETBYID` procedure (Figure 15).

Figure 15. Deploy web service – step 6 (GETBYID)



Deploy New Service

Specify Resource Method Information - Step 6 of 8

Procedures are mapped to resource methods. Each resource method needs to be defined to handle client requests by mapping an HTTP request

Procedure name:	GETBYID
URI path template for resource:	/
HTTP request method:	GET
URI path template for method:	/{id}
HTTP header information:	*NONE
Allowed input media types:	*ALL
Returned output media types:	*JSON
Identifier for input wrapper element:	GETBYIDInput
Identifier for output wrapper element:	GETBYIDResult

☐ Wrap input parameters

Input parameter mappings:

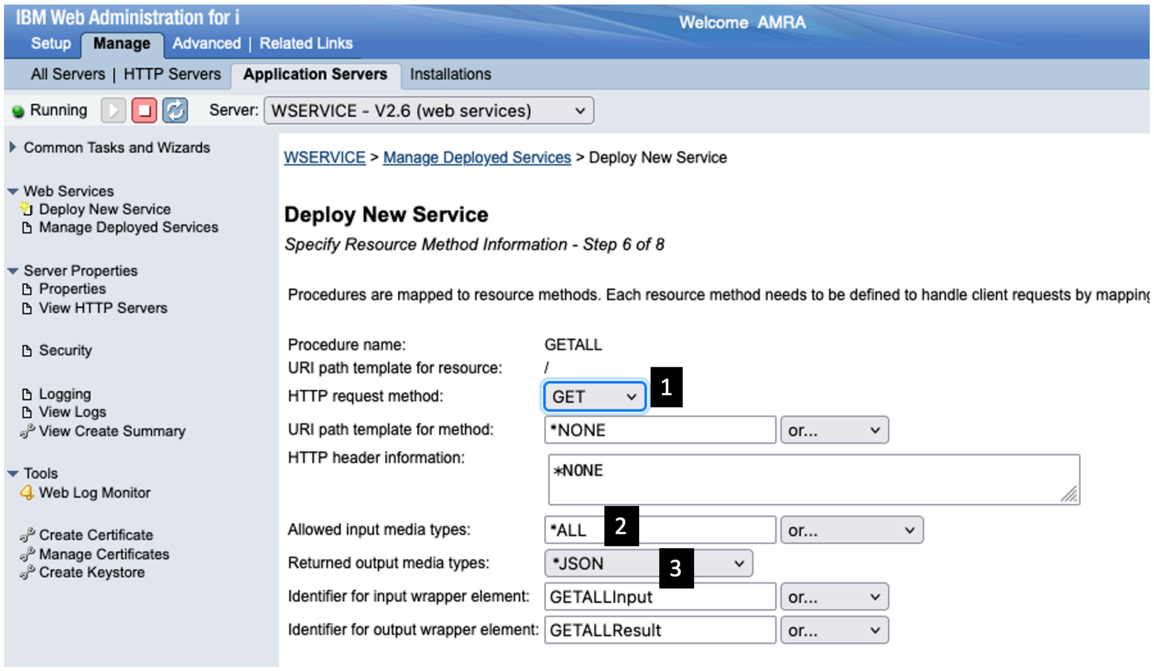
Parameter name	Data type	Input source	Identifier	Default Value
studentID	CHAR	*PATH_PARAM	id	*NONE

Looking at Figure 15, you can find that:

- The HTTP request method (**1**) is set to GET.
- Recall from Table 3 that the URI has the following format:
  /*context-root*/students/*{id}*
  That is, the student identifier information is passed in as part of the URI. So, we specify a URI path template (**2**) so that any HTTP GET request that matches the URI is passed to the `GETBYID` procedure.
- There is no input data. So, we can simply consider the default values to accept all input media types (**3**).
- The format of the output data is JSON (**4**).
- We want to inject the URI path variable `id` into the `studentID` parameter. We do this by specifying the input source as `*PATH_PARAM` (**5**) and selecting the identifier to be inserted.

Click **Next** to process the `GETALL` procedure (Figure 16).

Figure 16. Deploy web service – step 6 (GETALL)



Deploy New Service

Specify Resource Method Information - Step 6 of 8

Procedures are mapped to resource methods. Each resource method needs to be defined to handle client requests by mappin

Procedure name:	GETALL	
URI path template for resource:	/	
HTTP request method:	GET	
URI path template for method:	*NONE	or...
HTTP header information:	*NONE	
Allowed input media types:	*ALL	or...
Returned output media types:	*JSON	
Identifier for input wrapper element:	GETALLInput	or...
Identifier for output wrapper element:	GETALLResult	or...

Looking at Figure 16, you can find that:

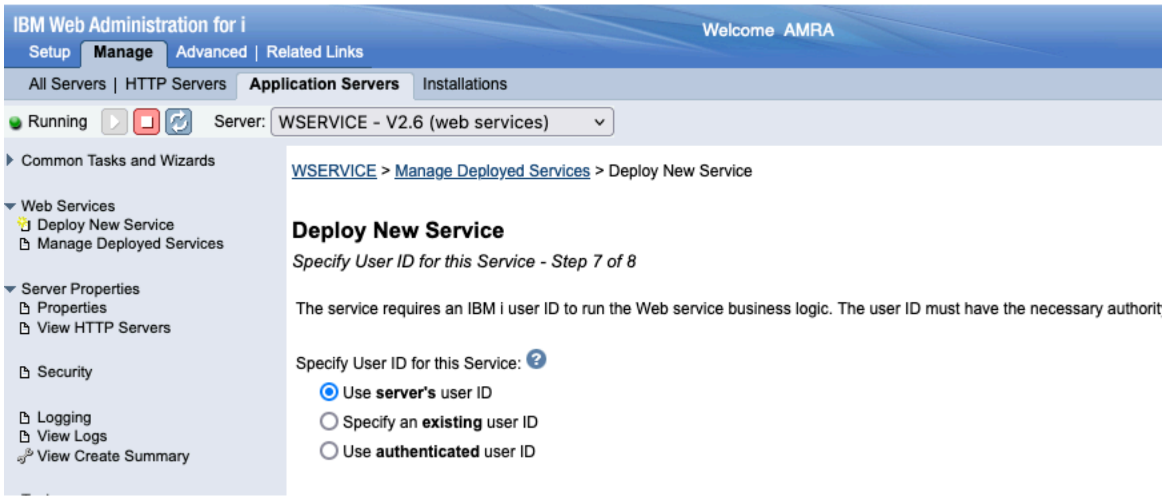
- The HTTP request method (**1**) is set to GET.
- There is no input data. So, we can simply consider the default values to accept all input media types (**2**).
- The format of the output data is JSON (**3**).

At this point, we have completed setting REST information. Click **Next**.

Step 3-7. Specify user ID for this service

We now need to specify the user ID to run the service. As shown in Figure 17, you can run the service using the server's user ID, specifying an existing user ID, or using an authenticated user ID (this would require you to enable basic authentication – see the Security chapter in the Integrated Web Services Server Administration and Programming Guide for details on how to do this).

Figure 17. Deploy web service – step 7



WSERVICE > Manage Deployed Services > Deploy New Service

Deploy New Service

Specify User ID for this Service - Step 7 of 8

The service requires an IBM i user ID to run the Web service business logic. The user ID must have the necessary authorit

Specify User ID for this Service:

- [✓ Use **server's** user ID]
- Specify an **existing** user ID
- Use **authenticated** user ID

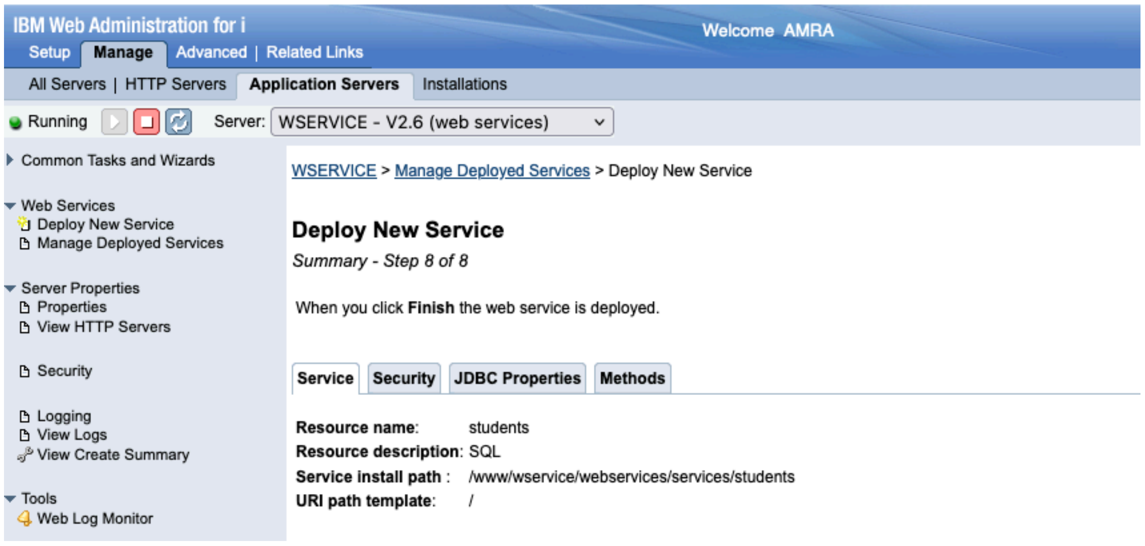
For the web service to run correctly, the user ID status must be set to *ENABLED and the password must be set to a value other than *NONE. If a user ID that is disabled or has a password of *NONE is specified, a warning message is displayed, and the service may not run correctly. In addition, ensure that the specified user ID has the proper authorities to any resources and objects that the web service needs, such as libraries, databases, and files.

In this example, we accept the default values. Click **Next**.

Step 3-8. Deploy web service – step 8

The web service deployment wizard shows you a summary page (see Figure 18), giving you a chance to see the details relating to the web service being deployed.

Figure 18. Deploy web service – step 8 (Summary)



WSERVICE > Manage Deployed Services > Deploy New Service

Deploy New Service

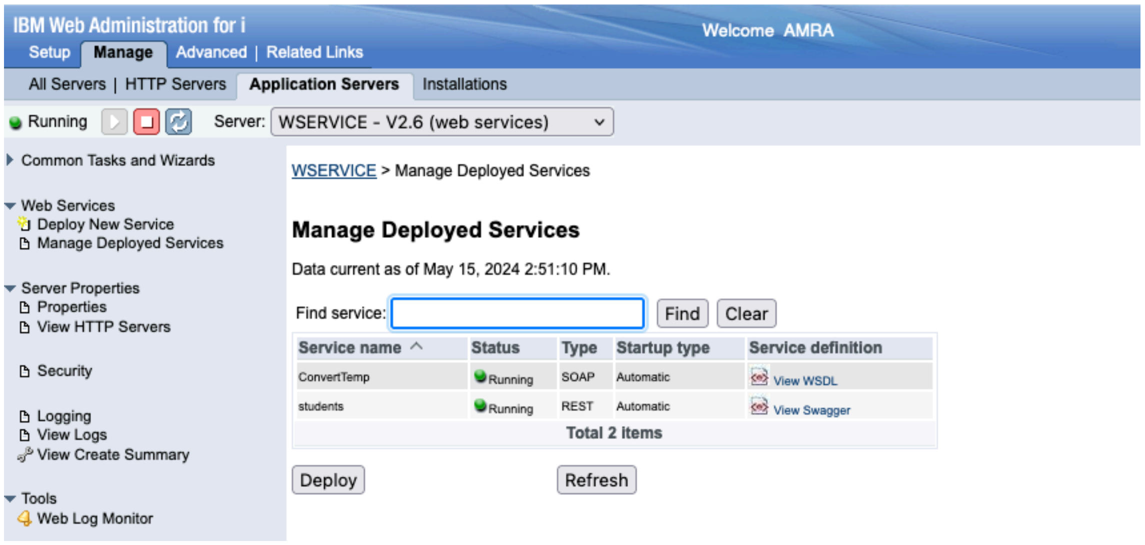
Summary - Step 8 of 8

When you click **Finish** the web service is deployed.

Resource name:	students
Resource description:	SQL
Service install path :	/www/wservice/webservices/services/students
URI path template:	/

Click **Finish** at the bottom of the summary page to complete the installation process. When the web service is deployed, the deployed service becomes active (indicated with a green dot to the left of service name) as in Figure 19:

Figure 19. Successfully deployed RESTful web service



WSERVICE > Manage Deployed Services

Manage Deployed Services

Data current as of May 15, 2024 2:51:10 PM.

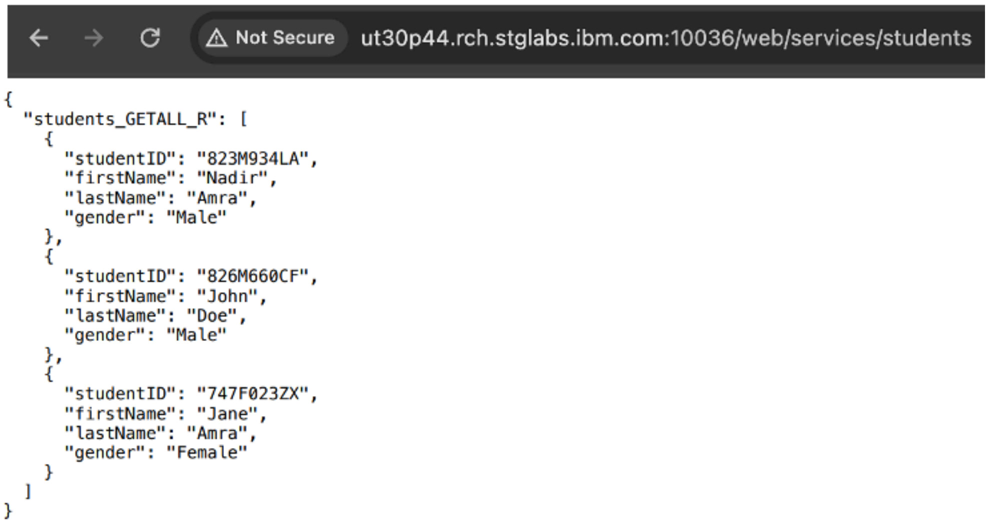
Service name	Status	Type	Startup type	Service definition
ConvertTemp	Running	SOAP	Automatic	View WSDL
students	Running	REST	Automatic	View Swagger

Total 2 items

Congratulations, you have now successfully deployed SQL statements as a RESTful web service.

You can easily test the resource methods that are bound to the HTTP GET method using a browser. Figure 20 shows the results of the request that returns all registered students.

Figure 20. Testing web service – return all registered students

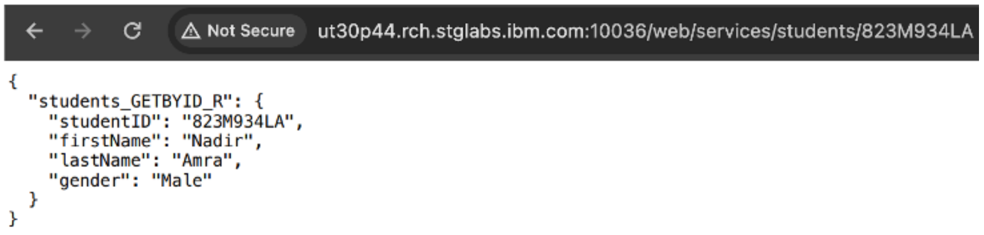


ut30p44.rch.stglabs.ibm.com:10036/web/services/students

```
{
  "students_GETALL_R": [
    {
      "studentID": "823M934LA",
      "firstName": "Nadir",
      "lastName": "Amra",
      "gender": "Male"
    },
    {
      "studentID": "826M660CF",
      "firstName": "John",
      "lastName": "Doe",
      "gender": "Male"
    },
    {
      "studentID": "747F023ZX",
      "firstName": "Jane",
      "lastName": "Amra",
      "gender": "Female"
    }
  ]
}
```

Figure 21 shows the result of a request for a student record with a student ID of 823M934LA.

Figure 21. Testing web service – return a registered student

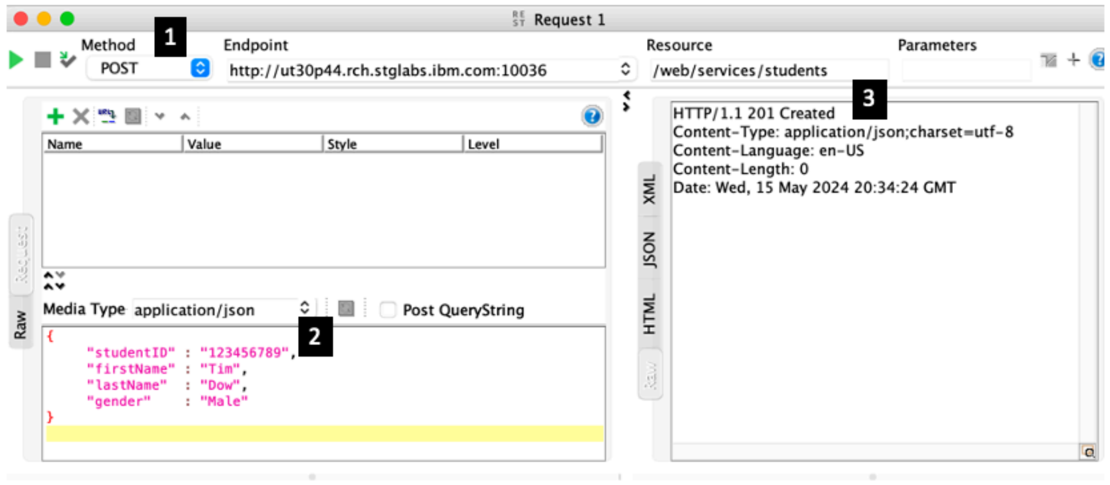


Not Secure ut30p44.rch.stglabs.ibm.com:10036/web/services/students/823M934LA

```
{
  "students_GETBYID_R": {
    "studentID": "823M934LA",
    "firstName": "Nadir",
    "lastName": "Amra",
    "gender": "Male"
  }
}
```

To test the other resource methods, an external tool (such as SoapUI) must be used. Figure 22 shows the result of a request to create a new student registration using SoapUI.

Figure 22. Testing web service – create a new student registration



Method: POST
Endpoint: http://ut30p44.rch.stglabs.ibm.com:10036
Resource: /web/services/students
Media Type: application/json

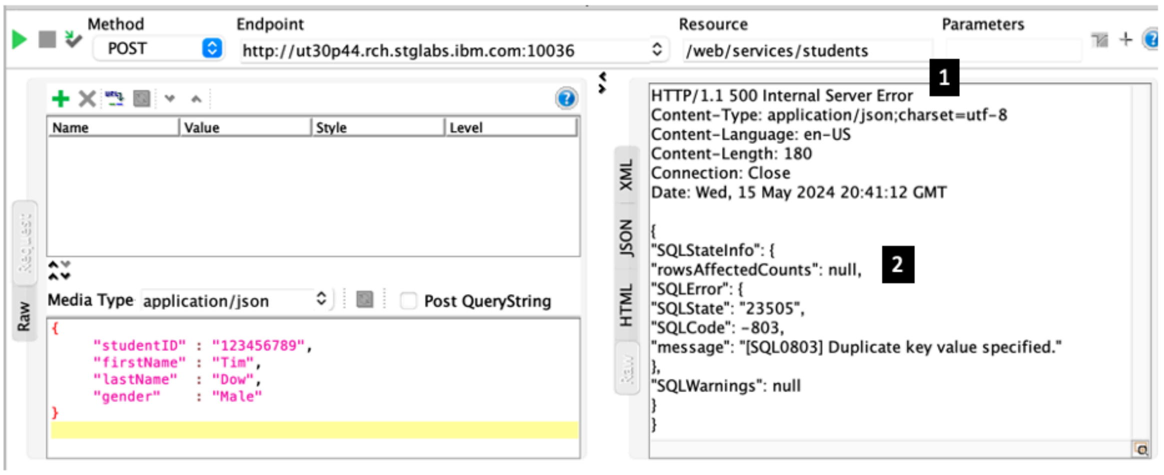
```
{
    "studentID" : "123456789",
    "firstName" : "Tim",
    "lastName"  : "Dow",
    "gender"    : "Male"
}
```

```
HTTP/1.1 201 Created
Content-Type: application/json;charset=utf-8
Content-Language: en-US
Content-Length: 0
Date: Wed, 15 May 2024 20:34:24 GMT
```

Looking at Figure 22, as we are attempting to create a new student registration, the HTTP method is POST (**1**). The sub pane numbered (**2**) is the new student registration data in JSON format that will be sent to the server as part of the HTTP POST request. After submitting the request, the server response did not return any JSON data (**3**). Because the create request succeeded, the REST service returned the HTTP status code of 201 (Created).

Figure 23 shows the results when we ran the same request again.

Figure 23. Testing web service – create a student registration error



```
{
    "studentID" : "123456789",
    "firstName" : "Tim",
    "lastName"  : "Dow",
    "gender"    : "Male"
}
```

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json;charset=utf-8
Content-Language: en-US
Content-Length: 180
Connection: Close
Date: Wed, 15 May 2024 20:41:12 GMT

{
"SQLStateInfo": {
"rowsAffectedCounts": null,
"SQLError": {
"SQLState": "23505",
"SQLCode": -803,
"message": "[SQL0803] Duplicate key value specified."
},
"SQLWarnings": null
}
}
```

Looking at Figure 23, as we are attempting to create a new student registration with a student ID that already exists in the database, the server returned an HTTP status code of 500 (Server Error) (**1**) with information regarding the error (**2**), which includes the SQL state, SQL code, and the error message.

Summary

In this tutorial, you learned how to deploy a REST API based on SQL statements using the integrated web services server support for IBM i.

The integrated web services server REST support provides a solid foundation for creating and deploying REST APIs based on ILE programs, service programs, and SQL statements on the IBM i platform. Add the highly intuitive IBM Web Administration for i GUI for deploying web services, and you've got everything you need to quickly prototype and deploy your own custom REST API.

Resources

- For everything about the integrated web services support on IBM i see the product web page.