**Practical Guide To Create REST API using Node.js and MS-Sql- Part II**

<https://jinalshah999.medium.com/practical-guide-to-create-rest-api-using-node-js-and-ms-sql-part-ii-fa7eae2a153b>

**To download Microsoft SQL Server Express edition:**

<https://www.microsoft.com/en-us/download/details.aspx?id=55994>

**To download SQL Server Management Studio:**

<https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>

**CRUD Operation using MsSql Database:**

First, let’s create a Database as “**Demo**” using SQL Server Management Studio and then create a **Product table** like below:

Graphical user interface, application, table

Description automatically generated

**Product Table:**

Now back to our node js application, we will require some packages which can connect us to the database.

So using the terminal, install **MsSql** package and **Express** package from npm like this:

**npm install mssqlnpm install express**

After installing you can also check in your **package.json** file that it has added dependencies for your packages.

Now let’s make one class as **dbconnection.js** where we will have configurations of our database like this:

Here, we have created an object for mssql package and it has a **connect()** method in which you can pass parameters of your database name, password, port number, etc.

Now create a folder “**modules**” at the root level of the app and inside this folder also create another folder as “**product**” for product-related files.

Now inside this “**product**” folder create some new files as **product.controller.js, product.js, and product.mssql.js**:

Graphical user interface, text, application

Description automatically generated

**product.mssql.js**

Here, we have added a reference of our dbconnection file and we have created a class as ProductMSSql and inside that, we are just sending a query to the database.

**product.js**

Here, we have added a reference of our product.mssql file and we have created a class as Product. Inside this class, we have a function that invokes the getAllProducts() method of productmssql class, and then it will send its response to the user.

**product.controller.js**

This Product controller will call the method of Product.js based on which type of HTTP request is made with its URL. Moreover, we have also added the ‘base path’ for the product module in app.use() method.

Now, we will create a file — **index.js** which is a module file, this will decide it should call which controller and when.

So inside this **module**folder, create a new file as **index.js**

In this file, we are initializing the local object of the application inside its constructor. Then using the init() method we will initialize our controller.

Now, it’s time to create an **app.js** file. Create a new file as app.js at the root level of the application as below:

Here in app.js, we are just defining our application-level settings.

This app.js will also have an init() method through which it can call the module file(index.js).

Now coming to the **index.js** file which is on the root level of our application, here we need some changes like this:

So finally run this app using

**node index.js**

command and we can see the below output:

Graphical user interface, text, application

Description automatically generated

**Summary diagram:**

Diagram

Description automatically generated

This was way too simple to understand. But, it is not something always used as per the industry standards. Industry always works with stored procedures or API. We will work with SPs. But, before that let us understand what stored procedure is actually?

**What is Stored Procedure?**

* Stored Procedures are saved SQL queries, so that code can be reused.
* Rather than writing the same code and queries, again and again, SPs just need a call and execution will be done easily.
* If required, then parameters are passed to the Stored Procedures.

**Using Stored Procedure :**

From now we will use Stored Procedures instead of SQL Query to do any operation with the database.

For this create some stored procedures like given below:

**Note**: SQL scripts are attached at the end of the article.

Graphical user interface, application

Description automatically generated

**Stored Procedure 1 : getAllProducts:**

Use Demo  
 GOSET ANSI\_NULLS ON  
 GO  
 SET QUOTED\_IDENTIFIER ON  
 GOCREATE PROCEDURE dbo.getAllProducts   
 AS  
 BEGIN  
 SELECT \* from product;  
 END  
 GO

**Stored Procedure 2 : addProduct:**

Use Demo  
 GOSET ANSI\_NULLS ON  
 GO  
 SET QUOTED\_IDENTIFIER ON  
 GOCREATE PROCEDURE dbo.addProduct  
 @product\_name varchar(50),  
 @product\_price numeric(18,2),  
 @product\_description varchar(250),  
 @product\_qty numeric(18,2)  
 AS  
 BEGIN  
 INSERT INTO product(product\_name,product\_price,product\_description,product\_qty)  
 VALUES(@product\_name,@product\_price,@product\_description,@product\_qty);  
 END  
 GO

**Stored Procedure 3 : deleteProduct:**

Use Demo  
 GOSET ANSI\_NULLS ON  
 GO  
 SET QUOTED\_IDENTIFIER ON  
 GOCREATE PROCEDURE dbo.deleteProduct  
 @product\_id int  
 AS  
 BEGIN  
 DELETE FROM product WHERE product\_id = @product\_id;  
 END  
 GO

**Stored Procedure 4 : updateProduct:**

Use Demo  
 GOSET ANSI\_NULLS ON  
 GO  
 SET QUOTED\_IDENTIFIER ON  
 GOCREATE PROCEDURE dbo.updateProduct  
 @product\_id int,  
 @product\_name varchar(50),  
 @product\_price numeric(18,2),  
 @product\_description varchar(250),  
 @product\_qty numeric(18,2)  
 AS  
 BEGIN  
 UPDATE product SET product\_name= @product\_name, product\_price=@product\_price,  
 product\_description=@product\_description, product\_qty=@product\_qty  
 WHERE product\_id = @product\_id;  
   
 END  
 GO

As you can see we have created simple stored procedures for Select, Update, Insert and Delete operations.

Now open your **product.mssql.js**and instead of the **query()** method we will use **execute()** method and pass the stored procedure’s name there.

Run the command **node index.js**and you can see the same output as previously got.

**Insert, Update, Delete Operation :**

Just like above we will use the stored procedure **addProduct**to insert the record in the databaseand for this, we need to pass the parameters like product name, price, etc using the **input()**method.

In the same way, we can also use other **updateProduct** and **deleteProduct**stored procedures:

Now on **Product.js,**we will call this above addProduct, updateProduct, and deleteProduct functions with their parameters.

Now next step is to define a **path**for every method in **product.controller.js.**

So this is how we can define our Routes. For Insert we used POST, for the update we used PUT and for delete, we used DELETE method above.

Now next step is we need to import**body-parser** in **app.js**.

Since we need to pass the data to the server using the request’s body, we have to import a package called body-parser.

For this open **package.json** and here add a **dependency** for body-parser like this:

"body-parser": "^1.17.1"

Now run the command **npm install** to download this package.

Okay, so now let’s import this body-parser in **app.js** file and we can also limit the body size of the request being made. The default limit is 4MB but we can define it like below also:

Now, run the command **node index.js**to run the app and check the above APIs using **Postman** like this:

**getAllProducts:**

Graphical user interface, text, application

Description automatically generated

**addProduct:**

Graphical user interface, text, application, email

Description automatically generated

**updateProduct:**

Graphical user interface, text, application, email

Description automatically generated

**deleteProduct:**

Graphical user interface, text, application, email

Description automatically generated

**Summary diagram**

Diagram

Description automatically generated