**tinyWebApi**

tiny Web Api supports almost any type of input and output and connects to Microsoft SQL Server/Oracle database (very much extendable to other relational databases) and allows teams to focus only on database and UI development without even having to make any change in the service. This service does not have any service specific logic and hence could be easily used within any project having Microsoft SQL Server, Oracle as backend database. This supports Get, Post, Put, Delete method automations which could be overridden for extension.

Once deployed this service does not need any future deployment apart from configuration changes.

**Get:** Using this one could execute any query and pass on any input parameter via the query params as part of URL.

**Post/Put/Delete:** Using this one could input/upload individual variables, single level array of single type objects, single level array of complex objects via request body supported y JSON (string, number, boolean, datetime, array of string, array of number, array of datetime, array of boolean and mix and match of all).

The API is capable of mapping out any input to the supported query or stored procedure and get the output back by mapping the input field names ***as it is*** with sql query or stored procedure.

Supported input type is JSON only but it could take in file content as a byte array of file type Excel, CSV and BLOB.

Excel (First sheet only) and CSV file content is uploaded as byte array within request body the filed name in which byte array is passed is shared with api along with file type and a flag stating file content is present in the request. Sheet name is also shared if specific sheet within excel is needed to be read else first sheet of excel will be read. Based on file type data is converted in to xml and shared with stored procedure/query as an xml. BLOB file type is also shared same way and passed on to query/stored procedure as varbinary type.

**Note:**

Multiple file upload at a time is supported provided fileContetFieldNames are separated by comma and all files are of same type.

Complex arrays are mapped using SQL table Types which need to follow ADO.Net/Microsoft SQL Server constraint of mapping the types in the order in which the field names are defined in the type. But browsers JSON serializers tend to serialize the objects and arrange properties alphabetically which hence the type definition in database to be done alphabetically.

For Oracle the UDT’s are supported from Oracle 21c onwards and hence before that the UDT’s could be mapped as XML or JSON and made available to the queries/stored procedures as XML/JSON serialized CLOBS.

This api supports **hot-reload** of the queries and hence restart of the service is not required. Queries are identified using unique key passed on with the calls.

**key (Mandatory parameter)**:key value helps top identity unique query to be executed from queries.json.

**executionType (Mandatory parameter):**

* ScalarText 🡺 Executes a query and returns the count of affected rows.
* NonQueryText 🡺 Executes a query and returns success of failure.
* ScalarProcedure 🡺Executes stored procedure and returns the count of affected rows.
* NonQueryProcedure 🡺 Executes a stored procedure and returns success of failure.
* DataTableText 🡺 Executes a query and returns the first result set.
* DataSetText 🡺Executes a query and returns multiple result set.
* DataTableProcedure 🡺 Executes a stored procedure and returns the first result set.
* DataSetProcedure 🡺 Executes a stored procedure and returns multiple result set.

**outPutType (Optional parameter and default is JSON):**

* JSON 🡺JSON output is given.
* EXCEL 🡺 Data is exported via EXCEL file having one or more result set.
* PDF 🡺Data is exported via PDF file having only one or more result set.
* CSV 🡺Data is exported via CSV file having only one result set.

**filecontentType (Optional, supported with Post/DELETE/PUT method and needed when file content is shared with request body, multiple field names should be comma separated but of same type):**

* EXCEL 🡺 Shared as Byte Array with Api and mapped with query/stored procedure as xml and only first sheet data or data of sheet name shared.
* CSV 🡺 Shared as Byte Array with Api and mapped with query/stored procedure as xml.
* BLOB 🡺 Shared as Byte Array with Api and mapped with query/stored procedure as VarBinary.

**sheetName (Optional, supported with Post method and passed on if specifie sheet content is to be read from Excel shared).**

**hasFileContent (Optional, set to be true that is 1 when file content is shared as byte array within request body, default is false).**

|  |  |
| --- | --- |
| **Web Method** | **Sample URL** |
| **Get** | [http://localhost:5000/api/Tiny/Get/{key}/{executionType}/{outPutType?}](http://localhost:5000/api/DynamicWebApi/Get/%7bkey%7d/%7bexecutionType%7d/%7boutPutType?%7d%20) |
| **Post** | [http://localhost:5000/api/Tiny/Post/{key}/{executionType}/{outPutType?}/{hasFileContent?}/{fileContentType?}/{fileContentFieldName?}/{sheetName?}](http://localhost:5000/api/DynamicWebApi/Post/%7bkey%7d/%7bexecutionType%7d/%7boutPutType?%7d/%7bhasFileContent?%7d/%7bfileContentType?%7d/%7bfileContentFieldName?%7d/%7bsheetName?%7d) |
| **Put** | [http://localhost:5000/api/Tiny/Put/{key}/{executionType}/{outPutType?}/{hasFileContent?}/{fileContentType?}/{fileContentFieldName?}/{sheetName?}](http://localhost:5000/api/TinyWebApi/Put/%7bkey%7d/%7bexecutionType%7d/%7boutPutType?%7d/%7bhasFileContent?%7d/%7bfileContentType?%7d/%7bfileContentFieldName?%7d/%7bsheetName?%7d) |
| **Delete** | [http://localhost:5000/api/Tiny/Delete/{key}/{executionType}/{outPutType?}/{hasFileContent?}/{fileContentType?}/{fileContentFieldName?}/{sheetName?}](http://localhost:5000/api/TinyWebApi/Delete/%7bkey%7d/%7bexecutionType%7d/%7boutPutType?%7d/%7bhasFileContent?%7d/%7bfileContentType?%7d/%7bfileContentFieldName?%7d/%7bsheetName?%7d) |

**appsettings.json sample:**

{

"AllowedHosts": "\*",

"AllowedCorsHosts": "http://\*:\*;http://localhost:4200",

"ConfigurationDirectoryPath": "",

"environment": "local",

"Logging": {

"LogLevel": {

"Default": "Trace",

"Microsoft": "Trace",

"Microsoft.Hosting.Lifetime": "Trace"

},

"options": {

"file": "Log\_$|Date[dd\_MMM\_yyyy]|$.log",

"size": 1073741824

}

}

}

**connectionstring.<environment>.json sample:**

{

"db1": {

"ConnectionString": "sql server connection string",

"IsEncrypted": false,

"IsImpersonationNeeded": true,

"RunAsUser": "systemuser1",

"EncryptionKey": "",

"ConnectionTimeOut": 1200,

"DatabaseType": "MSSQL"

},

"db2": {

"ConnectionString": "oracle connection string",

"IsEncrypted": true,

"IsImpersonationNeeded": false,

"RunAsUser": "",

"EncryptionKey": "some base 64 encryption key",

"ConnectionTimeOut": 1200,

"DatabaseType": "ORACLE"

}

}

**queries.<environment>.json sample:**

{

"GetEmployeeDetails\_Query": {

"Query": "select employeeid as eid, employeename as ename from dbo.employee",

"ExecutionType": "DataTableText",

"InputFieldNamesInSequence\_UDTDollarSeperatedByType": "",

"Outputs\_RefCursor\_InSequence\_ColonSeperated\_NotRequiredForMSSQL": "",

"Database": "db1",

"IsMapUDTAsJSON\_ApplicableForOracle": false,

"IsMapUDTAsXML\_ApplicableForOracle": false,

"ExternalDllPathImplementingIProcessDataInterface\_PreProcessing": "",

"ExternalDllNameImplementingIProcessDataInterface\_PreProcessing": "",

"FullyQualifiedNameOfClassImplementingInterfaceIProcessDataInterface\_PreProcessing": "",

"ExternalDllPathImplementingIProcessDataInterface\_PostProcessing": "",

"ExternalDllNameImplementingIProcessDataInterface\_PostProcessing": "",

"FullyQualifiedNameOfClassImplementingInterfaceIProcessDataInterface\_PostProcessing": "",

"IsSendOutputViaEmailAlso": false,

"Mailer": "somemailername1",

"IsAllowSendingJSONInMail": false,

"IsCachingRequired": false,

"CacheDurationInSeconds": 0

},

"GetEmployeeDetails\_StoredProcedure": {

"Query": "usp\_GetEmployeeDetails @EID = @EmployeeId",

"ExecutionType": "DataTableText",

"InputFieldNamesInSequence\_UDTDollarSeperatedByType": "",

"Outputs\_RefCursor\_InSequence\_ColonSeperated\_NotRequiredForMSSQL": "",

"Database": "db1",

"IsMapUDTAsJSON\_ApplicableForOracle": false,

"IsMapUDTAsXML\_ApplicableForOracle": false,

"ExternalDllPathImplementingIProcessDataInterface\_PreProcessing": "",

"ExternalDllNameImplementingIProcessDataInterface\_PreProcessing": "",

"FullyQualifiedNameOfClassImplementingInterfaceIProcessDataInterface\_PreProcessing": "",

"ExternalDllPathImplementingIProcessDataInterface\_PostProcessing": "",

"ExternalDllNameImplementingIProcessDataInterface\_PostProcessing": "",

"FullyQualifiedNameOfClassImplementingInterfaceIProcessDataInterface\_PostProcessing": "",

"IsSendOutputViaEmailAlso": false,

"Mailer": "somemailername2",

"IsAllowSendingJSONInMail": false,

"IsCachingRequired": false,

"CacheDurationInSeconds": 0

}

}

**mailers.<environment>.json sample:**

{

"somemailername1": {

"SMTP\_SERVER": "smtp.server",

"SMTP\_PORT": 1234,

"IsEncrypted": false,

"IsImpersonationNeeded": false,

"RunAsUser": "systemuser1",

"From": "noreply@noreply.com",

"To": "noreply@noreply.com",

"CC": "noreply@noreply.com",

"BCC": "noreply@noreply.com",

"Subject": "some subject",

"Body": "some body",

"AttachmentName": "some attachment name : DD-MMM-YYYY",

"IsBodyHtml": false

},

"somemailername2": {

"SMTP\_SERVER": "smtp.server",

"SMTP\_PORT": 1234,

"IsEncrypted": false,

"IsImpersonationNeeded": false,

"RunAsUser": "systemuser1",

"From": "noreply@noreply.com",

"To": "noreply@noreply.com",

"CC": "noreply@noreply.com",

"BCC": "noreply@noreply.com",

"Subject": "some subject",

"Body": "some body",

"AttachmentName": "some attachment name : DD-MMM-YYYY",

"IsBodyHtml": false

}

}

**users.<environment>.json sample:**

{

"systemuser1": {

"RunAsDomain": "domain",

"RunAsUserName": "username",

"RunAsPassword": "password",

"IsRunAsPasswordEncrypted": true,

"EncryptionKey": "some base 64 encryption key"

},

"systemuser2": {

"RunAsDomain": "domain",

"RunAsUserName": "username",

"RunAsPassword": "password",

"IsRunAsPasswordEncrypted": true,

"EncryptionKey": "some base 64 encryption key"

}

}

**Program.cs:**

CreateDefaultBuilderAndTinyLogger(args).ConfigureWebHostDefaults(wb =>

{

wb.UseStartup<Startup>();

}).Build().Run();

**Startup.cs**

public class Startup

{

public void ConfigureServices(IServiceCollection services)

{

'ConfigureServices'

\_ = services.AddTinyWebApi(services.BuildServiceProvider().GetRequiredService<IConfiguration>(), services.BuildServiceProvider().GetRequiredService<IWebHostEnvironment>(), services.BuildServiceProvider().GetRequiredService<ILoggerFactory>(), new TinyWebApiConfigurations()

'ConfigureServices'

{

ConfigurationDirectoryPath = new FileInfo(System.Reflection.Assembly.GetExecutingAssembly().Location).DirectoryName,

ConnectionStringJSONFileNameWithoutExtension = "connectionstring",

MailerJSONFileNameWithoutExtension = "mailer",

QueriesJSONFileNameWithoutExtension = "queries",

RunAsUserJSONFileNameWithoutExtension = "users",

DatabaseSpecifications = new(),

MailerSpecifications = new(),

QuerySpecifications = new(),

RunAsUserSpecifications = new()

});

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

app.UseTinyWebApi(env);

}

}