**newman** the cli companion for postman. export your Postman Collection as a json file from the Postman App and run it using Newman.

Go to cmd => newman run collection.json -e environment.json --global-var api-key="key123abc" --global-var hostUrl=<http://test.myapi.com/>

**NPM Behind a corporate Proxy Server**:

$ npm config set proxy http://proxy-server-url:<port>

$ npm config set https-proxy [http://proxy-server-url:<port](http://proxy-server-url:%3cport)>

**With user/password**:

npm config set proxy http://NK2774:password@proxy.dmz.ige:8080

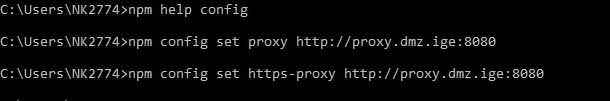
npm config set https-proxy [http://NK2774: password@proxy.dmz.ige:8080](http://NK2774:%20password@proxy.dmz.ige:8080)

npm config set strict-ssl=false

npm config set proxy http://<username>:<password>@<proxy-server-url>:<port>

npm config set https-proxy http://<username>:<password>@<proxy-server-url>:<port>

npm config set proxy "http://domain\username:password@servername:port/"

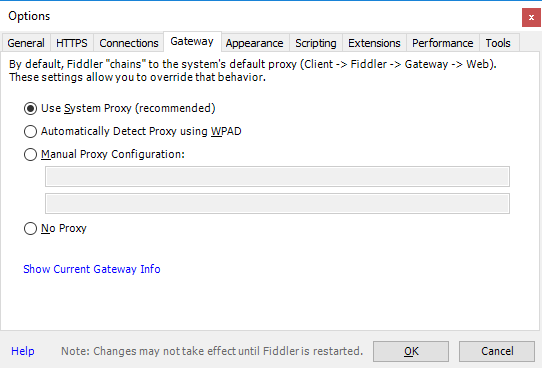


**Git**: git config --global http.proxy <http://mydomain\\myusername:mypassword@myproxyserver:8080>

git config --global http.proxy <http://proxy.mycompany:80>

To find proxy use fiddler and type: http://wpad/**wpad.dat** in any browser to open wpad.dat file.

Click on current Gateway info



The ASP.NET MVC framework supports 5 different types of filters:

* Authentication Filters: Authentication filter runs before any other filter or action method
* Authorization filters – Implements the IAuthorizationFilter attribute.
* Action filters – Implements the IActionFilter attribute.
* Result filters – Implements the IResultFilter attribute.
* Exception filters – Implements the IExceptionFilter attribute.

Filters are executed in the order listed above. For example, authorization filters are always executed before action filters and exception filters are always executed after every other type of filter.

**rewindable stream (use middleware):**

If you're using application/x-www-form-urlencoded or multipart/form-data, you can safely call context.Request.ReadFormAsync() multiple times as it returns a cached instance on subsequent calls.

If you're using a different content type, you'll have to manually buffer the request and replace the request body by a rewindable stream like MemoryStream.

do using an inline middleware (you need to register it soon in your pipeline):

<https://stackoverflow.com/questions/31389781/read-request-body-twice>

--- **Generate DbContext from command**

Scaffolding DbContext and Models with EntityFramework Core 2.0 and the CLI:

Put following in a scaffoldDbcontext.cmd file.

dotnet ef dbcontext scaffold "Data Source=local;Database=MyDb;Integrated Security=True" Microsoft.EntityFrameworkCore.SqlServer --context MyDBContext --context-dir . --output-dir ..\MyProj.Domain --schema dbo --schema Audit --force

--- End DbContext usng command

used \_ in parameter names

------------------ API Headers ----

//Put in controller

[FromHeader(Name = "header-name-xxx")]

public string HeaderNameXxx { get; set; }

//For response headers

HttpContext.Response.Headers["header -response-id"] = " some value 111";

===>------------ End API Headers ------------

---------- For swagger headers

public void Apply(Operation operation, OperationFilterContext context)

{

operation.Parameters.Add(new NonBodyParameter

{

Name = "X-MYHEADER",

In = "header",

Type = "string($date-time)",

Required = false

});

operation.Parameters.Add(new NonBodyParameter

{

Name = "X-MYHEADER\_OtherOne",

In = "header",

Type = "integer",

Required = true

});

//if (operation.Parameters == null)

// operation.Parameters = new List<Parameter>();

//operation.Parameters.Add(new Parameter

//{

// name = "X-User-Token",

// @in = "header",

// type = "string",

// required = false

//});

}

--- End swagger headers

PUT vs PATCH vs JSON-PATCH

PUT method only allows a complete replacement of an existing resource.

PATCh is for updating one or few fields at a time or say partial updates.

performs partial resource update similar to JSON Patch: https://github.com/Morcatko/Morcatko.AspNetCore.JsonMergePatch

Install Morcatko.AspNetCore.JsonMergePatch nugget

Why use **Azure Key Vault**?

benefits of key management, of being able to easily rotate as have two keys before deleting/creating new switch to 2nd, change, audit key access, outside from my application

**Middleware** sits in between client and server pipe line, and has the ability to inspect all incoming requests and outgoing responses, and if necessary, return a custom response.

Middleware act on every single request that comes in to the application.

MVC filters on the other hand only run for requests that come to MVC. So for example, if I wanted to enforce that all requests must be done over HTTPS, I would have to use a middleware for that.

Am I concerned about errors in the asp.net framework - use middleware.

Am I concerned only about errors in my controller and actions code - use filter.

do I need to capture the global errors in the asp.net framework as well as I have unique logic based on where the error occurred - use both.

CommonHttpResponseHandler

Shared

The order that middleware components are added in the Startup.Configure

Startup.Configure method adds **middleware** components for common app scenarios:

1. Exception/error handling
2. HTTP Strict Transport Security Protocol
3. HTTPS redirection
4. Static file server
5. Cookie policy enforcement
6. Authentication
7. Session
8. MVC

------------ HttpContextAccessor vs HttpContext

HttpContext:

HttpContext has had a bit of a shifting around in ASP.net core.

Inside a controller, you can still access HttpContext by doing the following :

var myUser = HttpContext.User;

Inside services is a little tricker (by using HttpContextAccessor)

First in your startup.cs, you need to register **IHttpContextAccessor** as a service like so :

services.AddMvc();

services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();

When you create a helper/service class, you can then inject in the IHttpContextAccessor and use it. It would look like something not too dissimilar to this :

var context = \_httpContextAccessor.HttpContext;

return context.User.Identities.Any(x => x.IsAuthenticated);

public class UserService : IUserService

{

private readonly IHttpContextAccessor \_httpContextAccessor;

public UserService(IHttpContextAccessor httpContextAccessor) {

\_httpContextAccessor = httpContextAccessor;

}

public bool IsUserLoggedIn() {

var context = \_httpContextAccessor.HttpContext;

return context.User.Identities.Any(x => x.IsAuthenticated);

}

}

---- End HttpContext

-------- SQL ------------

------ **Get column names from a table in SQL Server**?

SELECT \* FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME = Table123'

and COLUMN\_NAME like '%lo%'

--- **Script to write data**

SELECT

'(''' + [int\_XYZ\_Field] + ''','

+ '''' + [Char\_XYZ\_Field] + ''''

+ ', ' + ISNULL('''' + CONVERT(VARCHAR(50), CAST([Date\_XXXXX\_Field] AS DATETIME), 121 ) + '''', 'NULL') + ''

+ ', ' + ISNULL('''' + CONVERT(VARCHAR(210), CAST([char\_XYX\_Name] AS VARCHAR(210) )) + '''', 'NULL') + '' --=> Nullable char

+'),'

FROM [dbo].[SomeTable]

-- REPLACE([char\_XYX\_Name], '''', '''''' )

------End script to write data

----------- End SQL --------

Transforming the Collection of Student Information (TCSI)

Project Overview: The TSCI project team are tasked with transforming how student information is collected, improve administration and data collection activities for education providers and ensure student claims are simpler and more accurate. TCSI will replace the Higher Education Provider Client Assistance Tool (HEPCAT) for all education providers and the Centrelink Academic Reassessment Transformation (CART) for universities. TCSI is a joint project between DHS, DET and education providers.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\* React \*\*\*\*\*\*\*\*\*\*\***

playground for JavaScript and React: <https://jscomplete.com/repl>

**Reactjs**: is for building user interfaces. Complex React applications require the use of additional libraries for state management, routing, and interaction with an API.

Using React, writing UI test cases become extremely easy.

Components are the building blocks of a React application’s UI. These components split up the entire UI into small independent and reusable pieces. Then it renders each of these components independent of each other without affecting the rest of the UI. Components can contain other components.

**render()** in React: Each React component must have a render() mandatorily. It returns a single React element which is the representation of the native DOM component. If more than one HTML element needs to be rendered, then they must be grouped together inside one enclosing tag such as <form>, <group>,<div> etc. This function must be kept pure i.e., it must return the same result each time it is invoked.

We can embed two or more components into one class by **extends** key word.

With Reactive we write html in Java script. In other liberties like jQuery, angular enhances html.

**Virtual DOM:** uses the virtual DOM instead of the real DOM.

virtual DOM is a lightweight JavaScript object which originally is just the copy of the real DOM. It is a node tree that lists the elements, their attributes and content as Objects and their properties. React’s render function creates a node tree out of the React components.

This Virtual DOM works in three simple steps.

Whenever any underlying data changes, the entire UI is re-rendered in Virtual DOM representation.

Then the difference between the previous DOM representation and the new one is calculated.

Once the calculations are done, the real DOM will be updated with only the things that have actually changed.

React follows uni-directional data flow or one way data binding.

**Props** is the shorthand for Properties in React. They are read-only components which must be kept pure i.e. immutable. They are always passed down from the parent to the child components throughout the application. A child component can never send a prop back to the parent component. This help in maintaining the unidirectional data flow and are generally used to render the dynamically generated data.

**States** are the heart of React components. States are the source of data and must be kept as simple as possible. Basically, states are the objects which determine components rendering and behaviour. They are mutable unlike the props and create dynamic and interactive components. They are accessed via this.state().

**Class and Function components:** Function is simple one while Class component is more featured which holds its private internal state along with props.

We need other liberties for a complete solution.

**JSX** is a shorthand for JavaScript XML. JSX is XML-like (HTML like template) syntax but JSX is optional. To enable a browser to read JSX, first, we need to transform JSX file into a JavaScript object using JSX transformers like Babel and then pass it to the browser.

We can modularize code by using the export and import properties. They help in writing the components separately in different files.

We can use playground like CodePen or CodeSandbox.

**React Router** is a powerful routing library built on top of React.

**Flux** is an architectural pattern which enforces the uni-directional data flow. It controls derived data and enables communication between multiple components using a central Store which has authority for all data. Any update in data throughout the application must occur here only. Flux provides stability to the application and reduces run-time errors.

**Redux** is libraries for front-end development. It is a **predictable state container** for JavaScript applications and is **used for the entire applications state management**. Applications developed with Redux are easy to test.

Redux uses ‘**Store’** for storing the application’s entire state at one place. So all the component’s state are stored in the Store and they receive updates from the Store itself. The single state tree makes it easier to keep track of changes over time and debug or inspect the application.

Redux is composed of the following components:

Action – It’s an object that describes what happened.

Reducer – It is a place to determine how the state will change.

Store – State/ Object tree of the entire application is saved in the Store.

View – Simply displays the data provided by the Store.

------------ JSON

When receiving data from a web server, the data is always a string.

Parse the data with JSON.parse(), and the data becomes a JavaScript object.

When sending data to a web server, the data has to be a string.

Convert a JavaScript object into a string with JSON.stringify().

Integrated Tertiary Systems Platform (ITSP) to support the Transformation of the Collection of Student Infromation (TCSI) project

**AutoRest** tool generates client libraries for accessing RESTful web services using OpenAPI Specification format as Input.

**Extension methods** are defined as static methods but are called by using instance method syntax. Their first parameter specifies which type the method operates on, and the parameter is preceded by the this modifier. Extension methods are only in scope when you explicitly import the namespace into your source code with a using directive.

The following example shows an extension method defined for the System.String class.

public static class MyExtensions {

public static int WordCount(**this String str**) {

return str.Split(new char[] { ' ', '.', '?' },StringSplitOptions.RemoveEmptyEntries).Length;

} }

Exceptions:

source = source ?? throw new ArgumentNullException(nameof(source));

++========\*\*\*\*\*\*\*=======+++++++++

Transforming the Collection of Student Information (TCSI): RESTful APIs for managing Higher Education Providers and Staff, Students, Applications and Offers, Curriculum, Admissions and Enrolments, Loans and Payments, References and Errors.

API’s for Course of Study, Courses, Campuses, Course on Campus and Units of Study

Swagger/OpenAPI Specification

Microsoft Azure, Visual Studio 2017, ASP.NET Core 2, Azure SQL Server, Ms SQL Server 2016, Entity Framework Core 2, Xunit, Moq, Integration Testing, Swashbuckle, DevOps continuous integration and deployment, Git, Postman, Newman, PowerShell.

Katalon Studio, ARM, APIM, Visual Studio Code, Service Fabric, AutoRest (<https://github.com/Azure/autorest/blob/master/docs/user/cli.md> )

XXXXXXXX------------+++++++++++++++++++++++++++

**SQL Server Analysis Services** is an analytical data engine used in decision support and business analytics.

It provides data models for business reports and client applications such as Power BI, Excel, Reporting Services reports.

+++++++\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**camelCase/JavaScript** naming conventions: helloWorld

**Snake case**: hello\_world, Hello\_World

\_\_\_+++++++++++

**SQL Bulk Copy Utility**:

To check the BCP version open Command Prompt, execute bcp /v command and confirm that 15.0.1000.34 or higher is in use.

The bulk copy program utility (bcp) bulk copies data between an instance of Microsoft SQL Server and a data file in a user-specified format. The bcp utility can be used to import large numbers of new rows into SQL Server tables or to export data out of tables into data files.

The utility requires no knowledge of Transact-SQL.

It is faster than the merge statement. It doesn't have the memory issues that merge has. The insert into values has a limit of 1000 records per statement

-+++++++++++++

**Entity Framework : Create/Update common field in multiple tables**:

LastModifiedBy - Logged In User Name

LastModifiedDate - Today's date and time

<https://stackoverflow.com/questions/38618418/entity-framework-create-update-common-field-in-multiple-tables>

<https://benjii.me/2014/03/track-created-and-modified-fields-automatically-with-entity-framework-code-first/>

foreach (var entry in context.ObjectStateManager

.GetObjectStateEntries(EntityState.Added | EntityState.Modified))

{

}